Actes des 17es Journées Francophones de Programmation par Contraintes

Nicolas Prcovic

AfIA

Association française
pour l'Intelligence Artificielle

JFPC

Journées Francophones
de Programmation par Contraintes

PFIA 2022

Table des matières

Éditorial

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 dix-septièmes Journées Francophones de Programmation par Contraintes ont eu lieu du 27 au 29 juin 2022 à Saint-Etienne pour la première fois dans le cadre de la Plate-Forme Intelligence Artificielle (PFIA'22). 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, vingt-six soumissions (douze articles longs, trois articles courts et onze résumés d'articles déjà publiés dans une conférence internationale de renom l'an passé) ont été relues, toutes ont été acceptées (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 négative attribuée à aucun papier (ce qui a grandement simplifié le travail du président du comité de programme, aucune discussion sur l'acceptation d'un papier n'ayant donc eu besoin d'avoir lieu).

En plus des présentations orales des articles que vous retrouverez dans ces actes, nous avons eu pendant la conférence trois exposés invités : celui de Tias Guns "Learning and Reasoning with Constraint Solving", celui de Pierre Deransart "50 ans de Prolog, et après ?" dans le cadre de la célébration des 50 ans de Prolog (session commune avec CNIA) et celui de Chu-Min Li "Intégration du raisonnement SAT dans la résolution des problèmes difficiles".

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.

Nicolas PRCOVIC

Comité de programme

Président

— Nicolas Prcovic, LIS, Université d'Aix-Marseille.

Membres

— Quentin Cappart, Ecole Polytechnique de Montréal ;
— Clément Carbonnel, CNRS ;
— Mohamed Sami Cherif, Aix-Marseille Université, LIS ;
— Thi-Bich-Hanh Dao, Université d'Orléans ;
— Sophie Demassey, CMA, MINES ParisTech ;
— Guillaume Derval, Université Catholique de Louvain ;
— Jean-Guillaume Fages, COSLING S.A.S. ;
— Xavier Gillard, Université Catholique de Louvain ;
— Jin-Kao Hao, Université d'Angers ;
— Arnaud Lallouet, Huawei Technologies Ltd ;
— Olivier Lhomme, IBM France ;
— Chu-Min Li, Université de Picardie Jules Verne ;
— Giovanni Lo Bianco, Université de Toronto ;
— Pierre Montalbano, ANITI/INRAE ;
— Margaux Nattaf, Grenoble INP - G-SCOP ;
— Samba Ndojh Ndiaye, Liris ;
— Bertrand Neveu, Ecole des ponts ;
— Abdelkader Ouali, GREYC, Université de Caen Normandie ;
— Marie Pelleau, Université Côte d'Azur, I3S ;
— Guillaume Perez, Université Côte d'Azur, I3S ;
— Eric Piette, DKE ;
— Charles Prud'Homme, IMT Atlantique, LS2N ;
— Mathieu Py, LIS, Aix-Marseille Université ;
— Julien Vion, LAMIH, Université de Valenciennes et du Hainaut-Cambrésis ;
— Romain Wallon, CRIL, Université d'Artois ;
— Ghiles Ziat, LIP6.

Session 1 : Résolution de CSP

Échantillonnage de solutions pratiquement uniforme en programmation par contraintes

Gilles Pesant[1], Claude-Guy Quimper[2] and Hélène Verhaeghe[1]

[1]Polytechnique Montréal, Canada
gilles.pesant@polymtl.ca, helene.verhaeghe@polymtl.ca
[2]Université Laval, Canada
claude-guy.quimper@ift.ulaval.ca

Juin 2022

Résumé

La capacité d'échantillonner des solutions dans un espace de recherche combinatoire contraint a des applications dans des domaines tels que le raisonnement probabiliste et la vérification de matériel/logiciel. Une propriété souhaitable d'un tel échantillonnage est l'uniformité du tirage aléatoire parmi les solutions. Notre approche se base sur un hachage universel ayant des garanties sur l'uniformité du tirage. Nous résumons ici notre article "Practically Uniform Solution Sampling in Constraint Programming", accepté à CPAIOR2022 [3].

Mots-clés

Échantillonnage uniforme, hachage, dénombrement

Abstract

The ability to sample solutions of a constrained combinatorial space has important applications in areas such as probabilistic reasoning and hardware/software verification. A highly desirable property of such samples is that they should be drawn uniformly at random, or at least nearly so. Our approache approaches is based on universal hashing which provide probabilistic guarantees about sampling uniformity. We summarise here our paper "Practically Uniform Solution Sampling in Constraint Programming" accepted at CPAIOR2022 [3].

Keywords

Uniform sampling, haching

1 Introduction

De nombreux mais importants problèmes peuvent être exprimés sur un espace combinatoire contraint : trouver un élément *satisfaisant* (i.e. une solution) dans cet espace et chercher pour un *optimal* selon un critère sont probablement les tâches les plus communes. *Compter* combien de solutions il y a et *échantillonner* des solutions de manière uniforme aléatoirement ont également des applications importantes. Des exemples de telles tâches existent en planification, inférence Bayesienne, vérification de code, pour ne nommer que celles-ci.

Les compteurs de modèles qui suivent une approche basée sur le hachage universel bénéficient de garanties probabilisitiques concernant la qualité du dénombrement approximatif qu'ils donnent. Dans le contexte des modèles SAT, de tels hachages universels prennent la forme de contraintes de parité (XOR) générées de manière aléatoire [1]. Dans un contexte plus large de variables avec domaine fini et de modèles PPC, ces contraintes se généralisent par des contraintes d'égalités linéaires en arithmétique modulaire générées de manière aléatoire, telles qu'investiguées dans [2].

Récemment, un échantillonneur de solutions pour la PPC [4] s'inspire de l'idée de découpe de l'espace de recherche en cellules et se concentre sur une telle cellule. Cette découpe est faite en ajoutant des contraintes de tables aléatoires de faible arité au modèle PPC. Malheureusement cette méthode n'est ni supportée par des garanties théoriques garantissant la qualité de l'échantillonnage ni n'a montré un échantillonnage uniforme durant les expériences.

Notre méthode ajoute certaines contraintes linéaires modulaires, partageant les mêmes garanties probabiliste que les contraintes XOR pour SAT, afin d'échantillonner les solutions d'un modèle PPC. En pratique, nos expériences montrent que nous arrivons à un échantillonnage presque uniforme. Notre méthode est indépendante du modèle échantillonné et peut être appliquée tant que le solveur PPC supporte les contraintes requises.

2 Échantillonnage uniforme

Notre but est d'utiliser des fonctions de hachage indépendantes les unes des autres pour diviser l'espace de solutions en cellules contenant environ le même nombre de solutions chacune. L'échantillonnage se fait par énumération des solutions d'une cellule de la taille choisie. Un des avantages de telles cellules obtenues par hachage de l'espace est que l'énumération des solutions contenues est plus rapide dû à un espace plus petit. Un autre avantage est la garantie de ne pas avoir de doublon au sein d'une même cellule.

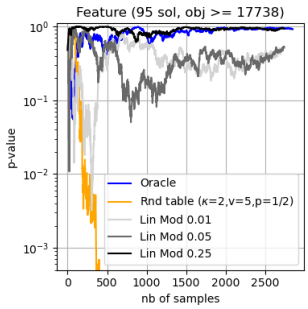


Figure 1: Problème des modèles de caractéristique avec contrainte additionnelle sur l'objectif ($obj \geq 17738$)

Le partitionnement presque uniforme des solutions peut être atteint en ajoutant le système de m inégalités linéaires modulaires sur n variables avec domaine entier fini

$$A\boldsymbol{x} \leq \boldsymbol{b} \pmod p$$

où $\boldsymbol{x}$ est le vecteur des n variables avec domaine entier fini, A est une matrice $m \times n$ dont les éléments appartiennent à $[p] \triangleq \{0, 1, \ldots, p-1\}$, $\boldsymbol{b}$ est un vecteur de m éléments de $[p]$, et p est le modulo. Les inégalités linéaires modulaires possèdent 2 propriétés si le modulo p est un nombre premier. Premièrement, un partitionnement uniforme des solutions, et, secondement, l'indépendance garantie entre deux solutions. En utilisant un nombre premier, on peut également appliquer une élimination de Gauss-Jordan sur le système d'inégalités linéaires afin de le réécrire sous forme paramétrique. Son ensemble de solutions, encodable comme une contrainte de table, peut être obtenu par simple énumération sur les domaines des variables. Cela permet d'arriver à la consistance des domaines par une contrainte de table ajoutée au modèle. La proportion de solutions (ou taille de la cellule) est contrôlé par le nombre m d'inégalités linéaires modulaires ajoutées ainsi que par le choix des les valeurs dans $\boldsymbol{b}$.

3 Expériences

Durant nos expériences, nous avons testé notre méthode sur quatre problèmes: le problème des N-Reines, le problème des modèles de caractéristique (problème de configuration de logiciel), un modèle synthétique_n_d (créé avec n variables de taille de domaine d, de telle sorte que le nombre total de solutions soit calculable analytiquement) et le problème Myciel (coloriage de graphe). Pour vérifier l'uniformité de l'échantillonnage sur les examplaires avec un nombre raisonnable de solutions, nos tests échantillonnent 30 fois le nombre de solutions. Des cellules contenant 1%, 5% et 25% des solutions sont utilisées pour effectuer l'échantillonnage. Pour les examplaires avec plus de solutions, une centaine de solutions sont échantillonnées via des cellules contenant 0.001% de solutions.
La qualité de l'échantillonnage est analysée en calculant l'évolution de la p-valeur. La figure 1 montre l'évolution de la p-valeur à chaque solution échantillonnée en plus sur l'examplaire du problème des modèles de caractéristiques auquel une contrainte à été ajoutée sur la valeur de l'objectif (afin de réduire le nombre de solution à 95). Plus celle-ci est proche de 1, plus l'échantillonnage est uniforme. Nous nous comparons à un oracle (ligne bleue) qui suppose toutes les solutions connues et qui tire une solution au hasard à chaque fois (basé sur le générateur aléatoire de `java`). Nous nous comparons également à la méthode de Vavrille et al. [4] (ligne orange). Nous pouvons voir sur le graphe que notre méthode obtient un échantillonnage le plus uniforme parmi les méthodes. L'uniformité est également meilleure plus les cellules considérées sont grandes, ce qui est attendu puisque les solutions d'une même cellule sont garanties distinctes. Des résultats similaires sont obtenus sur d'autre problèmes.
D'un point de vue du temps d'exécution, pour les examplaires avec beaucoup de solutions, pour une énumération d'une cellule contenant 0.005% des solutions, nous avons pu observer que notre méthode est beaucoup plus rapide que d'énumérer l'entièreté des solutions pour ensuite sélectionner le nombre de solutions équivalent au contenu de la cellule. Nous obtenons également une bien meilleure uniformité que [4].

4 Conclusion

Notre papier décrit une nouvelle approche pour échantillonner de manière uniforme un ensemble de solutions d'un modèle de PPC. Les résultats empiriques démontrent que notre méthode atteint presque l'uniformité.

References

[1] Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. A scalable approximate model counter. In *Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*, pages 200–216. Springer, 2013.

[2] Gilles Pesant, Kuldeep S. Meel, and Mahshid Mohammadalitajrishi. On the usefulness of linear modular arithmetic in constraint programming. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 18th International Conference, CPAIOR 2021, Vienna, Austria, July 5-8, 2021, Proceedings*, pages 248–265. Springer, 2021.

[3] Gilles Pesant, Claude-Guy Quimper, and Hélène Verhaeghe. Practically uniform solution sampling in constraint programming. In *CPAIOR22*, 2022.

[4] Mathieu Vavrille, Charlotte Truchet, and Charles Prud'homme. Solution sampling with random table constraints. In *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021*, pages 56:1–56:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

Résolution de problèmes dynamiques à l'aide du Model Checking

Florian Régin, Elisabetta De Maria, Marie Pelleau

Université Côte d'Azur, CNRS, I3S, France

[prénom.nom]@univ-cotedazur.fr

Résumé

Dans cet article nous introduisons une approche basée sur le Model-Checking pour résoudre des problèmes combinatoires dynamiques. Le Model-Checking est une technique qui permet de tester automatiquement si un système vérifie des propriétés qui concernent son évolution dynamique. Étant donné un problème de programmation par contraintes dynamique, la stratégie proposée consiste à exploiter des appels à un model checker pour couper assez tôt des branches infructueuses de l'arbre de recherche. Nous illustrons notre approche avec un exemple basé sur des feux de circulation.

Mots-clés

Programmation par Contraintes, Système Dynamique, Logique Temporelle, Model-Checking à la Volée.

Abstract

In this paper we introduce an approach based on Model-Checking to solve dynamic combinatorial problems. Model-Checking is a technique that allows to automatically test if a system verifies some properties concerning its dynamic evolution. Given a dynamic constraint programming problem, the proposed strategy consists in exploiting some calls to a model checker to cut off unsuccessful branches of the search tree quite early. We illustrate our approach with an example based on traffic lights.

Keywords

Constraint Programming, Temporal Logic, On-the-fly Model-Checking.

1 Introduction

Dans ce papier nous présentons une nouvelle approche d'utilisation de Model-Checking en Programmation par Contraintes. Plus précisément, le but de ce travail est de combiner le Model-Checking et la Programmation par Contraintes afin d'optimiser la recherche de solutions dans des problèmes combinatoires dynamiques. Le Model-Checking est une technique formelle qui permet de tester automatiquement des propriétés qui concernent l'évolution dynamique d'un système [Clarke et al., 1999]. L'idée est de faire appel à un outil de Model-Checking pour vérifier des propriétés de logique temporelle dont les réponses permettraient de couper assez tôt des branches de l'arbre de recherche qui n'amènent à aucune solution. Le Model-Checking est donc utilisé comme un oracle pour guider la recherche. Pour combiner un solveur de Programmation par Contraintes avec un model checker nous avons choisi de travailler dans la partie exploration afin d'intervenir directement quand le solveur prend une décision. Nous illustrons notre stratégie avec une variante du problème des feux de la circulation. Nous modélisons le problème choisi à la fois avec le solveur Choco [Prud'homme et al., 2016], qui est un logiciel libre développé en Java, et avec le langage Promela mis à disposition par le model checker Spin [Holzmann, 2003]. Les propriétés dynamiques d'intérêt sont formalisées en utilisant la logique temporelle LTL (Linear Time Logic).

Il existe de nombreux travaux sur la combinaison du Model-Checking avec la Programmation par Contraintes mais ces travaux ont principalement comme but d'utiliser la Programmation par Contraintes pour améliorer le Model-Checking. Dans [Bart, 2017] l'auteur présente une méthode qui utilise la Programmation par Contraintes pour résoudre des problèmes du domaine de la vérification (domaine du Model-Checking). Il introduit une nouvelle contrainte globale et de nouveaux modèles pour traiter les problèmes de vérification probabiliste. Une méthode qui combine le Bounded-Model-Checking et la Programmation par Contraintes pour l'aide à la localisation d'erreurs est proposée dans [Bekkouche, 2017]. La méthode consiste à appliquer la Programmation par Contraintes sur le contre-exemple fourni par le model checker pour obtenir l'ensemble des MUS (Minimal Unsatisfiable Subset) du programme donné en entrée. Dans [Chan et al., 1997] la Programmation par Contraintes est utilisée pour réduire la taille des BDD (Binary Decision Diagrams) dans le cadre du Model-Checking symbolique. Dans [Collavizza et al., 2008] les auteurs présentent l'outil Constraint-Programming Framework for Bounded Program Verification (CPBPV). Cet outil utilise des techniques de Programmation Par Contrainte pour vérifier des programmes. Dans [Esparza and Melzer, 1997] les auteurs introduisent une méthode qui consiste à utiliser la Programmation par Contraintes pour éviter l'explosion des états avec la logique LTL pour de problèmes particuliers. L'auteur de [Flanagan, 2003] présente un model checker qui utilise la Programmation par Contraintes pour représenter un modèle, dans le but d'éviter l'explosion des états et de pouvoir utiliser des techniques de Programmation par Contraintes en Model-Checking. En-

fin dans [Nilsson and Lübcke, 2000] les auteurs présentent une méthode qui consiste à utiliser la Programmation par Contraintes pour représenter un problème avec la logique temporelle CTL dans le cas du Model-Checking Symbolique et du Model-Checking à la volée, pour éviter l'explosion des états.
En Programmation par Contraintes nous raisonnons avec des CSP (Constraint Satisfaction Problem), en Programmation par Contraintes Dynamique nous raisonnons avec des CSP dynamiques, des DCSP. Un DCSP est une séquence de CSP, chaque CSP est une transformation d'un CSP antérieur, dû à un ajout/retrait d'une variable ou un changement de domaine. Tous ces changements peuvent être exprimés sous forme d'additions ou de retraits (relaxations) de contraintes sur les différents CSP. Un DCSP peut être considéré comme une sorte de décomposition de problème en différent sous-problèmes qui forment une séquence.
L'article est organisé comme suit. Dans la Section 2 nous introduisons le Model-Checking (concepts de système de transition et logique temporelle) et le model checker Spin. Dans la Section 3 nous présentons le cas d'étude et son encodage sous forme de problème de Programmation par Contraintes dynamique et de système de transition. La Section 4 est consacrée à nos premiers résultats expérimentaux. Dans la Section 5 nous décrivons les développements que nous envisageons pour ce travail.

2 Model-Checking

Le Model-Checking est une approche de vérification formelle [Clarke et al., 1999]. Il s'agit d'une technique qui permet de vérifier si un système réactif respecte une ou plusieurs propriétés dynamiques. Un système réactif se caractérise par les trois points suivants : il ne termine pas nécessairement, il ne calcule pas un résultat mais plutôt maintient une interaction entre les états de l'environnement, les types de données des états sont en général simples (types de données numériques) alors que le contrôle est complexe (*e.g.,* exécution en parallèle de plusieurs états).
En Model-Checking les systèmes sont modélisés sous forme de systèmes de transition (structures de Kripke) et les propriétés sont spécifiées à l'aide de formules de logique temporelle. Une structure de Kripke est un graphe orienté dont les nœuds représentent les états accessibles du système et dont les arcs représentent les transitions entre les états. Chaque état est étiqueté avec les propositions atomiques qui sont vraies dans l'état.

Definition 1. *Un système de transition portant sur un ensemble AP de propositions atomiques est un n-uplet M=(Q, T, L), où Q est un ensemble fini d'états, $T \subseteq Q \times Q$ est une relation totale de transition (c'est-à-dire, pour tout état $e \in Q$ il existe un état $e' \in Q$ tel que $T(e, e')$), et $L : Q \to 2^{AP}$ est une fonction d'étiquetage qui associe à chaque état l'ensemble de propositions atomiques qui sont vraies dans l'état.*

La Figure 1 donne un exemple de système de transition pour deux feux de circulation qui se trouvent dans la même intersection. Ce système porte sur six propositions atomiques :

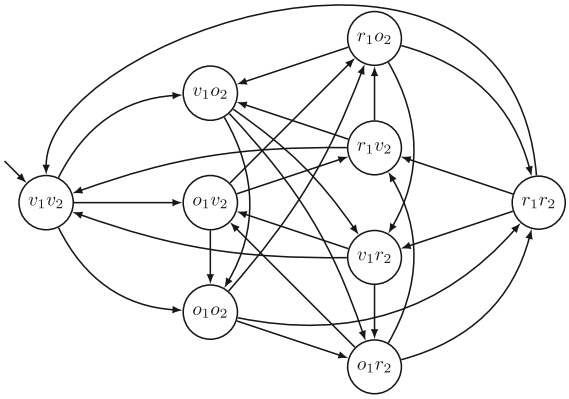


FIGURE 1 – Exemple de système de transition qui modélise deux feux de circulation.

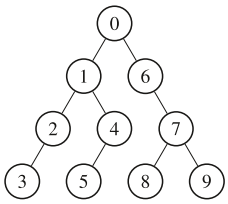
$\{v_1, v_2, o_1, o_2, r_1, r_2\}$, où pour $i \in \{1, 2\}$ la proposition v_i (r_i, o_i, respectivement) est vraie dans un état si le feu i est vert (rouge, orange, respectivement). Les deux feux sont initialisé à vert (pour cela, l'état initial est celui étiqueté par les propositions v_1 et v_2). A chaque unité de temps chacun des deux feux a la possibilité de passer à la couleur suivante ou bien de garder la couleur courante. Chaque état du système possède un auto-boucle (les auto-boucles n'ont pas été inclues dans la figure pour avoir plus de lisibilité). Les chemins des systèmes de transition sont infinis, le système de transition de cet exemple n'a donc pas d'état final.
Il est à noter qu'il est toujours possible d'obtenir un arbre de calcul à partir d'une structure de Kripke en dépliant la structure à partir de l'état initial.

2.1 Logique temporelle

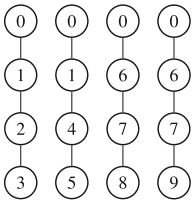
Les formules de logique temporelle permettent d'exprimer des propriétés liées à l'évolution dynamique d'un système [Clarke et al., 1986]. La logique temporelle CTL* englobe les deux logiques les plus connues, CTL (Computation Tree Logic) et LTL (Linear Temporal Logic).
En CTL* les formules sont composées de propositions atomiques, de connecteurs Booléens et d'opérateurs temporels spécifiques : les *opérateurs de chemin* et les *opérateurs d'état*. Il y a deux opérateurs de chemin, **A** ("pour tous les chemins") et **E** ("il existe un chemin"). Ces opérateurs sont utilisés sur un état pour préciser si tous les chemins ou au moins un chemin commençant par cet état doivent respecter une propriété. Les opérateurs d'état décrivent les propriétés d'un chemin, il en existe quatre :

- **X**P ("next") spécifie que la propriété P est vraie dans le prochain état du chemin.
- **F**P ("in the future") spécifie que la propriété P est vraie dans un futur état du chemin.
- **G**P ("globally") spécifie que la propriété P est vraie dans tous les états du chemin.
- p**U**q ("P until Q) spécifie que la propriété Q est vraie à un état du chemin et dans tous les états précédents du chemin la propriété P est vraie.

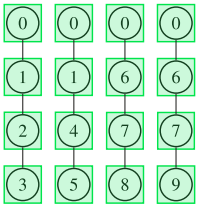
La distinction entre les logiques CTL et LTL réside dans



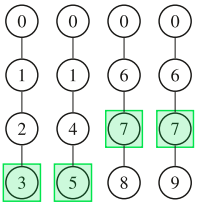
(a) Exemple d'arbre de calcul.



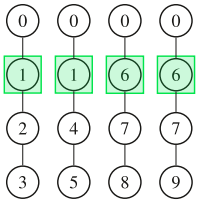
(b) Chemins de l'arbre de la Figure 2a.



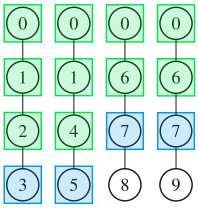
(c) Formule AGp.



(d) Formule AFp.



(e) Formule AXp.



(f) Formule $ApUq$.

FIGURE 2 – Exemples de formules en logique temporelle

la façon dont elle gèrent l'arbre de calcul sous-jacent. En CTL les opérateurs temporels parlent des chemins qui sont possibles à partir d'un état donné. En LTL les opérateurs décrivent des événements le long d'un seul chemin de calcul. CTL est restreinte au sous-ensemble de CTL* où chaque opérateur d'état est précédé immédiatement par un opérateur de chemin. D'autre part, LTL se compose de formules de la forme $A\phi$, où ϕ est une formule qui ne peut pas contenir des opérateurs de chemin. LTL est la logique que nous avons choisi pour ce travail.

Pour illustrer les formules de la logique temporelle, considérons un fragment d'arbre de calcul qui se compose de 10 états, comme dessiné dans la Figure 2a (les différents chemins de l'arbre, utiles pour évaluer les formules de LTL, sont détailles dans la Figure 2b). Soient p et q deux propositions atomiques. Dans les Figures 2c, 2d, 2e et 2f, les états qui satisfont p sont coloriés en vert et les états qui satisfont q sont colorié en bleu. Dans la Figure 2c les états ont été coloriés de telle sorte à ce que la formule AGp soit satisfaite (pour chaque chemin, la proposition p est toujours vraie). La Figure 2d illustre un exemple de satisfaction de la formule AFp (pour chaque chemin, il existe un état où p est vraie). Dans la Figure 2e les états ont été coloriés de telle sorte à ce que la formule AXp soit satisfaite (pour chaque chemin, la proposition p est vraie dans le deuxième état du chemin). La Figure 2f illustre un exemple de satisfaction de la formule $ApUq$ (pour chaque chemin, il existe un état où q est vraie et, pour tout état précèdent, p est vraie).

Lorsqu'un model checker prouve qu'une formule est

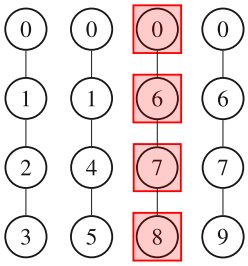


FIGURE 3 – Contre-exemple pour AGp.

fausse, il est capable de générer un contre-exemple, c'est-à-dire une trace d'exécution qui commence par l'état initial et qui atteste que le système ne respecte pas la propriété désirée. Reprenons en considération les chemins de la Figure 2b et la formule AGp. Supposons que cette formule ne soit pas vérifiée, c'est-à-dire, supposons qu'il existe un chemin avec un état où p est fausse. Si par exemple p n'est pas vraie à l'état 7, le model checker peut donner comme contre-exemple le chemin indiqué en rouge dans la Figure 3.

2.2 Model-Checking à la volée

Pour palier au problème de l'explosion des états, les deux approches les plus utilisées sont le Model-Checking à la volée [Holzmann, 1996] et le Model-Checking symbolique [Clarke et al., 1996]. Le Model-Checking à la volée consiste à construire uniquement la portion du système qui sert à prouver la formule. Le Model-Checking symbolique consiste à utiliser des OBDD (Ordered Binary Decision Diagram) pour compresser certains états et ainsi réduire la taille du système de transition.

Nous avons comparé NuSMV [Cimatti et al., 1999], le model checker le plus connu utilisant le Model-Checking symbolique, et Spin [Holzmann, 2003], le model checker le plus connu utilisant le Model-Checking à la volée. Ayant obtenu de bien meilleurs résultats avec Spin nous avons choisi de l'utiliser. Nous utilisons la logique temporelle LTL, car c'est la seule compatible avec Spin pour utiliser du Model-Checking à la volée.

Il existe une technique de Model-Checking basée sur PDR (Property Directed Reachability) [Welp and Kuehlmann, 2013, Vizel and Gurfinkel, 2014], qui partant d'états non valides essaye d'atteindre un état initial. Si un état initial est atteint, la propriété est donc fausse. Au contraire, si aucun état non valide n'arrive à atteindre un état initial la propriété est vraie. Afin d'utiliser cette technique, il faut pouvoir générer tous les états non valides. En fonction du nombre de feux et d'unités de temps le nombre d'état est exponentiel. Nous n'avons donc pas envisagé cette technique pour ce type de problème.

2.3 Spin

Spin [Holzmann, 2003] est un model checker crée par Gerard J. Holzmann et qui à été rendu public pour la première fois en 1991. Il utilise le langage Promela (Process Meta Langage) pour représenter le système de transition et LTL pour faire du Model-Checking à la volée. Promela est

FIGURE 4 – Exemple avec quatre feux.

un langage de spécification de systèmes, il permet la description de systèmes concurrents ainsi que la création dynamique de processus. La communication entre différents processus peut se faire en partageant les variables globales ou en utilisant des canaux de communication. Promela permet de simuler des communications synchrones ou asynchrones et d'accepter du code en langage C.

2.3.1 Automatisation

Pour utiliser Spin il faut :

- Créer un modèle en Promela.
- Utiliser Spin qui compile le modèle Promela en un fichier dans le langage C.
- Compiler le fichier C pour avoir un exécutable.
- Utiliser l'exécutable avec les options appropriées (*e.g.,* la formule LTL à tester, la mémoire utilisable, etc.).

Normalement entre deux utilisations de Spin, même si nous changeons uniquement la valeur de certaines variables, nous devons recréer le modèle Promela et recommencer le processus. Pour éviter de perdre du temps dans la compilation, nous créons une seule fois le fichier C et le modifions pour que l'exécutable prenne en paramètre la valeur des variables.
Après la première utilisation de Spin, pour chaque autre utilisation (du même modèle de base) nous utilisons uniquement l'exécutable avec la valeur des variables en paramètre.

3 Cas d'étude

Nous avons choisi comme cas d'étude le problème des feux de la circulation, qui peut être modélisé en Programmation par Contraintes Dynamique.
Nous avons un nombre n de feux, et un nombre T d'unités de temps. Les feux ont un état (vert, orange ou rouge). Les contraintes de transitions dans le temps varient selon le modèle, si un feu change d'état alors il passe de vert à orange, d'orange à rouge ou de rouge à vert. Chacun des n feux peut être instancié selon l'un des modèles suivants :

- Le modèle **classique**, où à chaque unité de temps chaque feu doit changer d'état.
- Le modèle **limité**, où à chaque unité de temps chaque feu peut changer d'état ou rester dans l'état courant (s'il est resté dans l'état courant un nombre de fois inférieur ou égal à une constante prédéfinie).

Étant donné n feux et $0 \leq k \leq n$, nous supposons qu'il y a k feux instanciés selon le modèle classique et $n - k$ feux selon le modèle limité.
La photo de la Figure 4 contient un exemple d'intersection avec quatre feux.

3.1 Modèle en Programmation par Contraintes

En Programmation par Contraintes nous représentons le problème de la façon suivante :

Les variables. Nous représentons les variables par la matrice f. $f_{i,t}$ correspond au feu numéro i au temps t. Le domaine des variables est $D = \{0, 1, 2\}$ (0 pour vert, 1 pour orange et 2 pour rouge).

Les contraintes de transitions dépendent du modèle :

- Modèle **classique**. $\forall i, \forall t, 0 < i \leq n, 0 < t \leq T$ nous voulons autoriser uniquement les combinaisons $\{0, 1\}, \{1, 2\}, \{2, 0\}$ pour $\{f_{i,t}, f_{i,t+1}\}$: $(f_{i,t} + 1)\%3 = f_{i,t+1}$
- Modèle **limité**. Nous combinons deux ensembles de contraintes. Le premier ensemble de contraintes interdit les transitions impossibles entre deux temps successifs (vert passe à rouge, orange passe à vert et rouge passe à orange). Le deuxième ensemble de contraintes va limiter le nombre de fois consécutive qu'une variable peut-être égale à la même valeur (pour éviter que le premier feu reste toujours à vert par exemple).
  1. $f_{i,t} \neq (f_{i,t+1} + 1)\%3$, pour les combinaisons interdites.
  2. $atLeastNValues$, pour éviter que les feux ne changent jamais de valeurs.

Nous allons avoir k feux qui respectent la contrainte du modèle **limité** et $n - k$ feux qui respectent la contrainte du modèle **classique**.

3.2 Modèle en Model-Checking

Nous représentons le système de transitions par des états et des transitions.
Les propositions atomiques du modèle sont les suivantes : f, où f_i représente le feu numéro i et $f_i \in \{vert, orange, rouge\}$. $sumVert$, $sumRouge$ et $sumOrange$ représentent la somme des feux verts, oranges et rouges respectivement. Le modèle **limité** utilise les valeurs du modèle **classique** et rajoute l'ensemble $stagne$, où $stagne_i$ est égal au nombre d'unités de temps écoulées depuis la dernière fois où le feu a changé de couleur.
L'état initial des modèles dépend de la propriété à tester, ce sera expliqué dans la Section 4. Les transitions des feux dépendent du modèle choisi.

3.3 Contraintes Temporelles

Nous avons choisi d'utiliser les contraintes temporelles suivantes pour notre cas d'étude :

1. À chaque temps, au moins 1 feu est vert.
2. Au dernier temps, au moins $n-1$ feux sont rouges.

En Programmation par Contraintes

— $AtLeast(1, f_{i,t}, 0)$ $\quad \forall t, 0 \leq t \leq T$
— $AtLeast(n-1, f_{i,T}, 2)$

En Model-Checking

Nous utilisons les variables $nbVert$ et $nbRouge$ qui comptent le nombre de feux à vert/rouge respectivement à l'état courant. Nous exprimons les contraintes temporelles à l'aide de la formule de logique temporelle suivante :

$$E((G(nbVert \geq 1) \wedge F(nbRouge \geq n-1))$$

Cette formule exprime l'existence d'un chemin tel que :

— $G(nbVert \geq 1)$: à chaque état il y a au moins un feu qui est vert.
— $F(nbRouge \geq n-1)$: il y a un état futur où au moins $n-1$ feux sont rouges.

Comme Spin accepte uniquement des formules de la forme $A\phi$, nous construisons la négation logique de la formule et considérons que la formule initiale est vraie si sa négation est fausse.

3.4 Stratégie d'intégration du model checker dans le solveur

La Figure 5 illustre notre approche. Nous travaillons pendant la recherche du solveur de Programmation par Contraintes, pour un temps donné t. Pour rappel, f_i correspond au feu i. Pendant la recherche le solveur va essayer d'affecter des valeurs aux variables, ici il essayera d'affecter les feux à vert, puis à orange et à rouge. Le solveur affecte une valeur au premier feu, puis au deuxième, etc. jusqu'au dernier feu. Supposons par exemple que les trois premiers feux suivent le modèle classique, les trois premiers feux sont donc obligés de changer d'état à chaque temps. La combinaison des trois premiers feux $\{rouge, orange, vert\}$ ne donnera aucune solution au problème car au dernier pas de temps les feux ne respecteront pas la contrainte 2. En effet, étant donné que les trois premiers feux changent de valeur à chaque temps, au moins un d'entre eux sera orange à chaque temps et donc avoir, au dernier pas de temps, un feu à vert et tous les autres feux à rouge n'est pas possible. Le solveur ne le sait pas et va perdre du temps en continuant d'affecter des valeurs aux variables jusqu'à ce que toutes les variables soient affectées et tester toutes les combinaisons de valeurs aux autres variables pour se rendre compte qu'aucune ne fonctionne. Le model checker donne "faux" pour cette affectation de variables, nous pouvons donc dire au solveur de backtracker immédiatement et éviter de perdre du temps à affecter inutilement des valeurs aux variables. Dans l'approche que nous proposons, à chaque fois que le model checker est appelé par le solveur PPC, le solveur PPC attend la réponse rendue par le model checker avant de continuer. Un objectif de notre travail est de déterminer à quels moments il est rentable d'utiliser un model checker pour pouvoir couper l'arbre de recherche et ainsi faire gagner du temps au solveur.

4 Résultats Préliminaires

Nous utilisons la version 6.5.1 de 2020 de Spin et la version 4.10.7 de Choco. Les expérimentations ont été effectuées sur un processeur 11th Gen Intel(R) Core(TM) i7-11850H @ 2.50GHz.

Nous avons testé différents modèles avec $n = 6$ et $n = 8$ feux, où la première moitié des feux sont instanciés selon le modèle classique et le reste selon le modèle limité avec une limite $l = 1$ (les feux peuvent avoir la même valeur sur au maximum deux unités de temps successives).

Nous avons implémenté l'appel au model checker à l'aide d'une contrainte sur les feux. Cette contrainte est appelé par le solveur Choco à chaque propagation, dés que le domaine d'une variable est modifié.

Nous comparons la résolution du problème des feux avec le solveur Choco seul, à la résolution avec la contrainte spéciale pour le model checker. Le Tableau 1 présente les résultats préliminaires de nos expérimentations.

Notre approche est beaucoup plus lente en temps que le solveur Choco seul, mais nous arrivons, quelque soit l'instance, à réduire d'environ 1% le nombre de backtracks effectués et coupons donc environ 1% des nœuds. Cela est prometteur et nous a donc poussé à regarder d'autres types de contraintes. Des premiers résultats très préliminaires montrent que des contraintes de type "No-Good" [Lecoutre et al., 2007] permettraient d'être plus rapide. C'est ce que nous comptons explorer.

Le Tableau 1 met en évidence que les appels à notre contrainte sont très rapides (entre 0.3 millisecondes et 10 millisecondes), mais qu'il y a beaucoup trop d'appels. Sur ce très grand nombre d'appels, très peu coupent l'arbre de recherche (appelés appels utiles dans le Tableau 1). Nous souhaitons donc trouver quels sont les points stratégiques où appeler notre contrainte. Nous avons remarqué qu'avec peu de variables instanciées (moins de la moitié pour un temps donné) nous gagnons le plus de temps et coupons le plus l'arbre de recherche. De plus, même si nous trouvons beaucoup de combinaisons fausses quand nous avons beaucoup de variables instanciées (plus des deux tiers pour un temps donné), le gain dans l'arbre de recherche est minimal.

5 Conclusion et Perspectives

Dans cet article nous avons montré que combiner le Model-Checking avec la Programmation par Contraintes pour résoudre un problème combinatoire dynamique est une approche prometteuse pour couper l'arbre de recherche. Nous avons beaucoup d'espoir pour la suite. Nous aimerions arriver à identifier quand il est rentable d'appeler le model checker afin d'éviter des appels inutiles qui font perdre du temps. Notre objectif à long terme serait d'identifier une classe de problèmes de programmation par contraintes dy-

		Choco seul			Avec Model-Checking			# Propagations de notre contrainte	
n	T	Temps CPU (s)	# Backtrack	# Nœuds	Temps CPU (s)	# Backtrack	# Nœuds	# Appels utiles	# Appels inutiles
6	8	**0.16**	12 339	9 778	140	**12 247**	**9 732**	58	40 790
6	10	**0.5**	121 899	96 094	289	**121 263**	**95 776**	420	407 988
6	12	**5**	1 793 719	1 420 066	2 220	**1 784 955**	**1 415 684**	5 450	5 953 216
8	6	**0.2**	23 179	17 236	738	**23 085**	**17 189**	449	74 533
8	8	**1**	544 814	417 613	4 044	**543 111**	**416 760**	8 517	1 664 988

TABLE 1 – Solveur Choco avec et sans Model Checking.

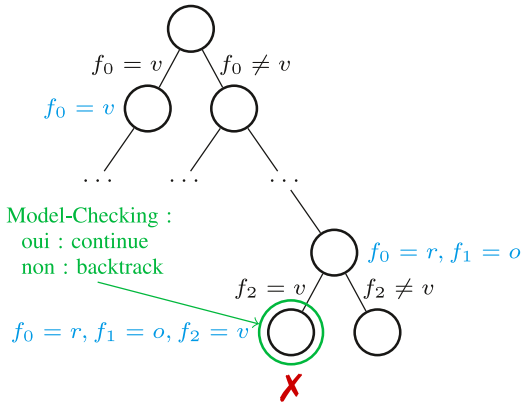


FIGURE 5 – Arbre de recherche pour un temps donné.

namique pour laquelle notre stratégie serait efficace. Les problèmes bien codifiés comme celui de l'inter-blocage dans des systèmes d'aiguillage [Winter, 2002] sont des bons candidats. Pour que l'utilisateur n'ait à se soucier que de l'implémentation en Programmation par Contraintes, nous souhaitons écrire un traducteur automatique de modèles en Programmation par Contraintes vers des modèles en Promela, le langage utilisé par le model checker Spin. La rédaction de propriétés de logique temporelle appropriées et leur appel à partir d'un solveur seront également automatisés.

Références

[Bart, 2017] Bart, A. (2017). *Constraint modelling and solving of some verification problems. (Modélisation et résolution par contraintes de problèmes de vérification)*. PhD thesis, IMT Atlantique, France.

[Bekkouche, 2017] Bekkouche, M. (2017). Combining techniques of bounded model checking and constraint programming to aid for error localization. *Constraints An Int. J.*, 22(1) :93–94.

[Chan et al., 1997] Chan, W., Anderson, R. J., Beame, P., and Notkin, D. (1997). Combining constraint solving and symbolic model checking for a class of a systems with non-linear constraints. In *CAV*, volume 1254 of *Lecture Notes in Computer Science*, pages 316–327. Springer.

[Cimatti et al., 1999] Cimatti, A., Clarke, E. M., Giunchiglia, F., and Roveri, M. (1999). NUSMV : A new symbolic model verifier. In *Computer Aided Verification, 11th International Conference, CAV '99, Trento, Italy, July 6-10, 1999, Proceedings*, pages 495–499.

[Clarke et al., 1986] Clarke, E. M., Emerson, E. A., and Sistla, A. P. (1986). Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2) :244–263.

[Clarke et al., 1999] Clarke, E. M., Grumberg, O., and Peled, D. A. (1999). *Model Checking*. MIT Press, Cambridge, MA, USA.

[Clarke et al., 1996] Clarke, E. M., McMillan, K. L., Campos, S. V. A., and Hartonas-Garmhausen, V. (1996). Symbolic model checking. In Alur, R. and Henzinger, T. A., editors, *Computer Aided Verification, 8th International Conference, CAV '96, New Brunswick, NJ, USA, July 31 - August 3, 1996, Proceedings*, volume 1102 of *Lecture Notes in Computer Science*, pages 419–427. Springer.

[Collavizza et al., 2008] Collavizza, H., Rueher, M., and Hentenryck, P. V. (2008). CPBPV : A constraint-programming framework for bounded program verification. In *CP*, volume 5202 of *Lecture Notes in Computer Science*, pages 327–341. Springer.

[Esparza and Melzer, 1997] Esparza, J. and Melzer, S. (1997). Model checking LTL using constraint programming. In *ICATPN*, volume 1248 of *Lecture Notes in Computer Science*, pages 1–20. Springer.

[Flanagan, 2003] Flanagan, C. (2003). Automatic software model checking using CLP. In *ESOP*, volume 2618 of *Lecture Notes in Computer Science*, pages 189–203. Springer.

[Holzmann, 2003] Holzmann, G. (2003). *Spin Model Checker, the : Primer and Reference Manual*. Addison-Wesley Professional, first edition.

[Holzmann, 1996] Holzmann, G. J. (1996). On-the-fly model checking. *ACM Comput. Surv.*, 28(4es) :120.

[Lecoutre et al., 2007] Lecoutre, C., Sais, L., Tabary, S., and Vidal, V. (2007). Nogood recording from restarts. In Veloso, M. M., editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 131–136.

[Nilsson and Lübcke, 2000] Nilsson, U. and Lübcke, J. (2000). Constraint logic programming for local and symbolic model-checking. In *Computational Logic*, volume 1861 of *Lecture Notes in Computer Science*, pages 384–398. Springer.

[Prud'homme et al., 2016] Prud'homme, C., Fages, J.-G., and Lorca, X. (2016). Choco solver documentation. *TASC, INRIA Rennes, LINA CNRS UMR*, 6241.

[Vizel and Gurfinkel, 2014] Vizel, Y. and Gurfinkel, A. (2014). Interpolating property directed reachability. In *CAV*, volume 8559 of *Lecture Notes in Computer Science*, pages 260–276. Springer.

[Welp and Kuehlmann, 2013] Welp, T. and Kuehlmann, A. (2013). QF BV model checking with property directed reachability. In *Design, Automation and Test in Europe, DATE 13, Grenoble, France, March 18-22, 2013*, pages 791–796.

[Winter, 2002] Winter, K. (2002). Model checking railway interlocking systems. In Oudshoorn, M. J., editor, *Computer Science 2002, Twenty-Fifth Australasian Computer Science Conference (ACSC2002), Monash University, Melbourne, Victoria, Australia, January/February 2002*, volume 4 of *CRPIT*, pages 303–310. Australian Computer Society.

Cadre expérimental pour l'évaluation de méthodes hybrides en configuration interactive sous contraintes

Idir. Boumbar[1, 2], Élise. Vareilles[1], Paul. Gaborit[2], Xavier. Lorca[2]

[1] ISAE Supaero, DISC, Toulouse
[2] IMT Mines Albi, CGI, Albi

[1]prenom.nom@isae-supaero.fr [2]prenom.nom@mines-albi.fr

Résumé

La configuration de systèmes est généralement formalisée sous forme d'un problème de satisfaction de contraintes. Le modèle générique d'une famille de systèmes se confond alors avec le CSP. Deux types d'approches permettent de résoudre un CSP et de définir ainsi une solution de configuration : les approches par filtrage (qui agissent de façon itérative sur le réseau de contraintes) et les approches par compilation (qui raisonnent sur l'espace des solutions). Ces deux approches ont chacune des avantages et des inconvénients, en termes de temps de compilation, temps de filtrage, compacité des solutions, etc. Dans nos travaux de thèse, nous souhaitons hybrider ces deux approches afin de tirer profit de leurs avantages et de limiter leurs défauts, sur des problèmes de configurations. Dans ce contexte, le but de cet article est d'établir un cadre expérimental qui permet de valider la pertinence d'une approche hybride des approches par filtrage et par compilation. Nous analyserons les caractéristiques des approches hybrides, pour mettre en évidence des critères comme l'évolutivité d'un CSP. Ensuite, nous reprendrons un protocole de configuration basé sur des CSP majoritairement discrets, avec des problèmes de configuration de tailles différentes, avec et sans variables continues. Nous nous proposons de l'élargir à d'autres types de CSP pour prendre en compte les spécificités des problèmes de configuration (architecture de CSP, CSP dynamique, etc.). Grâce à ce cadre, des premières hypothèses d'hybridation seront formulées.

Mots-clés

configuration hybridation compilation de connaissances propagation de contraintes

Abstract

The configuration of systems is usually formalized as a constraint satisfaction problem. The generic model of a family of systems is then merged with the CSP. Two types of approaches are used to solve a CSP and thus define a configuration solution : filtering approaches (which act iteratively on the constraint network) and compilation approaches (which reason on the solution space). Both approaches have advantages and disadvantages, in terms of compilation time, filtering time, compactness of solutions, etc. In our thesis work, we wish to hybridize these two approaches in order to take advantage of their advantages and limit their drawbacks, on configuration problems. In this context, the aim of this paper is to establish an experimental framework that allows us to validate the relevance of a hybrid approach of the filtering and compilation approaches. We will analyze the characteristics of hybrid methods, to highlight criteria such as the scalability of a CSP. Then, we will take up a configuration protocol based on mostly discrete CSPs, with different sizes of configuration problems, with and without continuous variables. We propose to extend it to other types of CSP to take into account the specificities of configuration problems (CSP architecture, dynamic CSP, etc). Thanks to this framework, first hybridization hypotheses will be formulated.

Keywords

configuration hybridization knowledge compilation constraint propagation

1 Introduction

Depuis les dernières décennies le marché doit faire face à une demande croissante en produits ou services personnalisés dans tous les secteurs. De plus, cette demande s'accompagne d'un essor des technologies du web pour le commerce qui permettent un lien plus direct entre les clients et les entreprises. Pour répondre à ces enjeux, de nombreuses entreprises proposent maintenant à leurs clients potentiels de personnaliser leurs produits ou services directement en ligne. Pour y parvenir les entreprises doivent faire face à deux problématiques : la définition de systèmes configurables et la mise en ligne de configurateurs interactifs.

La définition de systèmes configurables repose sur l'exploitation d'un catalogue de systèmes ou de familles de systèmes mais aussi des différentes connaissances métier directement ou indirectement liées aux systèmes. Pour représenter et manipuler l'ensemble de ces connaissances dont la combinatoire peut rapidement exploser, il est nécessaire de recourir à des formalismes adaptés. Une des méthodes souvent utilisées est la modélisation de la connaissance des familles de systèmes sous forme d'un problème de satisfaction de contraintes ou CSP. D'une part le formalisme CSP

parvient à bien représenter les relations entre les composants du système, son architecture, les compatibilités de ses composants, ses variantes, etc. et les choix des utilisateurs. D'autre part, il fournit de nombreuses approches pour traiter les modèles de connaissances et supporter le processus de configuration interactive.

Parmi ces approches se distinguent les approches par filtrage et les approches par compilation. Les approches par filtrage produisent des déductions sur le problème de satisfaction de contraintes afin de retirer tout au long de la configuration des valeurs qui ne seraient plus cohérentes avec le choix de l'utilisateur. Ces méthodes ont l'avantage de traiter de nombreux types de contraintes mais ne garantissent pas la *complétude* du filtrage : des valeurs incompatibles avec les choix courants peuvent ne pas être retirées. Les approches par compilation proposent de transformer le problème de satisfaction de contraintes sous la forme d'un graphe qui permet de naviguer efficacement dans l'espace des solutions. Bien qu'elles permettent de retirer les valeurs incompatibles avec les choix courants, leur complexité spatiale est théoriquement exponentielle.

Afin d'améliorer la réponse aux exigences de la configuration interactive (garantie d'un espace de solutions atteignables et conformes, temps de réponse réduit, etc.), cet article propose les premières idées d'hybridation des méthodes par filtrage et par compilation[18]. Par hybridation, nous entendons une utilisation conjointe des deux approches pour l'aide à la configuration interactive de systèmes. Ceci permettrait de tirer avantage des deux méthodes et de limiter leurs défauts.

L'objectif de cet article est de décrire un cadre expérimental qui permettrait de valider des approches hybrides de filtrage et de compilation. En effet, pour répondre à cette problématique, il est important d'identifier les critères qui mesurent les exigences de la configuration interactive. Ainsi, grâce à un protocole de configuration[19], il sera possible de valider la pertinence des approches d'hybridation qui sont évoquées dans cet article.

La section 2 pose les définitions générales de configuration, de programmation par contraintes, et de typologie de CSP. Les sections 3 et 4 présentent les deux approches utilisées pour résoudre interactivement un CSP ainsi que leurs avantages et inconvénients. La section 5 présente l'approche hybride de filtrage et de compilation. Ensuite, le cadre expérimental de l'étude ainsi que les premières pistes d'hybridation à explorer sont définies.

2 Définitions générales

La configuration interactive fait souvent appel à la notion de *problème de satisfaction de contraintes* pour représenter et modéliser toute la connaissance sur la famille de systèmes. En effet, le formalisme CSP permet de bien traduire les exigences de la configuration interactive. De plus, ce formalisme offre de nombreuses façon de modéliser la connaissance (différents types de CSP).

2.1 Besoin et exigences fonctionnelles de la configuration interactive

La configuration est une forme de conception routinière qui donne le choix à l'utilisateur des fonctionnalités et/ou composants qu'il désire dans son système. Elle se définit comme suit[13] : soit un ensemble de composants A, décrits par des propriétés et des compatibilités, d'une description désirée B du produit et d'un ensemble de critères d'évaluation C. Configurer consiste à trouver une solutions parmi les composants qui réponde à toutes les exigences soulevées par les éléments de A, B et C.

En configuration interactive [17], l'utilisateur final ou le client spécifie de façon itérative le produit désiré jusqu'à aboutir à une solution acceptable. Chaque choix réalisé par l'utilisateur durant la configuration élimine les solutions qui ne respectent pas les besoins formulés et les critères définis par l'ensemble des composants. Ainsi, l'ensemble des choix possibles se réduit jusqu'à aboutir à une solution complètement configurée.

La notion de modèle générique[1, 12] est souvent utilisée pour formaliser la connaissance sur une famille de systèmes. Le modèle générique décrit :

- la nomenclature, qui structure tous les composants du système ;
- les connaissances sur chaque composant ;
- la cardinalité de chaque composant ;
- les options possibles du système ;
- les contraintes intra/inter composants-composés du système ;
- l'ensemble des KPI ("key performance indicator") du système.

Ainsi, la configuration de certains modèles génériques peut se heurter à la difficulté de représenter des systèmes fortement combinatoire.

Un configurateur est un outil qui permet d'exploiter ce modèle générique. Sa fonction est d'aider à trouver une solution respectant toutes les spécifications et répondant aux besoins de l'utilisateur. Plus particulièrement dans le cadre de la configuration interactive[16, 17], le configurateur doit permettre à chaque itération de :

- visualiser les choix déjà formulés jusqu'à cette étape ;
- formuler des nouveaux choix sur le système parmi ceux respectant le modèle générique ;
- évaluer le système configuré relativement à des indicateurs de performance (coût, délai de livraison).

Enfin le configurateur doit assurer que les choix proposés à l'utilisateur mènent bien à un système réalisable et doit garantir des temps de réponses/traitements.

2.2 Problème de satisfaction de contraintes

L'une des méthodes couramment utilisées par les configurateurs pour décrire un modèle générique est le problèmes de satisfaction de contraintes (CSP : *Constraint Satisfaction Problem*). Un CSP [12] est un triplet $P = \{X, D, C\}$ où X est un ensemble de variables, D un ensemble de domaines et C un ensemble de contraintes portant sur les variables de

X.
Une *assignation* d est un ensemble d'affectations de variables de X. Une assignation est dite *complète* (resp. *partielle*) si elle concerne toutes les (resp. une partie des) variables de X. Une *solution* du CSP est définie comme une assignation complète des variables X qui satisfait toutes les contraintes de C. L'ensemble des solutions est noté S.
Dans un CSP $P = \{X, D, C\}$, une valeur $v \in D_{x_i}$ est globalement cohérente inverse[4] (GIC) ssi il existe $s \in S$ une solution de P telle que $s(x_i) = v$ dans l'assignation de s. Un CSP $\{X, D, C\}$ est globalement cohérent inverse (GIC) ssi les valeurs de tous ses domaines sont GIC.
Ainsi le modèle générique d'un système peut être décrit grâce à un CSP : les caractéristiques des composants génériques sont représentées par des variables, chaque variante d'une caractéristique est représentée par un élément du domaine de sa variable et les exigences et compatibilités entre composants sont représentées par des contraintes. Ensuite, l'utilisation du CSP permet de mener une activité de configuration : la spécification des besoins se traduit par des assignations particulières des variables et le rôle du configurateur est de maintenir le CSP globalement cohérent inverse à chaque assignation.

2.3 Typologie de CSP

En fonction du système, il existe différentes façons de formaliser la connaissance sous forme de CSP. Les variables peuvent être de différents types (booléennes, entières, réelles, ...). Les domaines peuvent adopter des formes différentes (intervalle, collection de valeurs, union d'intervalles...)[12, 17]. Les contraintes peuvent être formulées de différents types (formules logiques, formules mathématiques, tables de combinaisons de valeurs autorisées ou interdites, contraintes aux bornes, ...). De plus, les contraintes peuvent être distinguées de deux façons :

- compatibilité : pour exprimer les relations entre les variables et formaliser l'espace de solutions ;
- activation : pour modifier l'espace de solution par l'ajout explicite de variables, domaines et contraintes.

Plusieurs types de CSP différents doivent ainsi être combinés pour formaliser les connaissances et construire un modèle générique de famille de systèmes :

- les CSP discrets et continus pour formaliser les composants de la nomenclature ;
- les CSP génératifs pour l'aspect hiérarchique de la nomenclature ;
- les CSP dynamiques pour représenter les options ;
- les CSP continus pour évaluer les critères de performance.

3 Méthodes par filtrage

Les méthodes par filtrage filtrent le CSP pour retirer du domaine de chaque variable, les valeurs qui ne satisfont pas les contraintes. On utilise ainsi la notion de propagation de contrainte sur le domaine d'une variable. Pour retirer d'un domaine un élément qui ne satisfait pas une contrainte, il est suffisant de vérifier qu'il ne satisfait pas une sous-partie de cette contrainte : on dit qu'on vérifie une incohérence localement. Bien que cette approche permettent de retirer efficacement (i.e. dans un temps polynomial) des valeurs incohérentes, elle ne garantit pas que toutes les valeurs incohérentes sont supprimées. Ainsi la façon dont la contrainte est évaluée permet de définir différents niveaux de cohérence [14] :

- la cohérence d'arc ;
- la cohérence de domaine ;
- la cohérence d'intervalle.

Le filtrage de cohérence locale regroupe les méthodes qui assurent un niveau de cohérence local : cohérence d'arc, d'intervalle et de domaine. Elles ont l'avantage d'être rapides et donc de garantir l'interaction avec l'utilisateur. De plus, elles permettent de traiter de nombreux types de CSP(continus, dynamiques etc.). Cependant l'utilisation de niveau local ne garantit pas le retrait de toutes les non solutions : ils ne garantissent pas la cohérence globale inverse (GIC). Par exemple, pour établir la cohérence d'arc , des algorithmes performants existent comme AC–7[5] ou encore AC2001[6]. Des algorithmes comme [8, 2] permettent d'assurer une cohérence d'arc sur des contraintes de tables. Des algorithmes de Box-cohérence[3] permettent de traiter les variables à domaines continus en travaillant sur la cohérence d'intervalle. Le filtrage de cohérence globale regroupe les algorithmes qui garantissent la cohérence globale inverse (GIC). On peut par exemple citer l'algorithme GIC4[4] qui réalise un filtrage sur les domaines et vérifie que les valeurs non filtrées permettent d'aboutir à une solution. Cependant, pour assurer la GIC cet algorithme peut nécessiter plus de temps pour le traitement ce qui peut nuire à l'interaction avec l'utilisateur.
Dans le cadre de la configuration interactive, les méthodes de filtrage permettent ainsi de propager les choix de l'utilisateur et de lui permettre de faire des choix qui aboutissent à une solution. Toutefois, le recours à un filtrage de cohérence locale peut conduire à conserver dans le CSP des valeurs incohérentes globalement, bien qu'elles soient cohérentes à un certain niveau. Ceci pouvant mener l'utilisateur à construire une solution inexistante durant le processus de configuration et à retourner en arrière pour corriger ses choix.

• Avantages Raisonnement sur tous types de problèmes de configuration (discret, continu ou mixte, etc.)
• Inconvénients Ne garantit toujours pas la GIC. Possible utilisation du mécanisme de retour en arrière pour rétablir la cohérence.

TABLE 1 – Méthodes par filtrage

4 Méthodes par compilation

Les méthodes par compilation [11, 10] consistent à transformer le CSP en compilant ses contraintes dans une structure de données capable de les représenter de façon efficace.

Ensuite la configuration consiste à permettre à l'utilisateur de naviguer dans l'espace des solutions. La phase de compilation qui se déroule avant la phase de configuration peut se ramener à une recherche exhaustive de toutes les solutions du CSP. De ce fait, elle peut occuper un temps significatif et nécessiter un espace important pour stocker le résultat de compilation.
L'aspect positif de cette approche est qu'il permet de garantir la cohérence globale inverse. De plus cette méthode s'affranchit d'un temps de calcul nécessaire pour filtrer les domaines à chaque assignation de variable contrairement à la méthode de filtrage.
L'utilisation de diagrammes de décision multivaluées (Multivalued Decision Diagram - MDD)[15] est particulièrement intéressante pour la compilation de solutions. En effet, elle fournit des outils de traitement en temps polynomial qui permettent de garantir l'interaction durant la configuration. Néanmoins puisque la compilation du CSP s'effectue en amont de la session de configuration, il n'est pas possible de traiter des problèmes à structure dynamique.

• Avantages : Garantie de parvenir à une solution sans retour en arrière. • Inconvénients : Impossibilité de compiler tous les types de problèmes de configuration. Explosion spatiale dans le pire des cas.

TABLE 2 – Méthodes par compilation

5 Discussions et premières pistes d'hybridation

Les approches par filtrage et les approches par compilation sont toutes deux utilisées en configuration interactive (voir sections 3 et 4). Cependant aucune des deux approches seules ne parvient à répondre complètement aux exigences de celle-ci. D'une part, la garantie de la cohérence globale inverse et l'interactivité ne peuvent être garanties que sur des problèmes de configuration discrets par les approches par compilation. Les approches par filtrage parviennent à traiter des problèmes de configuration continus mais sans garantir la GIC. D'autres part, les approches par compilation nécessitent un certain temps de compilation avant utilisation. Ainsi elles peuvent être incompatibles pour des modèles génériques souvent mis à jour. Cette propriété que l'on définit comme *évolutivité d'un modèle* peut être un désavantage pour l'utilisation de la compilation. L'utilisation de solveurs CP pour garantir la GIC est une approche possible mais elle ne peut être envisagée en configuration. En effet, elle a une complexité temporelle trop importante pour garantir l'interaction avec l'utilisateur.
En réponse à cette problématique, une méthode utilisant conjointement les approches par filtrage et par compilation serait une solution. Cette approche hybride permettrait de garantir le caractère GIC sur des problèmes de configuration à variables continues et de limiter les défauts des deux approches classiques.
Pour établir la pertinence de l'approche hybride, il est nécessaire de valider ses performances dans un cadre expérimental. Le protocole de configuration cité dans [19] permet de simuler des sessions de configuration interactive. Il a été utilisé au préalable sur un cas d'étude du projet BR4CP [1] de configuration sur un problème statique et à variables discrètes. Néanmoins pour prouver l'intérêt de la méthode hybride, il est nécessaire de tester des cas d'étude de problèmes dynamiques et/ou à variables continues. Ensuite, il faut évaluer les grandeurs permettant de vérifier les exigences de la configuration interactive[19]. Ces grandeurs peuvent être regroupée selon plusieurs critères :

— critères d'interaction avec l'utilisateur : temps d'exécution total, temps d'exécution de la phase de propagation ;
— critères de cohérence : nombre de valeurs supprimées dans les domaines à chaque propagation, nombre de situations d'impasse ;
— autres : temps de compilation, mémoire utilisée.

Une des premières approches d'utilisation conjointe des approches par filtrage et par compilation imaginées[18] consiste à diviser le problème de configuration en fonction des sous familles de systèmes et de compiler séparément chaque sous problème. Ensuite, il s'agit de propager les choix de l'utilisateur parmi les différents sous-systèmes grâce à des méthodes de filtrage. La figure 1 illustre un exemple de modèle générique sur lequel pourrait s'appliquer cette approche hybride.

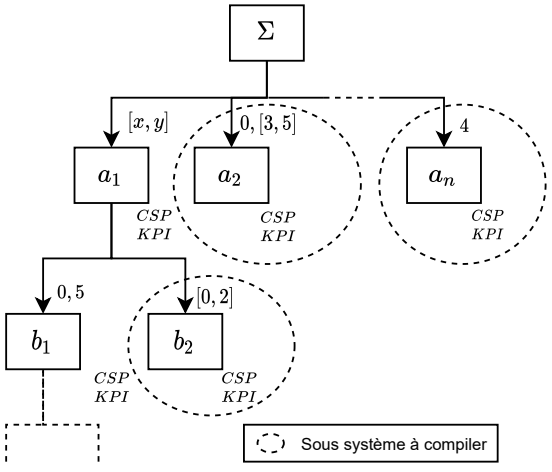


FIGURE 1 – Exemple d'utilisation d'une approche hybride sur un système configurable

Une première implémentation sera menée grâce à l'outil Choco[7]. Après avoir compilé le CSP de chaque sous système en un MDD, l'outil choco sera utilisé pour propager les contraintes durant le processus de configuration. Bien que cette approche soit analogue à l'utilisation de

1. Projet BR4CP ("Business Recommendation for Configurable Products") (ANR-11-BS02-008).

contraintes tables pour représenter les sous systèmes, l'utilisation de MDD est privilégiée car elle permet de garantir la GIC. De plus la forme compilée d'un sous système en MDD a l'avantage d'être bien plus compacte que sous la forme d'une table listant l'ensemble des tuples possibles.
En hypothèse, cette première approche permet de résoudre certaines problématiques des méthodes classiques. D'une part, cela permettrait de traiter des problèmes de configuration dynamiques grâce à la propagation de contraintes d'activation sur des sous systèmes compilés. D'autre part, cela permettrait de traiter des problèmes évolutifs en maintenant plus facilement le système par la compilation unique des sous parties du problème à mettre à jour au lieu de réaliser la compilation de tout le problème.

6 Remerciements

Des remerciements particuliers sont adressés à ISAE-Supaero, IMT Mines Albi et ANITI pour leur soutien scientifique et financier.

Références

[1] M.M. Amini, T. Coudert, É. Vareilles et M. Aldanondo : System Configuration Models : Towards a Specialization Approach. *In IFAC MIM*, 2022.

[2] G. Audemard, C. Lecoutre et M. Maamar : Segmented Tables : An Efficient Modeling Tool for Constraint Reasoning. *In ECAI*, pages 315–322, 2020.

[3] F. Benhamou, D. McAllester et P.V. Hentenryck : CLP(Intervals) Revisited, *In ILPS*, pages 1-21, 1994.

[4] C. Bessiere, H. Fargier et C. Lecoutre : Global Inverse Consistency for Interactive Constraint Satisfaction, *In Proceedings of ICPPCP*, pages 159–174, 2013.

[5] C. Bessiére, E.C. Freuder et J.-C. Regin : Using constraint metaknowledge to reduce arc consistency computation. *In Artificial Intelligence*, pages 125–148, 1999.

[6] C. Bessiere et J.-C. Régin : Refining the Basic Constraint Propagation Algorithm., *In IJCAI*, pages 309–315, 2001.

[7] C. Prud'homme, J-G. Fages et X. Lorca : Choco Solver Documentation. *choco-solver.org/docs/*, 2016

[8] J. Demeulenaere, R. Hartert, C. Lecoutre, G. Perez, L. Perron, J.-C. Régin et P. Schaus : Compact-Table : Efficiently Filtering Table Constraints with Reversible Sparse Bit-Sets, *In Proceedings of ICPPCP*, pages 207–223, 2016.

[9] A. Falkner, A. Haselböck, G. Krames, G. Schenner, H. Schreiner et R. Taupe : Solver Requirements for Interactive Configuration. *In Journal of Universal Computer Science*, pages 343–373, 2020.

[10] H. Fargier, P. Marquis et N. Schmidt : Compacité pratique des diagrammes de décision valués. Normalisation, heuristiques et expérimentations. *Revue d'Intelligence Artificielle*, pages 571–592, 2014.

[11] T. Hadzic, S. Subbarayan, R.M. Jensen, H.R. Andersen, J. Møller et H. Hulgaard : Fast backtrack-free product configuration using a precompiled solution space representation. *In Proceedings of ICPPCP*, 2004.

[12] F.Rossi, P. van Beek, T. Walsh, C. Bessiere et U. Junker : Handbook of Constraint Programming, emphElsevier, pages 27-83 et 835-873, 2006.

[13] D. Sabin et R. Weigel : Product configuration frameworks - A survey. *In : IEEE Intelligent System and their Applications*, pages 30-31, 1998.

[14] T. Schiex : Réseaux de contraintes. *Habilitation à diriger des recherches, INRA*, 2000.

[15] N. Schmidt : Compilation de préférences : application à la configuration de produit *Thèse de doctorat, Artois*, 2015.

[16] J. Tiihonen et T. Soininen : Product Configurators : Information System Support for Configurable Products,*Technical Report, Helsinki University of Technology, Laboratory of Information Processing Science* 1997.

[17] É. Vareilles : Configuration interactive et contraintes : connaissances, filtrage et extensions *Habilitation à diriger des recherches, Institut National Polytechnique de Toulouse*, 2015.

[18] É. Vareilles, H. Fargier, M. Aldanondo et P. Gaborit : ICONIC : Interactive CONstraint-based Configuration, *In 19 Th International Configuration Workshop.* pages 28-32, 2017.

[19] Y. Izza : Résolution interactive et compilation de problèmes de satisfaction de contraintes. *Mémoire de Master, Université Paul Sabatier – Toulouse 3*, 2014.

Session 2 : Extension du formalisme CSP, contraintes globales

Vues de domaine globales

Dimitri Justeau-Allaire[1,2]
Charles Prud'homme[3]

[1] Institut Agronomique néo-Calédonien (IAC), Noumea, New Caledonia
[2] AMAP, Univ Montpellier, CIRAD, CNRS, INRA, IRD, Montpellier, France
[3] TASC, IMT-Atlantique, LS2N-CNRS, F-44307 Nantes

dimitri.justeau@gmail.com charles.prudhomme@imt-atlantique.fr

Résumé

Le concept de vues de domaine est une abstraction puissante en programmation par contraintes. Il permet de définir des variables qui ne déclarent aucun domaine, mais qui reposent sur une variable x et une fonction f, de sorte que $y = f(x)$ où y est la vue. En plus de faciliter la modélisation en fournissant une couche d'abstraction expressive, les vues offrent une alternative à la décomposition des contraintes qui n'implique pas de variables auxiliaires et de propagateurs. Dans cet article, nous introduisons la notion de vues de domaine globales. Une vue de domaine globale repose sur un nombre arbitraire de variables et une fonction telle que $y = f(x_1, ..., x_n)$. La combinaison de vues de domaine globales avec des variables ensemblistes et de graphes étend l'expressivité de la programmation par contraintes en permettant la définition de relations complexes entre différents types de variables dans un cadre léger et simple.

Mots-clés

Programmation par contraintes ;Vues de domaine ; Global

Abstract

The concept of domain views is a powerful abstraction in constraint programming. It permits to define variables that do not declare any domain but instead rely on a variable x and a function f, such that $y = f(x)$ where y is the view. In addition to making modelling easier by providing an expressive layer of abstraction, views provide an alternative to constraint decomposition that does not involve auxiliary variables and propagators. In this article, we introduce the notion of global domain view. A global domain view relies on an arbitrary number of variables and a function such that $y = f(x_1, ..., x_n)$. The combination of global domain views with set and graph variables extends the expressiveness of constraint programming by allowing the definition of complex relationships between different types of variables within a light and simple framework.

Keywords

Constraint programming ; ; Domain views ; Global

Avant-propos

Le présent article est un résumé de la lettre publiée dans *Constraints* [11]. Nous ne rappelons ici que les motivations et la contribution théorique, invitant le lecteur à lire l'article initial pour plus de détails.

1 Introduction

L'idée centrale de la programmation par contraintes (PPC) est la représentation des problèmes comme des réseaux de *variables* et de *contraintes* représentant des relations logiques qui doivent être satisfaites par toute solution. Chaque variable x d'un problème prend ses valeurs dans un domaine fini noté $D(x)$. L'un des principaux avantages distinctifs de la PPC par rapport à d'autres techniques telles que la programmation linéaire en nombres entiers mixtes (MILP) est son *expressivité*. En effet, le paradigme de modélisation de la PPC offre un langage riche qui permet aux utilisateurs de représenter leurs problèmes avec des niveaux d'abstraction élevés. La diversité des types de variables est un bon exemple de cette expressivité. Les problèmes peuvent être représentés avec des variables entières et booléennes classiques, mais aussi avec des variables ensemblistes [4, 5] ou des variables de graphes [2, 3], qui fournissent toutes deux une représentation compacte et efficace par des intervalles d'ensembles (resp. de graphes).

2 Vues

Le concept de *vues* a été introduit en PPC pour déclarer des variables qui sont définies à partir d'autres variables [8, 10]. Une vue y ne déclare pas directement de domaine mais s'appuie sur une variable observée x et une fonction $f : D(x) \mapsto D(y)$ telle que $y = f(x)$. Cette abstraction permet de déléguer toutes les opérations du domaine de y à x et offre un moyen simple et léger d'exprimer des contraintes complexes comme ALLDIFFERENT$(x_1+1, ..., x_n+n)$. Les vues offrent de nombreux avantages tant aux utilisateurs qu'aux développeurs de solveurs de PPC. Tout d'abord, elles offrent une couche d'abstraction simple et expressive qui rapproche le langage de modélisation de la PPC du langage naturel du modélisateur [8]. Les vues fournissent également une alternative légère à la décomposition des

contraintes, qui évite d'introduire des variables auxiliaires et des propagateurs [10]. Enfin, les vues permettent de réduire la quantité de code à écrire et à maintenir en éliminant le besoin d'implémenter différentes variantes de propagateurs [9]. Dans cet article, nous ne considérons que les *vues de domaine* qui ne délèguent que les opérations de domaine [10] par opposition aux *vues de variable* qui délèguent à la fois les opérations de domaine et de contraintes [8]. Les vues de domaine simplifient l'implémentation de la propagation basée sur la valeur et supportent de manière transparente les fonctions non-injectives [10].

Definition 1 (Vue de domaine) *Étant donné une variable $x \in D(x)$ et une fonction $f : D(x) \mapsto D(y)$, une* vue de domaine $y = f(x)$ *est une variable qui délègue toutes les opérations du domaine à x. Une vue de domaine y peut être manipulée et contrainte comme n'importe quelle autre variable, et supporte les fonctions non injectives.*

Alors que les variables stockent leur domaine et la liste des propagateurs à notifier lorsque le domaine change, les vues de domaine, telles que décrites dans [10], stockent les propagateurs mais délèguent toutes les opérations du domaine à la variable observée. Les variables doivent en retour stocker leurs vues pour les notifier en cas de modification du domaine. À notre connaissance, les vues de domaine n'ont été considérées que pour des fonctions unaires (définie à partir d'une variable) impliquant variables booléennes et entières (notons que les vues ensemblistes ont néanmoins été considérées dans [8] comme des vues de variables unaires).

Contribution Dans cet article, nous montrons que les vues de domaine peuvent être facilement étendues aux relations globales (définie à partir d'un nombre arbitraire de variables) tout en préservant leurs avantages comme le support des fonctions non-injectives. Les vues globales sur les variables ensemblistes et de graphes constituent une couche d'abstraction simple et expressive pour modéliser des problèmes complexes inter-domaines (c'est-à-dire impliquant différents types de variables). Par exemple, les vues algébriques sur les variables ensemblistes et de graphe, telles que l'union et l'intersection, sont simples à mettre en œuvre avec les vues de domaine globales. Ces extensions des vues en PPC fournissent des avancées significatives en termes d'expressivité et de possibilités de modélisation.

3 Vues de domaine globales

En PPC, lorsque le terme globalité est utilisé, il fait généralement référence aux contraintes. Dans [1], les auteurs proposent de décomposer la globalité d'une contrainte en trois aspects : sémantique, opérationnelle et algorithmique. Dans cet article, en associant *global* avec *vue*, nous décrivons simplement qu'une vue implique un nombre non fixe de variables ou plus formellement que f est une fonction *variadique* [6] qui accepte un nombre variable d'arguments. Sur la base de la description proposée dans [10], les vues de domaine peuvent être étendues pour supporter les vues globales avec quelques modifications. Une vue de domaine globale y est construite à partir d'un tableau de variables $\{x_1, ..., x_n\} \in D(x_1) \times ... \times D(x_n)$. Chaque variable x_i doit stocker une référence à la vue y pour programmer les contraintes requises sur les modifications du domaine. Notez que toute variable x_i peut être une vue, car les vues sont considéréee de manière transparente comme des variables.

Definition 2 (Vue de domaine globale) *Étant donné une fonction variadique f et un ensemble de variables $\{x_1, \ldots, x_n\}$, une* vue de domaine globale $y = f(x_1, \ldots, x_n)$ *est une variable qui délègue toutes les opérations du domaine à $\{x_1, \ldots, x_n\}$. Une vue de domaine globale y peut être manipulée et contrainte comme n'importe quelle autre variable.*

Les vues de domaine globales offrent les mêmes caractéristiques que les vues de domaine unaires, notamment la généralisation aux fonctions non-injectives.

Definition 3 (Inverse) *L'inverse de $f^{-1} : \mathcal{Y} \mapsto \mathcal{P}(D(x_1) \times \ldots \times D(x_n))$ d'une fonction variadique non injective $f : D(x_1) \times \ldots \times D(x_n) \mapsto \mathcal{Y}$ est définie comme :*

$$f^{-1}(y) = \begin{cases} \bot \text{ si } \nexists (v_1, ..., v_n) \in D(x_1) \times ... \times D(x_n) \\ \qquad \text{t.q. } f(v_1, ..., v_n) = y; \\ \{(v_1, ..., v_n) \in D(x_1) \times ... \times D(x_n) \mid f(v_1, ..., v_n) = y\} \\ \qquad \text{autrement}. \end{cases}$$

Conceptuellement, les vues de domaine globales ne diffèrent pas des vues de domaine telles que définies dans [10] mais elles étendent considérablement la portée des vues, notamment lorsqu'elles sont appliquées à des variables ensemblistes et de graphes. De plus, elles fournissent la base pour que les vues supportent des fonctions multivariées lorsque n est fixé ; *e.g.,* si $n = 3$ la vue est ternaire.

4 Discussion

Dans cet article, nous avons montré comment les vues de domaine s'étendent naturellement aux fonctions variadiques. Cette extension, comme la version unaire, supporte les fonctions non-injectives. Ces vues peuvent être construites pour des variables ensemblistes et de graphes et alors surpasser les modèles équivalents basés sur la décomposition de contraintes. Enfin, les vues sont simples à implémenter dans les solveurs [1], ont un fort pouvoir expressif et conduisent à des modèles plus compacts et plus proches du langage naturel des modélisateurs (par exemple, les utilisateurs n'ont pas besoin de déclarer les bornes d'une vue). Les vues constituent un grand atout pour une utilisation généralisée de la programmation par contraintes, car elles renforcent ses avantages distinctifs par rapport aux autres techniques de satisfaction de contraintes et d'optimisation sous contraintes : expressivité, flexibilité et modélisation inter-domaines. Les vues sont également beaucoup plus proches du langage naturel que leur homologue de décomposition de contraintes. Elles peuvent donc être d'une grande aide pour la génération automatique de modèles à partir du traitement du langage naturel.

1. Les vues ensemblistes et les vues de graphes ont été implémentées dans `choco-solver` 4.10.7 [7]

Références

[1] Bessière, C., Van Hentenryck, P. : To Be or Not to Be ... a Global Constraint. In : F. Rossi (ed.) Principles and Practice of Constraint Programming – CP 2003, Lecture Notes in Computer Science, pp. 789–794. Springer, Berlin, Heidelberg (2003).

[2] Dooms, G., Deville, Y., Dupont, P. : CP(Graph) : Introducing a Graph Computation Domain in Constraint Programming. In : Principles and Practice of Constraint Programming - CP 2005, Lecture Notes in Computer Science, pp. 211–225. Springer, Berlin, Heidelberg (2005).

[3] Fages, J.G. : On the use of graphs within constraint-programming. Constraints **20**(4), 498–499 (2015).

[4] Gervet, C. : Set Intervals in Constraint Logic Programming : Definition and implementation of a language. Ph.D. thesis, Université de Franche Comté Besançon (1995)

[5] Gervet, C. : Interval propagation to reason about sets : Definition and implementation of a practical language. Constraints **1**(3), 191–244 (1997).

[6] Leonard, H.S., Goodman, N. : The Calculus of Individuals and Its Uses. The Journal of Symbolic Logic **5**(2), 45–55 (1940).

[7] Prud'homme, C., Fages, J.G., Lorca, X. : Choco Documentation (2017)

[8] Schulte, C., Tack, G. : Views and Iterators for Generic Constraint Implementations. In : B. Hnich, M. Carlsson, F. Fages, F. Rossi (eds.) Recent Advances in Constraints, Lecture Notes in Computer Science, pp. 118–132. Springer, Berlin, Heidelberg (2006).

[9] Schulte, C., Tack, G. : View-based propagator derivation. Constraints **18**(1), 75–107 (2013).

[10] Van Hentenryck, P., Michel, L. : Domain Views for Constraint Programming. In : B. O'Sullivan (ed.) Principles and Practice of Constraint Programming, Lecture Notes in Computer Science, pp. 705–720. Springer International Publishing, Cham (2014).

[11] Justeau-Allaire, Dimitri and Prud'homme, Charles : Global domain views for expressive and cross-domain constraint programming. Constraints (2022).

Variables de Séquence pour les problèmes de tournée de véhicules

A. Delecluse[1,2], P. Schaus[1], P. Van Hentenryck[3]

[1] ICTEAM, UCLouvain, Belgique
[2] TRAIL, Belgique
[3] Georgia Institute of Technology, USA

6 mai 2022

Résumé

Nous proposons une variable ciblant les problèmes de tournées de véhicules : la variable de séquence. La représentation de son domaine permet une recherche sur base d'insertions dans une route partiellement formée, ainsi que l'implémentation d'algorithmes simples, mais puissants, permettant de respecter des temps de transition entre les visites ou des capacités dans un véhicule. Nos expériences démontrent que cette variable est suffisamment flexible pour modéliser des problèmes d'itinéraires très contraints, tout en les résolvant de manière efficace.

Mots-clés

Variable de séquence, Voyageur de commerce, livraison, transport de patients, programmation par contraintes.

1 Introduction

Les problèmes de tournées de véhicules (PTV) [19] apparaissent fréquemment dans la distribution de biens dans la chaîne logistique. De par l'augmentation de l'urbanisation et des défis écologiques, il est également attendu que les offres de transport flexibles, telles que le transport à la demande, soient davantage développées à l'avenir. Cela soulève de nouveaux enjeux pour l'optimisation, en particulier le développement d'outils génériques et réutilisables pour les nombreux contextes et variants du PTV.
La programmation par contraintes (PPC) est une des approches les plus flexibles pour la modélisation de PTV. L'approche standard consiste en un modèle de successeurs, introduisant une variable par lieu visité qui représente la visite suivante dans le parcours d'un véhicule. En dépit de sa simplicité, ce modèle souffre de deux limitations majeures : l'impossibilité de représenter des visites optionnelles sans ajouts de valeurs spéciales ainsi que le rajout d'une visite au milieu d'un itinéraire partiellement formé. Le but de la variable de séquence est de pallier à ces deux limitations :

1. Elle peut facilement modéliser l'exclusion de visites, similairement à une variable d'ensemble.
2. Inspirée de l'idée du graphe d'insertion [3], elle permet l'ajout d'une visite au milieu d'un itinéraire partiellement formé, permettant l'emploi d'algorithmes de recherche d'insertions en profondeur [3, 8] pour pouvoir réinsérer de manière optimale un ensemble de visites relaxées dans une recherche en large voisinage (RLV).

Nous commençons par un survol de travaux antérieurs sur approches avec séquences, avant de détailler notre variable, quelques contraintes applicables sur son domaine et son emploi sur 3 variants de PTV.

2 Travaux antérieurs

Dans [18], les auteurs ont introduit une variable de séquence pour des problèmes d'horaires et de PTV. La représentation du domaine de cette variable étend directement celle du sous-ensemble lié pour des variables d'ensemble [7], de part une partition des visites en requises, possibles et exclues, ainsi qu'une séquence partielle et un ensemble d'insertion
Bien que non publié, IBM ILOG CP Optimizer [11] dispose également de variables de séquences pour décider l'ordre de visites, davantage axé sur la planification mais néanmoins utilisé pour des problèmes de tournées de vehicules. Leurs fonctionalités et contraintes sont brièvement décrites [9, 10] sans pour autant donner leur implémentation exacte. Selon leur Interface de Programmation [5, 6], elles se basent sur une structure tête-queue, maintenant séparément l'agrandissement de la tête et de la queue pour rajouter des variables d'intervalle au début ou à la fin de la séquence, respectivement. Cette implémentation semble similaire à celle de Google OR-Tools [15] et ses propres variables de séquences [16]. Elle a été employée pour résoudre le problème du transport de patients dans [4] et [12].

3 Variable de séquence

Nous introduisons les notations sur les séquences avant de formaliser le domaine de notre variable et décrire comment l'implémenter dans un solutionneur de PPC. Notre variable se base essentiellement sur celle de [18] mais en y enlevant le set requis. Par conséquent, une visite possible doit être directement planifiée à un endroit précis dans une séquence partiellement formée, et ne peut pas être simplement requise. Cette modification, en dépit de sa simplicité, permet de simplifier grandement le raisonnement fait par les contraintes, leur complexité temporelle ainsi que l'implémentation d'heuristiques, tout en perdant relativement peu

Le premier auteur est un doctorant

de flexibilité en pratique. La variable que nous proposons peut être vue comme une généralisation de l'idée du *graphe d'insertion* [3], plus générique et encapsulée par l'implémentation interne du domaine de la variable de séquence.

3.1 Notations

Les notations sont largement issues de [18] mais réintroduites par souci de clarté. Chaque lieu pouvant être visité est qualifié de *nœud*, et leur ensemble est décrit par $\mathcal{X}$. Une séquence sur $\mathcal{X}$ est notée $\overrightarrow{S}$ et l'ensemble de toutes les séquences de $\mathcal{X}$ par $\overrightarrow{\mathcal{P}}(\mathcal{X})$. La notation $p \overset{\overrightarrow{S}}{\prec} q$ signifie que le nœud p précède q dans $\overrightarrow{S}$, et $p \xrightarrow{\overrightarrow{S}} q$ que p précède directement q dans $\overrightarrow{S}$. Elles seront simplement écrites $p \prec q$ et $p \rightarrow q$ lorsque le contexte le permet.

Une séquence peut être agrandie par l'utilisation d'un opérateur $insertion(\overrightarrow{S}, p, q)$, avec $q \notin S, p \in S$, qui résulte en l'insertion de q juste après p dans la séquence. Plus formellement, assumons $\overrightarrow{S} = \overrightarrow{S}_1 \cdot p \cdot \overrightarrow{S}_2$. La super-séquence résultante de l'opération est $\overrightarrow{S}' = \overrightarrow{S}_1 \cdot p \cdot q \cdot \overrightarrow{S}_2$. Cette opération est également notée $\overrightarrow{S} \underset{(q,p)}{\Longrightarrow} \overrightarrow{S}'$.

Étant donné I, un ensemble de tuples (q, p), correspondant chacun à une insertion potentielle dans $\overrightarrow{S}$, $\overrightarrow{S} \underset{I}{\Longrightarrow} \overrightarrow{S}'$ signifie que $\overrightarrow{S}'$ peut être obtenue en appliquant une insertion de I sur $\overrightarrow{S}$: $\exists (p, q) \in I \mid \overrightarrow{S} \underset{(p,q)}{\Longrightarrow} \overrightarrow{S}'$.

Plus généralement, la *dérivation en zéro étape ou plus* est définie par $\overrightarrow{S} \underset{I}{\overset{*}{\Longrightarrow}} \overrightarrow{S}' \equiv \overrightarrow{S} = \overrightarrow{S}' \vee \left(\exists (p, q) \in I \mid \overrightarrow{S} \underset{(p,q)}{\Longrightarrow} \overrightarrow{S}'' \wedge \overrightarrow{S}'' \underset{I \setminus \{(p,q)\}}{\overset{*}{\Longrightarrow}} \overrightarrow{S}' \right)$. Notons que I peut contenir des tuples qui ne correspondent pas à une insertion possible dans, $\overrightarrow{S}$ mais à la place dans une insertion possible dans une super séquence de $\overrightarrow{S}$.

3.2 Domaine

Définition 1. *Le domaine d'une variable de séquence Sq est représenté par $\langle \overrightarrow{S}, I, P, E \rangle$, avec $\overrightarrow{S}$ un ensemble de nœuds ordonnancés correspondant à la séquence et formant un tour partiel, des points d'insertions $I \subseteq \mathcal{X} \times \mathcal{X}$ et deux sous-ensembles de nœuds $P, E \subseteq \mathcal{X}$ pour les nœuds potentiellement insérés et exclus de la séquence, respectivement. Le domaine de Sq, également noté $D(Sq)$, est défini comme $\langle \overrightarrow{S}, I, P, E \rangle \equiv \left\{ \overrightarrow{S}' \in \overrightarrow{\mathcal{P}}(P \cup S) \mid \overrightarrow{S} \underset{I}{\overset{*}{\Longrightarrow}} \overrightarrow{S}' \right\}$ et capture tous les dérivations possibles valides du tour partiel $\overrightarrow{S}$ en utilisant les insertions de I.*

À son initialisation, la séquence est composée d'un tour partiel de deux nœuds, $\alpha \cdot \omega$, pour le début α et la fin ω du parcours, et aucune insertion n'est permise après ω pour garantir que ω demeure le dernier nœud visité. P est donc égal à $\mathcal{X} \setminus \{\alpha, \omega\}$, $E = \phi$ et l'ensemble d'insertions est $I = \{(p, q) \in P \times \mathcal{X} \mid p \neq \omega\}$. Imposer un nœud de début et de fin dans la séquence permet une modélisation facile des problèmes où le trajet d'un véhicule doit finir à son point de départ ou à un autre endroit (α se trouve au même endroit que ω ou non) et évite à l'interface de programmation de faire face au cas spécial de séquences vides, requérant l'ajout d'un symbole fictif comme dans [18].

Nous maintenons une cohérence assez faible sur le domaine, facile à calculer et capturant les invariants suivants :

$$S \cup P \cup E = X \wedge S \cap P = S \cap E = P \cap E = \phi \quad (1)$$

$$\forall (p, q) \in I : q \in P \wedge p \notin E \quad (2)$$

$$\forall q \in P : \exists p \in S \cup P \mid (p, q) \in I \quad (3)$$

(1) Les nœuds de la séquence partielle S, de l'ensemble possible P et de l'ensemble exclu E forment une partition de $\mathcal{X}$; (2) les insertions valides sont constituées de nœuds possibles placés après des nœuds non exclus (qui ne sont pas forcément encore présents dans la séquence partielle) ; (3) un nœud possible peut toujours être inséré après un autre mœud. Cette cohérence ne détecte pas si tous les arcs de I sont déconnectés de $\overrightarrow{S}$ et devraient être exclus.

3.3 Implémentation et structures de données

L'implémentation du domaine $\langle \overrightarrow{S}, I, P, E \rangle$ doit être réversible pour des solutionneurs à trace comme MiniCP [14], et ses mises à jours et itérations aussi efficaces que possible.

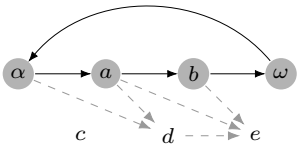
Le partitionnement entre les ensembles S, P, E est implémenté par un seul ensemble réversible [17], permettant la suppression et restauration en temps constant. Les points d'insertions I sont quant à eux partitionnés en un ensemble $I^x = \{p \in (S \cup P) : (p, x) \in I\}$ par nœud $x \in \mathcal{X}$, chacun composé des prédécesseurs valides du nœud x. Ils sont également implémentés par des ensembles réversibles. Les successeurs des nœuds sont quant eux accessibles et modifiables en tant constant via un tableau d'entiers réversibles. Les invariants maintenus par ces structures de données sont décrits dans les équations (4) à (7) et sont équivalents aux invariants (1) à (3). La Figure 1 illustre le domaine.

$$S \cup P \cup E = X \wedge S \cap P = S \cap E = P \cap E = \phi \quad (4)$$

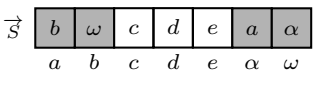
$$p \in E \implies I^p = \phi \wedge \forall x : p \notin I^x \quad (5)$$

$$p \in S \implies I^p = \phi \quad (6)$$

$$I^x = \phi \implies x \in S \vee x \in E \quad (7)$$



node x	I^x	$\lvert I^x \cap S \rvert$	$\lvert I^x \cap P \rvert$
a	ϕ	0	0
b	ϕ	0	0
c	ϕ	0	0
d	$\{\alpha, a\}$	2	0
e	$\{a, b, d\}$	2	1



$$P = \{d, e\}, E = \{c\}$$

FIGURE 1 – Variable de séquence, avec son ordonnancement $\overrightarrow{S} = \{\alpha, a, b, \omega\}$ et ses insertions potentielles (pointillées) pour les nœuds $x \in P$. Se trouve une table montrant les insertions des nœuds, les successeurs de la séquence (uniquement valide pour les nœuds $\in S$) ainsi que l'appartenance des nœuds à P et E.

4 Contraintes globales

Nous énumérons simplement quelques contraintes essentielles au PTV sans pour autant donner leur implémentation exacte, afin de conserver une taille raisonnable pour ce résumé. Les lecteurs souhaitant plus de détails peuvent se référer directement à l'implémentation [1] ou dans le futur à la version longue du papier, qui est en attente de relecture pour une autre conférence. Les contraintes globales que nous avons définies sont des contraintes de

Dépendance s'assurant qu'un ensemble de nœuds soient tous présents ou absents d'une séquence ;

Précédence veillant à ce qu'un ordonnancement précis de nœuds soit respecté dans la séquence ;

Disjointe garantissant qu'un nœud ne soit visité que par une seule séquence parmi un ensemble ;

Cumulative respectant une capacité maximale disponible, très similaire à l'approche de [18].

TempsDeTransition. Permet d'ajouter une dimension temporelle aux PTV. Elle est appliquée sur une séquence, une matrice de distance $tr \in R^{n \times n}$, une distance de parcours l et associe à chaque nœud $x \in \mathcal{X}$ une fenêtre de temps $début_x$ durant laquelle la visite du nœud doit se produire ainsi qu'une durée $durée_x$ pour y effectuer une action :

$$\texttt{TempsDeTransition}(Sq, [début], [durée], [[tr]], l) \equiv \left\{ \vec{S} \in D(Sq) \;\middle|\; \begin{pmatrix} \forall i,j \in \vec{S}, i \overset{\vec{S}}{\prec} j \implies \\ début_i + durée_i + tr_{i,j} \leq début_j \end{pmatrix} \right\} \quad (8)$$
$$l = \sum_{i,j \in \vec{S} \mid i \to j} tr_{i,j}$$

Nous considérons qu'il est possible d'atteindre un nœud avant le début de sa fenêtre de temps sans y commencer la tâche associée, expliquant l'emploi d'inégalités dans (8).

Filtrage Le pseudo-code pour le filtrage est présenté dans l'Algorithme 2. Nous commençons par mettre à jour les fenêtres de temps *début* des nœuds visités actuellement (ligne 2). Par la suite, nous calculons la longueur actuelle de la séquence et enlevons les insertions invalides : un prédécesseur qui ne permet pas d'atteindre le nœud dans sa fenêtre de temps (ligne 8), de rejoindre le successeur actuel dans la séquence (ligne 13) ou qui excéderait la longueur maximale permise (ligne 17). La complexité de ce filtrage est $\mathcal{O}(|P| \cdot |S|)$. En pratique, le filtrage s'exécute plus rapidement, car $I^x \cap S$ est récupéré en $\Theta(\min(|S|, |I^x|))$ dans notre implémentation. Comme nous ne raisonnons pas sur un ensemble de nœuds requis comme dans [18], nous nous débarrassons du problème NP-complet consistant à vérifier si un chemin valide visitant tous les nœuds requis existe.

Algorithme 1 : `TempsDeTransition`($Sq = \langle \vec{S} I, P, E \rangle, [début], [durée], [[tr]], l$)

```
1  for i ∈ S⃗ do
2  |  mise à jour de la fenêtre de temps début_i
3  longueur ← distance actuelle de la séquence
4  min (l) ← longueur
5  for x ∈ P do
6  |  for p ∈ I^x ∩ S do
7  |  |  arr_x ← min(début_p) + durée_p + tr_{p,x}
8  |  |  if arrivée_x > max(début_x) then
9  |  |  |  enlève p de I^x
10 |  |  else
11 |  |  |  q ← succ(Sq, p)
12 |  |  |  arr_q ← max (arr_x, min(début_x)) +
   |  |  |    durée_x + tr_{x,q}
13 |  |  |  if arr_q > max(début_q) then
14 |  |  |  |  enlève p de I^x
15 |  |  |  else
16 |  |  |  |  détour ← tr_{p,x} + tr_{x,q} − tr_{p,q}
17 |  |  |  |  if detour + longueur > l then
18 |  |  |  |  |  enlève p de I^x
```

5 Expériences

Notre variable permet de modéliser des problèmes de collecte et livraison (PCL), problèmes de transport de patients (PTP), ou encore le voyageur de commerce avec fenêtre de temps (PVCFT). Chacun de ces problèmes est modélisé avec une Variable de Séquence par véhicule ainsi qu'une contrainte `TempsDeTransition` sur ces variables. Dans le cadre du PCL et du PTP, une contrainte `Cumulative` est employée pour respecter la capacité maximale des véhicules.

Nos résultats avec une recherche par large voisinage sont présentés dans la Table 1 pour le PCL, la Table 2 pour le PTP et la Table 3 pour le PVCFT. Pour le PCL, nos résultats sont compétitifs avec l'état de l'art [8], avec un léger avantage sur les plus petites instances. Nos variables sont par ailleurs toujours plus performantes que celles de CP Optimizer (modèle issu de [18]), qui ne parvient pas toujours à fournir de solution améliorante. Pour le PTP, nous arrivons à améliorer les meilleurs résultats publiés[4]. Enfin, pour le PVCFT, notre implémentation est à même de battre l'état de l'art et trouver de nouvelles meilleures solutions pour le problème sur 32 instances issues de la suite standard de tests [13]. Nous ne montrons toutefois que les 10 nouvelles solutions sur le jeu d'instances *AFG* [2].

classe a		LNS-FFPA		Séquence		CPO	
m	n	Moyenne	Meilleur	Moyenne	Meilleur	Moyenne	Meilleur
3	24	191.76	191.40	**190.89**	**190.21**	196.11	196.00
4	36	**291.71**	**291.71**	294.72	292.72	318.97	318.97
5	48	308.95	306.97	**307.09**	**304.38**	327.37	327.00
6	72	532.55	524.97	**531.84**	**519.76**	579.79	579.77
7	72	**554.57**	550.42	554.65	**548.72**	614.02	614.00
8	108	**752.29**	**742.08**	794.86	755.00	924.04	923.86
9	96	**622.19**	614.65	625.68	**611.15**	740.26	740.26
10	144	**950.16**	**929.31**	1011.42	962.21	/	/
11	120	**699.32**	**687.99**	718.58	709.49	861.74	861.73
13	144	**878.33**	**864.81**	901.71	874.56	1042.82	1042.82
Moyenne		**578.18**	**570.43**	593.14	576.82	t/o	t/o

TABLE 1 – Valeurs objectives pour le PCL, lorsqu'une solution initiale est fournie. L'état de l'art (LNS-FFPA) et un modèle employant les variables de CP Optimizer (CPO) sont indiqués. / signifie qu'aucune solution améliorante n'a été trouvée. Dix essais ont été faits par instance et modèle.

Instances					SCHED+MSS	Séquence
Difficulté	Nom	$\|H\|$	$\|V\|$	$\|R\|$	Sol	Sol
Facile	RAND-E-8	32	12	128	**128**	**128**
Facile	RAND-E-9	36	14	144	142	**143**
Facile	RAND-E-10	40	16	160	**157**	156
Moyenne	RAND-M-8	64	8	128	83	**91**
Moyenne	RAND-M-9	72	8	144	81	**93**
Moyenne	RAND-M-10	80	9	160	99	**113**
Difficile	RAND-H-8	128	8	128	75	**87**
Difficile	RAND-H-9	144	8	144	72	**84**
Difficile	RAND-H-10	160	8	160	72	**84**

TABLE 2 – Résultats expérimentaux pour le PTP. $|H|$, $|V|$, $|R|$ sont le nombre d'hôpitaux, véhicules et requêtes, respectivement. L'objectif est le nombre de patients servis (Sol). SCHED+MSS est le meilleur modèle référencé [4]

Instance	Précédent	Nouvel	Temps [s]
rbg132.2	8200	8194	37.76
rbg132	8470	8468	0.76
rbg201a	12 967	12 948	152.53
rbg233.2	14 549	14 523	24.20
rbg092a	7160	7158	2.70
rbg152.3	9797	9796	0.41
rbg193.2	12 167	12 159	242.54
rbg193	12 547	12 538	55.57
rbg233	15 031	14 994	264.70
rbg172a	10 961	10 956	113.83

TABLE 3 – Solutions au PVCFT sur les instances AFG [2].

6 Conclusion

Nous avons présenté une version simplifiée de la Variable de Séquence introduite précédemment [18], une approche flexible pour la modélisation et la résolution de PTV. Nous avons détaillé son domaine et expliqué les contraintes globales qui peuvent y être appliquées. Nos résultats expérimentaux sur trois PTV montrent que nous sommes compétitifs avec les travaux similaires sur des variables de séquences, tout en étant suffisamment efficace pour trouver de nouvelles solutions améliorantes à un problème très étudié tel que le PVCFT. Nos prochains travaux sur cette approche se focaliseront sur son application à d'autres PTV, à de la planification ainsi qu'à l'amélioration de nos algorithmes.

Références

[1] *MiniCP Sequences - Anonymous GitHub*, Sep 2021. https://anonymous.4open.science/r/minicp-sequences-5EE3/README.md, [Online ; accessed 26. Feb. 2022].

[2] Ascheuer, Norbert: *Hamiltonian path problems in the online optimization of flexible manufacturing systems*. Thèse de doctorat, 1996.

[3] Bent, Russell et Pascal Van Hentenryck: *A two-stage hybrid local search for the vehicle routing problem with time windows*. Transportation Science, 38(4) :515–530, 2004.

[4] Cappart, Quentin, Charles Thomas, Pierre Schaus et Louis Martin Rousseau: *A Constraint Programming Approach for Solving Patient Transportation Problems*. Dans Hooker, John (rédacteur) : *Principles and Practice of Constraint Programming*, pages 490–506, Cham, 2018. Springer International Publishing, ISBN 978-3-319-98334-9.

[5] Center, IBM Knowledge: *Interval variable sequencing in CP Optimizer*, Mar 2021. https://www.ibm.com/docs/en/icos/12.9.0?topic=concepts-interval-variable-sequencing-in-cp-optimizer, [Online ; accessed 13. Jan. 2022].

[6] Center, IBM Knowledge: *Search API for scheduling in CP Optimizer*, Mar 2021. https://www.ibm.com/docs/en/icos/12.9.0?topic=c-search-api-scheduling-in-cp-optimizer#85, [Online ; accessed 13. Jan. 2022].

[7] Gervet, Carmen: *Interval Propagation to Reason about Sets : Definition and Implementation of a Practical Language*. Constraints, 1 :191–244, mars 1997.

[8] Jain, Siddhartha et Pascal Van Hentenryck: *Large neighborhood search for dial-a-ride problems*. Dans *International Conference on Principles and Practice of Constraint Programming*, pages 400–413. Springer, 2011.

[9] Laborie, Philippe et Jerome Rogerie: *Reasoning with Conditional Time-Intervals.* Dans *FLAIRS conference*, pages 555–560, 2008.

[10] Laborie, Philippe, Jerome Rogerie, Paul Shaw et Petr Vilím: *Reasoning with Conditional Time-Intervals. Part II : An Algebraical Model for Resources*. Dans *FLAIRS Conference*, 2009.

[11] Laborie, Philippe, Jérôme Rogerie, Paul Shaw et Petr Vilím: *IBM ILOG CP Optimizer for Scheduling*. Constraints, 23(2) :210–250, apr 2018, ISSN 1383-7133. https://doi.org/10.1007/s10601-018-9281-x.

[12] Liu, Chang, Dionne M. Aleman et J. Christopher Beck: *Modelling and Solving the Senior Transportation Problem*. Dans Hoeve, Willem Jan van (rédacteur) : *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 412–428, Cham, 2018. Springer International Publishing, ISBN 978-3-319-93031-2.

[13] López-Ibáñez, Manuel: *Instances for the TSPTW*, Sep 2020. https://lopez-ibanez.eu/tsptw-instances, [Online ; accessed 15. Feb. 2022].

[14] Michel, L., P. Schaus et P. Van Hentenryck: *MiniCP : a lightweight solver for constraint programming*. Mathematical Programming Computation, 13(1) :133–184, 2021. https://doi.org/10.1007/s12532-020-00190-7.

[15] Perron, Laurent et Vincent Furnon: *OR-Tools*. https://developers.google.com/optimization/.

[16] Perron, Laurent et Vincent Furnon: *OR-Tools Sequence Var*. https://developers.google.com/optimization/reference/constraint_solver/constraint_solver/SequenceVar.

[17] Saint-Marcq, Vianney le Clément de, Pierre Schaus, Christine Solnon et Christophe Lecoutre: *Sparse-sets for domain implementation*. Dans *CP workshop on Techniques foR Implementing Constraint programming Systems (TRICS)*, pages 1–10, 2013.

[18] Thomas, Charles, Roger Kameugne et Pierre Schaus: *Insertion Sequence Variables for Hybrid Routing and Scheduling Problems*. Dans *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 457–474. Springer, 2020.

[19] Toth, Paolo et Daniele Vigo: *The vehicle routing problem*. SIAM, 2002.

Un Algorithme Parallèle pour le Filtrage GAC de Alldifferent

Wijnand Suijlen Félix de Framond[†] Arnaud Lallouet
Antoine Petitet
Huawei Technologies France, Paris Research Center, CSI

{wijnand.suijlen, arnaud.lallouet, antoine.petitet}@huawei.com
[†]fel2fram@gmail.com

Résumé

L'article [1] propose une version parallèle de l'algorithme de filtrage GAC de Alldifferent dû à Régin [2]. Il le fait en utilisant un algorithme parallèle de recherche dans un graphe pour trouver un couplage maximal et trouver ses composantes fortement connexes. Des expériences résolvant un gros problème N-reines ou un problème d'ordonnancement avec contraintes de ressources montrent que le filtrage GAC peut être accéléré de manière significative sur un système de mémoire partagée à 64 cœurs et sur un système à mémoire distribuée à 200 cœurs

Abstract

The paper [1] parallelizes the Alldifferent generalized arc-consistent (GAC) filtering algorithm by Régin [2]. It does so by using a parallel graph search algorithm for finding a maximum matching and finding the strongly connected components. Experiments solving a large N-queens problem or a resource constrained scheduling problem show that generalized arc-consistent filtering can be significantly sped-up on a 64-core shared-memory system and on a 200-core distributed-memory system.

1 Introduction

Alldifferent est l'une des contraintes les plus utiles et les plus largement utilisées. Elle assure que les variables d'un vecteur prendront des valeurs différentes. La contrainte Alldifferent(x_1, x_2, x_3) est équivalente à la conjonction $x_1 \neq x_2$, $x_2 \neq x_3$, et $x_1 \neq x_3$. Ces conditions se posent souvent dans les problèmes d'optimisation où les ressources sont attribué exclusivement.

Régin [2] décrit un filtrage GAC qui opère sur le graphe biparti reliant les variables et l'ensemble des valeurs du domaine. Il peut être reformulé en trois grandes étapes : trouver un couplage maximal (CM) pour obtenir un support, interpréter ce couplage comme un flot et calculer son résidu, trouver ses composantes fortement connexes (CFC), et enfin enlever les arcs inter-composantes. Cet algorithme est une contribution majeure à la programmation par contraintes (PPC) et a ouvert la voie à une vaste littérature sur les contraintes globales [3]. Pourtant, tandis que des travaux antérieurs ont développé des optimisations, tous considèrent seulement une approche séquentielle ou un environnement parallèle très limité, ce qui pose la question de savoir ce qu'un environnement parallèle plus général peut ajouter, se concentrant uniquement sur l'algorithme de filtrage lui-même.

2 Algorithme

Cet article propose de paralléliser les deux composantes principales de l'algorithme : trouver le CM et des CFCs. Les deux peuvent être exprimés en termes de *requêtes d'accessibilité* qui peuvent être parallélisées. Pour trouver le CM, nous utilisons l'algorithme de Ford-Fulkerson, qui repose sur l'identification des chemins augmentants, ce qui constitue un problème d'accessibilité. Pour trouver des CFCSs, l'appartenance d'un sommet au CFC d'un autre sommet peut être décidée par des requêtes d'accessibilité dans le sens avant et arrière des arcs. Ensuite, avec la technique diviser pour régner, toutes les CFCs d'un graphe sont énumérées en supprimant itérativement les CFC découvertes.

Notre première version de l'algorithme parallèle utilise un algorithme de recherche en largeur (BFS) à source unique requêtes d'accessibilité. Ce BFS comporte deux files d'attente. Au début de la recherche, le sommet racine est placé dans la première file du processeur qui possède ce sommet. Ensuite, tous les processeurs s'engagent dans un calcul collectif. Chaque processeur visite tous les sommets de sa file en les choisissant un par un. Tous les voisins des sommets visités sont ensuite placés dans une seconde file avec leur parent. Lorsque tous les sommets de la première file d'attente sont traités, les processeurs envoient toutes les paires de sommets enfants-parents au processeur qui possède l'enfant. Cette procédure continue jusqu'à ce que tous les sommets soient visités ou quand un sommet cible a été visité. La figure 1 illustre le flux de données de cette procédure.

Nous proposons ensuite deux optimisations. La première remplace BFS par une traversée qui prioritise les sommets stockés localement, que nous appelons *Local-First Search* (LFS). Son flux de données est illustré en Figure 2. La seconde propose d'effectuer des parcours multi-sources qui calculent une forêt couvrante au lieu d'un arbre pour chaque composante. Ceci permet de trouver plusieurs chemins augmentants aux sommets disjoints et accélère la découverte du couplage maximal.

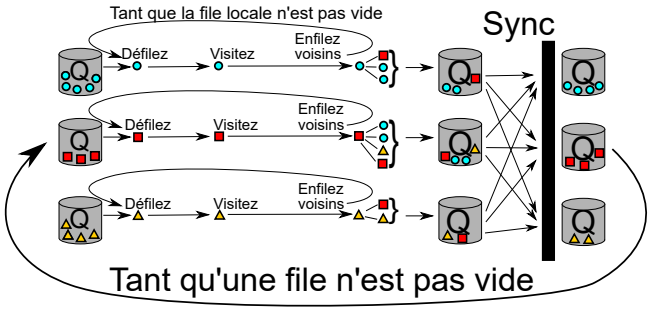


FIGURE 1 – Flux de données dans le BFS parallèle

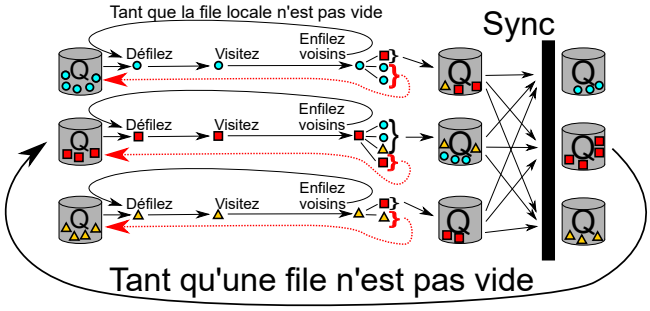


FIGURE 2 – Flux de données dans le LFS parallèle

3 Évaluation Expérimentale

Nous comparons le temps total de l'algorithme parallèle et ses optimisations à l'algorithme séquentiel en ce qui concerne trois paramètres différents d'un programme de contraintes donné : la taille totale du problème, la cardinalité du domaine, et le nombre de backtracks. Dans un problème de contraintes satisfiable, la cardinalité moyenne du domaine est proportionnelle au degré moyen des sommets dans le graphe variables-valeurs.

Pour nos expériences, nous avons choisi le problème des N-reines pour évaluer le temps de propagation en fonction de la taille totale du problème, un problème de comptage des N-reines pour évaluer la vitesse des retours arrières, et un problème d'ordonnancement pour évaluer le temps d'exécution en fonction du degré moyen. Comme machines de test, nous utilisons un système de mémoire partagée AMD Epyc Rome avec 64 cœurs (appelé Epyc64), et une petite grappe de serveurs avec un total de 200 cœurs basés sur Intel Ivybridge (appelée Ivy200) et connecté à EDR Infiniband.

Nos algorithmes sont intégrés dans HPCS, un solveur parallèle de contraintes développé en interne. Pour faciliter la comparaison, nous avons exécuté les mêmes problèmes dans Gecode 6.2.0. Sur tous les problèmes, HPCS et Gecode sont chargés d'explorer l'arbre de recherche exactement de la même manière. Les problèmes de N-reines sont traités par les deux avec le même nombre d'appels des propagateurs et le même nombre de retours arrières.

La propagation dans Gecode est la plus rapide pour les problèmes de très petite taille. À partir de quelques centaines de reines, le HPCS séquentiel est le plus rapide, jusqu'à 1200 dames sur Epyc64 et 4800 dames sur Ivy200. Après cela, l'algorithme basé sur LFS aux sources multiple est le plus rapide, par plus d'un facteur 9x sur 9000 reines sur Epyc64 et plus d'un facteur 3x sur Ivy200. Le goulet d'étranglement des performances dans l'algorithme parallèle est la communication inter-processus avec en particulier le délai de synchronisation qui prend plus de 100 fois plus de temps sur Ivy200.

L'algorithme parallèle doit se synchroniser au moins une fois à chaque tour de propagation. Cela ralentit la recherche avec backtrack, qui doit exécuter successivement le propagateur à plusieurs reprises tout en mettant à peine à jour le graphe résiduel. Pour le comptage de solutions sur le problème des 2400 reines sur Epyc64, Gecode est capable de traiter 25000 backtracks par seconde, HPCS séquentiel 1300, tandis que le HPCS parallèle n'en fait que 17. Sur Ivy200 avec 6000 reines, HPCS séquentiel exécute 6 backtracks par seconde, tandis que les versions parallèles ne dépassent pas 1 par seconde.

Le degré moyen ne détermine pas seulement le nombre d'arêtes du graphe variables-valeurs, mais il est également corrélé avec son diamètre. Le diamètre n'est pas pertinent pour les algorithmes séquentiels, qui fonctionnent le mieux sur les graphes avec le moins d'arêtes possible. Par contre, nos algorithmes parallèles bénéficient d'un diamètre plus petit, parce que le parcours nécessite moins de synchronisations pour parcourir le graphe entier. Cela conduit à la situation extrême où le HPCS parallèle résout le problème d'ordonnancement avec un graphe entièrement dense, qui a un diamètre de 1, dans à peu près le même temps que le problème d'ordonnancement avec seulement 10% des arêtes, tandis qu'un problème avec seulement 50% des arêtes est résolu plus lentement. Cette situation se produit sur les deux machines de test.

Références

[1] W. J.Suijlen, F. de Framond, A. Lallouet, and A. Petitet. A parallel algorithm for GAC filtering of the Alldifferent constraintt. In P. Schaus, editor, *Proc. 19th Int. Conf. Integr. Constraint Programm., Artif. Intell., and Operations Research CPAIOR 2022, June 20-23, 2022, Los Angeles, USA*, 2022.

[2] Jean-Charles Régin. A filtering algorithm for constraints of difference in CSPs. In Barbara Hayes-Roth and Richard E. Korf, editors, *Proc. 12th Nat. Conf. on Artif. Intell., Seattle, WA, USA, July 31 - August 4, 1994, Volume 1*, pages 362–367. AAAI Press / The MIT Press, 1994.

[3] W.-J. van Hoeve and I. Katriel. Global constraints. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 169–208. Elsevier, 2006.

Étude de la contrainte de produit

Steve Malalel, Victor Jung, Jean-Charles Régin, Marie Pelleau

Université Côte d'Azur, CNRS, I3S

{prénom.nom}@univ-cotedazur.fr

Résumé

La contrainte de produit assure que le produit de variables sera plus grand qu'une certaine valeur donnée. En d'autres termes, on souhaite satisfaire la contrainte $\Pi_{i=1}^{n} x_i \geq w$. Avec l'émergence de problèmes stochastiques, cette contrainte apparaît de plus en plus fréquemment en pratique. Les variables sont le plus souvent des variables de probabilité qui représentent la probabilité qu'un événement arrive, et la borne minimum est la probabilité minimum qui doit être satisfaite. Cela est dans la majorité des cas fait pour garantir un certain niveau de sécurité ou une certaine qualité de service. Pour appliquer cette contrainte, il est tentant comme le proposent de nombreux auteurs de prendre le logarithme de la somme et de la borne afin de transformer le produit en une somme. Dans cet article nous montrons que cette idée créé des problèmes et empêche un calcul exact. Nous proposons et comparons différentes représentations permettant de calculer exactement l'ensemble des solutions de ce problème ou jusqu'à atteindre une certaine précision. Nous donnons aussi une méthode efficace pour représenter cette contrainte par un Diagramme de Décision à valeurs Multiples (MDD) afin de pouvoir combiner cette contrainte avec d'autres MDDs.

Mots-clés

Contrainte de produit, Diagramme de décision à valeurs multiples, Variables flottantes, Variables décimales, Probabilité, Relâchement

Abstract

The product constraint ensures that the product of some variables will be greater than a given value, that is $\Pi_{i=1}^{n} x_i \geq w$. With the emergence of stochastic problems, this constraint appears more and more frequently in practice. The variables are most often probability variables that represent the probability that an event will occur and the minimum bound is the minimum probability that must be satisfied. This is often done to guarantee a certain level of security or a certain quality of service. To deal with this constraint, it is tempting as proposed by many authors to take the logarithm of the sum and the bound in order to transform the product into a sum. In this article we show that this idea creates many problems and forbids an exact calculation. We propose and compare different representations allowing to compute the set of solutions of this problem exactly or up to a certain precision. We also give an efficient method to represent that constraint by a Multi-valued Decision Diagram (MDD) in order to combine this constraint with some others MDDs.

Keywords

Product constraint, Multi-valued decision diagram, Float variables, Decimal variables, Probability, Relaxation

1 Introduction

De plus en plus de problèmes impliquent des données aléatoires associées à des probabilités. Pour des raisons de qualité de service et de sécurité, il est souvent imposé d'associer n'importe quelle solution à une probabilité minimum. Ce type de problème se modélise en définissant pour chaque variable x représentant des valeurs aléatoires, une variable p_x qui représente les probabilités de ces valeurs. Puis, les variables x et p_x sont liées ensemble ($x = a \Leftrightarrow p_x = p(a)$) et la contrainte $\Pi_{i=1}^{n} p_{x_i} \geq w$ est ajoutée au modèle afin de garantir que chaque solution sera associée à une probabilité plus grande qu'une valeur donnée.

Nous nous intéresserons principalement à la définition de cette contrainte sous forme de diagramme de décision à valeurs multiples (Multi-valued Decision Diagram ou MDD) [Kam and Brayton, 1990, Srinivasan et al., 1990, Bergman et al., 2016] comme cela est déjà fait pour la contrainte de somme. Nous supposons que les valeurs des variables sont des nombres décimaux avec une précision donnée.

Habituellement, la contrainte de produit est modélisée en prenant les logarithmes des valeurs concernées, puis en la transformant en contrainte de somme $\sum_{i=1}^{n} \log(p_{x_i}) \geq \log(w)$. Cela semble plus adapté car le produit de variable est limité dans les solveurs de programmation par contrainte à cause des dépassements (*overflows*). Cependant l'utilisation de logarithme a lui aussi un désavantage majeur : nous perdons la possibilité de faire des calculs exacts. En effet le logarithme d'un nombre ne peut pas être représenté de manière exacte dans un ordinateur car il s'agit d'un nombre transcendant. Les nombres flottants doivent donc être utilisés et les erreurs d'arrondi de cette représentation doivent être gérées.

Cet article introduit brièvement ce qu'est un MDD ainsi que la méthode de construction pour la contrainte qu'il doit représenter. Puis nous présentons les différentes méthodes étudiées pour définir le MDD de la contrainte de produit. Toutes ces méthodes ont été définies de manière à garantir

la complétude, c'est-à-dire que toutes les solutions soient représentées par l'ensemble des solutions du MDD. Pour cela elles ont aussi été définies à l'aide d'entiers pour représenter les nombres à virgule selon une précision δ. Par exemple avec $\delta = 4$, le nombre décimal 0.98 devient l'entier 9800.

2 Méthodes

La première méthode étudiée sert de cas de comparaison ; elle se base sur la somme des logarithmes habituellement utilisée pour résoudre ce problème. Elle a aussi été définie à l'aide d'entiers en utilisant la méthode de calcul chiffre par chiffre du logarithme [Goldberg, 2006]. Il est donc possible pour ces deux cas de faire une somme de logarithmes avec une précision λ donnée (c'est-à-dire avec λ chiffres après la virgule). Cela permet d'établir un compromis entre temps, mémoire et exactitude.

La seconde calcule le MDD exact du produit des variables, garantissant que l'ensemble des solutions est à la fois complet et correct conformément aux données. Cela nécessite d'utiliser une structure de données capable de représenter de très grands nombres, ainsi qu'une grande capacité de mémoire.

Nous proposons ensuite de construire le MDD de la précédente méthode de manière relâchée selon une certaine précision. Ce relâchement est effectué en faisant des multiplications relâchées, c'est-à-dire des multiplications qui ne prennent en compte que les ϵ premiers chiffres après le calcul. On note cette multiplication $\times^\epsilon$ tel que $\lceil x \times^\epsilon y \rceil = \lceil (x \times y)/10^\epsilon \rceil$. Par exemple $\lceil 9800 \times^4 9780 \rceil = 9585$. Cela permet de faire le même type de compromis que pour la méthode avec le logarithme.

Enfin nous présentons une méthode tirant avantage de ces relâchements pour construire de façon incrémentale un MDD par des itérations successives. Chaque itération correspond à une précision plus grande que celle de l'itération précédente. L'idée est d'éviter de considérer les parties du MDD qui seront toujours satisfaites quand la précision du calcul augmente. Par exemple, quelque soit la précision ajoutée (le nombre de décimales prises en compte), nous aurons toujours $(0.95... \times 0.95...) > 0.9$.

Cette méthode repose sur deux notions, celle de solution *incertaine* et celle d'arc *suspicieux*. Une solution est dite *certaine* si on peut dire qu'elle est valide à partir d'une précision ϵ donnée. Par opposition, une solution est dite *incertaine* si on ne peut déterminer si elle est valide à la précision actuelle ϵ. Un arc est dit *suspicieux* s'il appartient à au moins une solution *incertaine*.

Il s'agit alors d'identifier et extraire les arcs (et donc également de nœuds) dits *suspicieux*. Les arcs et noeuds appartenant à au moins une solution incertaine sont conservés dans un MDD appelé MDD_M. Ce MDD simplifié est la base pour la prochaine itération, où la précision sera plus élevée. Les arcs sont identifiés grâce à une propagation de propriétés à travers le MDD [Jung and Régin, 2021]. L'algorithme se termine lorsque le MDD extrait est vide ou si la précision maximale est atteinte. À chaque itération, on accumule les solutions *certaines* dans un MDD à part noté MDD_S. Notre algorithme peut renvoyer soit MDD_S contenant uniquement des solutions (correction), soit l'union de MDD_S et MDD_M contenant toutes les solutions (complétude). Notons que si MDD_M est vide notre méthode est à la fois correcte et complète.

3 Résultats

Diverses expériences ont été effectuées sur ces différentes méthodes à l'aide de données générées de manière semi-aléatoire. Les résultats mettent principalement en avant les désavantages de la méthode exacte en terme de temps de calcul et de mémoire utilisée, mais aussi les avantages de la méthode incrémentale sur ces mêmes aspects tout en ayant un ensemble de solutions très proche, si non égal, à l'ensemble de solutions du MDD exact.

4 Conclusion

Dans ce papier différentes méthodes ont été présentées et testées afin de construire au mieux la contrainte de produit. Il a été montré que la méthode relâchée permet d'obtenir des résultats précis (de manière proportionnelle) dans un temps raisonnable par rapport à un produit exact. Une méthode incrémentale utilisant efficacement ce relâchement a ensuite été élaborée permettant d'allier temps, mémoire et précision. Il pourrait être intéressant pour de futurs travaux de voir comment les résultats obtenus évolueraient si les probabilités étaient remplacées par des intervalles de probabilités. Un autre développement intéressant serait d'appliquer de manière similaire la méthode incrementale sur d'autres contraintes.

Références

[Bergman et al., 2016] Bergman, D., Ciré, A. A., van Hoeve, W., and Hooker, J. N. (2016). *Decision Diagrams for Optimization*. Artificial Intelligence : Foundations, Theory, and Algorithms. Springer.

[Goldberg, 2006] Goldberg, M. (2006). Computing logarithms digit-by-digit. *International Journal of Mathematical Education in Science and Technology*, 37(1) :109–114.

[Jung and Régin, 2021] Jung, V. and Régin, J.-C. (2021). Checking constraint satisfaction. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 332–347. Springer.

[Kam and Brayton, 1990] Kam, T. and Brayton, R. K. (1990). Multi-valued decision diagrams. Technical Report UCB/ERL M90/125, EECS Department, University of California, Berkeley.

[Srinivasan et al., 1990] Srinivasan, A., Ham, T., Malik, S., and Brayton, R. K. (1990). Algorithms for discrete function manipulation. In *1990 IEEE International Conference on Computer-Aided Design. Digest of Technical Papers*, pages 92–95.

Session 3 : Apprentissage, fouille de données

Réseaux de neurones antagonistes sur les graphes pour l'apprentissage d'une heuristique SAT

Thomas Fournier, Arnaud Lallouet, Télio Cropsal, Gaël Glorian,
Alexandre Papadopoulos, Antoine Petitet et Wijnand Suijlen

Huawei Technologies Ltd, Paris Research Center, CSI Paris

contact: arnaud.lallouet@huawei.com

Résumé

La question de savoir s'il est possible d'apprendre à partir de la résolution de problèmes existants a été abordée depuis longtemps par des solveurs portfolio où la meilleure heuristique est choisie par apprentissage automatique à partir de caractéristiques créées manuellement, et plus récemment par l'apprentissage profond sur les graphes (GNNs). Avec ce papier, nous nous inscrivons dans cette voie en proposant une nouvelle heuristique basée sur l'apprentissage par renforcement profond avec des récompenses antagonistes. Nous montrons que notre méthode réduit le nombre d'échecs pour atteindre une première solution de plus de 50% en comparaison avec MiniSat.

Mots-clés

Heuristique, Apprentissage automatique, Apprentissage profond, Apprentissage par renforcement, Graph Neural Networks.

Abstract

The question of whether it is possible to learn from solving existing problems has been addressed for a long time by portfolio solvers where the best heuristic is chosen by Machine Learning from handcrafted features, and more recently with Deep Learning by embedding this knowledge into a Graph Neural Network (GNN). In this paper, we build upon the latter category by proposing a new heuristic based on Deep Reinforcement Learning using two GNNs with opposing rewards. We show that our method reduces the number of fails to get the first solution by more than 50% compared to MiniSat.

Keywords

Heuristic, Machine Learning, Deep Learning, Reinforcement Learning, Graph Neural Networks.

1 Introduction

Des challenges considérables pour l'apprentissage automatique.
L'apprentissage automatique est difficile à appliquer aux problèmes combinatoires pour quatre raisons principales. Premièrement, les problèmes combinatoires n'ont pas de taille fixe. Ils sont représentés par un graphe dont la taille dépend du problème et des données qui vont avec. Les techniques classiques d'apprentissage automatique, qui fonctionnent sur des données tabulaires, sont difficiles à adapter à ce nouveau contexte. Deuxièmement, il existe une grande variabilité dans l'espace des solutions vis-à-vis de l'entrée. Deux modèles qui diffèrent en taille et en structure peuvent avoir les mêmes solutions et à l'inverse deux modèles syntaxiquement proches peuvent avoir des solutions très différentes. En particulier, l'ajout d'une unique contrainte non satisfiable peut faire passer le nombre de solutions d'un million à zéro. Troisièmement, la projection de l'ensemble des solutions sur un sous-ensemble des variables peut être le domaine entier, ce qui rend la classification des valeurs du domaine très inefficace, en d'autres termes, il est possible d'avoir une distribution uniforme des solutions dans l'espace de recherche. Finalement, l'expression du problème et les données sont par nature discrètes, ce qui ne permet pas de facilement exploiter les directions de descente induites par les gradients. Pour ces raisons, il n'est pas trivial d'établir une technique directe d'apprentissage de bout-en-bout pour résoudre des problèmes combinatoires.

Des solveurs spécialisés.
En revanche, les solveurs SAT [3] ou PC [15] sont capables de traiter de tels problèmes grâce à une combinaison de recherche et de propagation, mais l'heuristique nécessaire est souvent difficile à trouver. C'est un champ de recherche actif qui a notamment permis l'émergence des compteurs d'activité, de l'apprentissage des clauses avec les « nogoods », des redémarrages [14, 6]. Mais malgré tout, il est difficile d'exploiter les connaissances acquises lors de la résolution d'autres problèmes. Deux cadres existent pour permettre de réutiliser les informations apprises : les solveurs portfolio [21, 7] qui utilisent l'apprentissage automatique pour reconnaître les caractéristiques du problème et pour choisir statistiquement le solveur le plus approprié ainsi que les solveurs incrémentaux [8] qui peuvent conserver des connaissances exactes acquises lors de la résolution d'un sous-ensemble de clauses.

L'essor des GNNs.
Des travaux récents ont apporté un nouvel éclairage sur l'utilisation de l'apprentissage automatique pour acquérir des connaissances indépendantes de la résolution d'un

unique problème, en particulier en utilisant les GNNs [16, 2]. Une approche remarquable est celle de NeuroSAT [17] où un GNN est spécifiquement entrainé pour prédire la satisfiabilité d'une instance. L'idée principale de l'application de l'apprentissage par renforcement [19] à un problème SAT est de considérer le solveur comme un système contrôlable et d'apprendre une politique la plus optimale possible pour interagir avec cet environnement. Plusieurs articles ont exploré cette direction, comme [20, 18, 11, 4]. La difficulté pour intégrer le Machine Learning dans le cœur du solveur à été reconnu par [5] et a motivé le développement d'un solveur dédié permettant l'interaction entre le ML et la PPC.

Notre méthode antagoniste.
Dans ce travail préliminaire, nous présentons une nouvelle façon de construire une heuristique pour le problème SAT en utilisant les GNNs et l'apprentissage par renforcement, basée sur l'opposition entre le comportement attendu pour l'heuristique de choix de variable et celui de l'heuristique du choix de valeur. L'heuristique de choix de variable a un schéma de récompense qui vise à fermer la recherche dès que possible. Elle est entrainée conjointement avec une heuristique de choix de valeur qui cherche à ouvrir la recherche vers le plus de solutions possibles. Ces deux heuristiques sont implémentées avec des GNNs et sont entrainées en utilisant la bibliothèque SATLIB [9]. Nos expériences montrent que les connaissances structurelles et contextuelles acquises sont capable de réduire significativement le nombre de retours en arrière pour trouver la première solution lorsqu'elles sont utilisées sur des nouveaux problèmes non vus lors de la phase d'entrainement.

2 Notions préliminaires

Satisfiabilité booléenne
Soit V un ensemble des variables booléennes. Un *littéral* est soit une variable $x \in V$ ou la négation $\neg x$ d'une variable. Une *clause* est un ensemble de littéraux c interprété comme une disjunction. Une *formule SAT* en CNF est un ensemble de clauses F interprété comme une conjunction. Le problème SAT consiste à trouver une affectation M des variables de V (aussi appelée *modèle*) tel que la formule F est satisfaite, ou la preuve qu'une telle affectation n'existe pas. SAT a été le premier problème à être prouvé comme NP-complet [3], même si on se restreint aux problèmes 3SAT, qui ont des clauses avec au plus 3 littéraux. Les solveurs CDCL sont basés sur une recherche heuristique avec retour en arrière, des inférences et de l'apprentissage [3]. Il est à noter que le mot apprentissage est utilisé dans un sens différent de celui du Machine Learning car il n'implique pas de généralisation.

L'inférence est réalisée par la *propagation unitaire* qui donne à une variable la polarité de son littéral quand il apparaît dans une clause singleton. L'inférence crée aussi le *graphe d'implication* qui connecte et enregistre les littéraux propagés. L'apprentissage se fait lors d'un conflict en appliquant une résolution sur le graphe d'implication jusqu'à ce qu'une condition soit vérifiée (usuellement la découverte du premier UIP). Il en est déduit une clause redondante qui va empêcher que ce conflit apparaisse une seconde fois au cours de la recherche. Après que l'inférence a atteint un point fixe, le solveur sélectionne une variable et une valeur et éxécute une *décision* en ajoutant cette hypothèse au graphe d'implication. L'heuristique classique VSIDS (Variable State Independent Decaying Sum) [14] donne un score à chaque variable qui combine une information statistique sur le problème, une mise-à-jour avec de l'apprentissage par renforcement après chaque conflit et un taux d'escompte pour permettre de se concentrer sur les parties difficiles du problème après un *redémarrage*. Il en va de même pour de nombreuses heuristiques d'apprentissage online telles que CHB ou LRB [13, 12] La partie apprentissage est très efficace, mais le score initial est générique et l'heuristique a besoin d'un échauffement pour dévoiler son potentiel. Un *état* s peut-être défini comme un ensemble de littéraux affectés à toute étape de la recherche.

Apprentissage par renforcement
Un système contrôlable peut-être modélisé comme un *Processus de Décision Markovien* (PDM), défini comme un tuple (S, A, T, R, γ), où S est un ensemble d'états, A un ensemble d'actions, $T : S \times A \times S \rightarrow [0, 1]$ une fonction de transition probabiliste telle que $\forall s, s' \in S, \forall a \in A$, $T(s, a, s')$ est la probabilité d'atteindre l'état s' quand on réalise l'action a dans l'état s, $R : S \times A \rightarrow \mathbb{R}$ est une fonction récompense et $\gamma \in [0, 1)$ est le taux d'escompte. Un PDM est *déterministe* si l'état obtenu après l'application d'une action est complétement determiné par l'action et l'état de départ. Dans ce contexte, on peut écrire les fonctions de transition sous la forme $T : S \times A \rightarrow S$. Une politique $\pi : S \times A \rightarrow [0, 1]$ est une fonction qui définit une distribution de probabilité pour chaque état sur l'ensemble des actions. Une politique déterministe est une fonction $\pi : S \rightarrow A$. Une *exécution* de la politique π en partant de l'état s_0 est une séquence d'actions $[a_0, a_1, \ldots, a_n]$ telle que $s_{i+1} = \pi(s_i, a)$ si la politique est déterministe.
Soit un PDM M et une politique π, on peut récursivement definir le gain attendu d'une action en fonction d'un état, d'une action et d'une politique aussi appelé *Q-value*, tel que :

$$Q^{\pi}(s, a) =$$
$$R(s, a) + \sum_{s' \in S} [T(s, a, s') \times \gamma \sum_{a' \in A} (\pi(s', a') \times Q^{\pi}(s', a'))]$$

Dans le cas déterministe, on obtient :

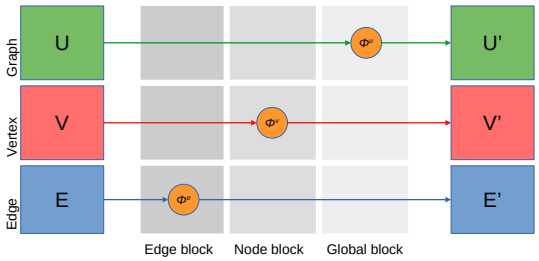
$$Q^{\pi}(s, a) = R(s, a) + \gamma \times Q^{\pi}(s', \pi(s')))$$
$$s' = T(s, a)$$

L'apprentissage par renforcement a pour objectif d'apprendre une politique optimale π^*, qui maximise la somme des récompenses jusqu'à un état terminal :

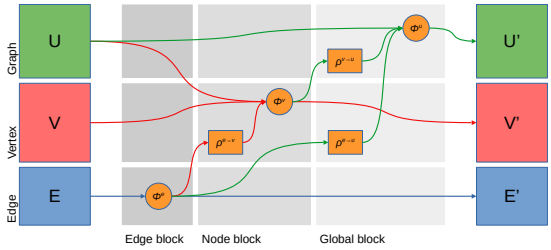
$$\forall (s, a) \in S \times A, \pi^* = argmax_{\pi} Q^{\pi}(s, a)$$

Une condition nécessaire pour l'optimalité d'une politique est de vérifier l'équation de Bellman, qui stipule que pour la politique optimale :

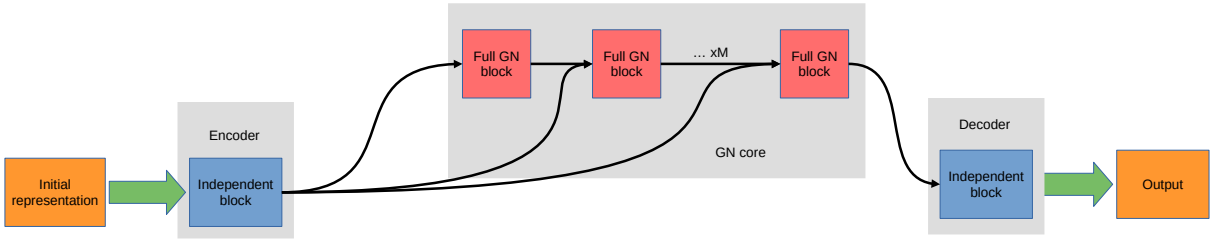
$$Q^{\pi^*}(s, a) = R(s, a) + \gamma \times \max_{a' \in A} Q^{\pi^*}(\pi^*(s, a), a')$$



(a) Bloc indépendant



(b) Bloc complet



(c) Architecture GNN

FIGURE 1 – Graph Neural Network

La programmation dynamique peut-être utilisée pour calculer π^* en calculant récursivement les Q-values des états résultants, pour toutes les actions. Mais, l'aspect combinatoire induit par le grand nombre d'actions dans chaque état rend le problème intraitable en pratique. Le Deep Q-Learning (DQN) construit une approximation des Q-values grâce aux épisodes générés et à l'équation de Bellman en tant que critère. Par la suite, nous allons utiliser un modèle paramétrisé Q_θ qui va être entrainé pour satisfaire l'équation de Bellmann. La politique induite par cette approximation est donc :

$$\hat{\pi}(s) = argmax_{a \in A} Q_\theta(s, a)$$

GNNs

Ils sont conçus pour apprendre à partir de représentations sous forme de graphe. On utilise le cadre général de [2]. Pour un graphe $G = (V, E)$ avec $E \subseteq V^2$, on considère trois plongements (embeddings) : $\mathbf{e} \in \mathbb{R}^e$ pour chaque arc, $\mathbf{v} \in \mathbb{R}^v$ pour chaque nœud et $\mathbf{u} \in \mathbb{R}^u$ pour le graphe dans sa totalité, où les paramètres e, v et u sont constants. Un GNN est composé de trois fonctions de mise à jour ϕ^e, ϕ^v, ϕ^u qui calculent les nouveaux embeddings à partir des précédents et de 3 fonctions d'agrégation $\rho^{e \to v}, \rho^{e \to u}, \rho^{v \to u}$ qui permettent à la fois de résonner sur la topologie du graphe et d'utiliser des fonctions de mise à jour d'arité fixe. Comme la plupart des approches, nous utilisons la somme comme fonction d'agrégation. Les fonctions ϕ et ρ sont les composantes principales pour constuire les blocs indépendants et complets comme présentés dans [2] et représentés avec les figures 1(a) et 1(b). Par la suite les blocs sont combinés pour former le GNN décrit avec la figure 1(c). Un bloc indépendant s'appele un *encodeur* et est utilisé pour prépaper le premier embedding. Puis cet embedding est connecté à un bloc complet, qui va être répété plusieurs fois. L'embedding final est utilisé comme entrée d'un dernier bloc indépendant, le *décodeur* qui produit l'embedding de sortie.

3 Apprentissage de l'heuristique de branchement

Les techniques d'apprentissage profond sont basées sur la notion d'embedding, qui est une représentation multidimensionnelle des données. Dans le cas des problèmes SAT, et puisque le problème est représenté sous forme de graphe, nous souhaitons un embedding du graphe. Nous nous appuyons sur le cadre introduit dans [11] qui utilise un GNN général [2] pour produire un tel embedding.

Un problème SAT est construit sur un ensemble fini B des variables booléennes définissant un ensemble L de littéraux $L = B \cup \neg B$ où $\neg B = \{\neg b \mid b \in B\}$. Un modèle est une application $M : B \to \mathbb{B}$. Un état est une représentation du problème courant à résoudre, à savoir les variables booléennes et les clauses. Un problème SAT peut être représenté comme un graphe biparti $G = (V, E)$ où un nœud est créé pour chaque variable et chaque clause : $V = B \cup C$, et pour chaque clause c, il existe un lien entre le nœud de la clause et chaque littéral de la clause. Le lien est étiqueté pour distinguer les littéraux positifs et négatifs.

Dans [11], un seul GNN a été utilisé pour traiter l'ensemble du problème SAT, il fait la distinction entre les variables et les clauses en utilisant un encodage "one-hot" : $(0, 1)$ pour les nœuds des variables et $(1, 0)$ pour les nœuds des clauses. De la même manière, un littéral positif d'une clause reçoit une étiquette $(0, 1)$ et un littéral négatif $(1, 0)$. Cet encodage est utilisé pour alimenter un bloc indépendant afin de produire les embeddings initiaux qui sont liés au noyau itéré.

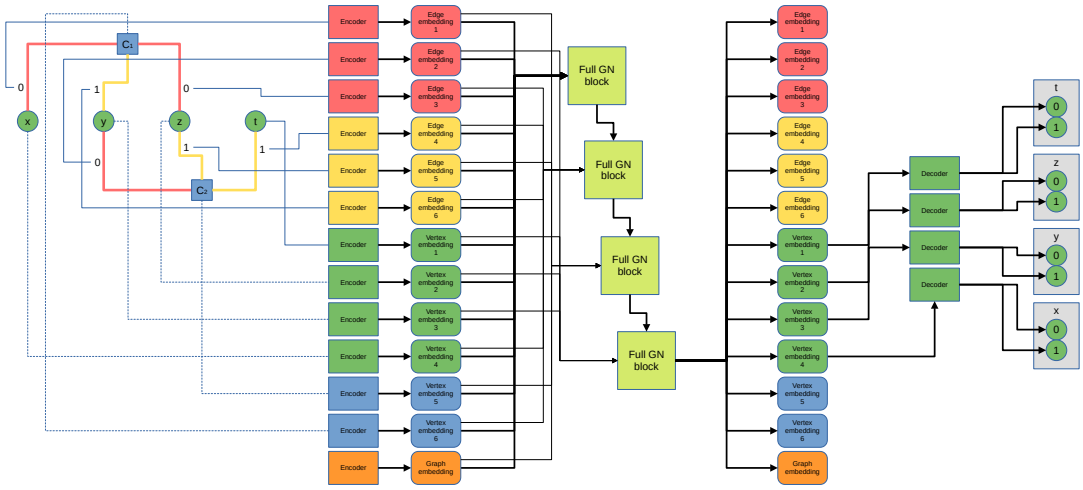


FIGURE 2 – Encodage du GNN sur un problème SAT simple

Comme le cadre de [11] se concentre uniquement sur la sélection des variables, le décodeur produit une Q-Value pour chaque polarité de chaque variable. Puisque la récompense est fixée à -0.1 pour une transition non-terminale et 0 pour une transition terminale, la maximisation de la Q-Value sélectionne le littéral ayant la plus grande probabilité de terminer la recherche.

Réseaux hétérogènes

Notre première amélioration est l'utilisation de GNNs hétérogènes [22] où différentes catégories de nœuds et d'arêtes ont des fonctions de mise à jour différentes. Nous définissons un ensemble différent de fonctions de mise à jour pour les nœuds des variables et pour les nœuds des contraintes. Cela nous permet non seulement de supprimer l'encodage one-hot des nœuds, mais aussi d'apprendre une fonction de mise à jour plus spécialisée pour chaque catégorie. Notre implémentation du GNN s'appuie sur l'extension TensorFlow `graph_nets` introduite dans [2]. À titre d'exemple, la figure 2 montre l'architecture pour deux clauses $x \vee \neg y \vee z$ et $y \vee \neg z \vee \neg t$. Pour chaque variable booléenne, on définit un encodeur (représenté en vert) qui produit un embedding initial. Il n'y a pas d'entrée pour cet encodeur puisque nous distinguons les nœuds des variables, des nœuds des contraintes en les utilisant dans des réseaux différents. Il s'agit donc simplement d'une couche linéaire qui produit un vecteur dans $\mathbb{R}^{64}$, suivie d'une couche de normalisation. Le même encodeur, mais chacun possédant son propre ensemble de paramètres, est défini pour les clauses (représentées en bleu), pour les littéraux positifs (représentés en rouge) et pour les littéraux négatifs (représentés en jaune), conduisant à des embeddings dans $\mathbb{R}^{64}$. L'embedding global du graphe est obtenu de la même manière, mais dans $\mathbb{R}^{32}$ (représentée en orange). Le bloc complet consiste en 4 itérations d'un perceptron multicouches récurrent avec une couche cachée de taille 1024 suivie d'une couche de normalisation (représenté en vert clair).

Entraînement antagoniste

Notre deuxième et principale amélioration consiste à définir deux GNNs pour les heuristiques de variable et de valeur au lieu d'un seul comme dans [11, 18]. Ces deux réseaux sont entraînés conjointement avec des récompenses antagonistes. Les couches du décodeur commencent, pour les deux réseaux, par un MLP entièrement connecté avec trois couches cachées de taille respective $256, 512$ et 256. Ensuite, une couche linéaire de taille 1 sort la Q-Value de chaque variable pour le réseau heuristique des variables, tandis que la couche linéaire de taille 2 sort la Q-Value de chaque polarité pour le réseau heuristique des valeurs.

La fonction de récompense associée à chacun de ces réseaux produit un comportement antagoniste entre les deux heuristiques. Le réseau heuristique variable est entraîné avec une récompense de type "premier échec" qui donne une valeur de 1 à tout état terminal, soit une solution ou un échec, et 0 à tout état intermédiaire. Il est intéressant de noter que, comme les succès sont beaucoup moins fréquents que les échecs, il arrive souvent que le réseau soit capable d'apprendre à trouver une solution sans en voir aucune ou presque. Il s'agit là simplement de la traduction du fait que l'on ne cherche pas à apprendre les solutions du problème mais bien un comportement permettant *en général* de guider la recherche. La récompense pour l'heuristique de valeur est le nombre de clauses supplémentaires qui sont satisfaites en prenant une action donnée. Les Q-Values associées à ces récompenses sont respectivement les valeurs attendues de la probabilité de clore un épisode et le nombre maximum de clauses satisfaites (MNSC). Nous appelons de façon globale la combinaison de ces deux heuristiques antagonistes AGNN (Antagonistic Graph Neural Network).

Le GNN est entraîné en collectant des échantillons (s, a, s') des triplets état-action-état décrivant l'effet d'une action dans un état donné. L'action est choisie selon la technique ϵ-

gloutonne [19] en utilisant l'état actuel de l'heuristique. La valeur de ϵ commence à 1 et est soumise à une décroissance exponentielle. Une fois qu'un problème SAT est choisi, il est exécuté jusqu'à ce qu'il échoue ou trouve une solution. Chaque échantillon est ajouté à une base d'expériences dans lequel les échantillons sont choisis aléatoirement pour l'apprentissage afin d'éviter les séquences fortement corrélées. La fonction objectif est donnée par l'apprentissage par différence temporelle. Nous conservons une version légèrement plus ancienne du réseau, appelée réseau cible, qui calcule une Q-Value Q_T. Nous exprimons alors la fonction objectif comme : *loss*$(Q(s,a), R + argmax\ Q_T(s',a'))$ où *loss* est la fonction objectif de Huber [10] appliquée à l'erreur quadratique moyenne.

$$H_\delta(y,\hat{y}) = \begin{cases} \frac{1}{2}(y-\hat{y})^2 & \text{if } |y-\hat{y}| \leq \delta \\ \delta(|y-\hat{y}| - \frac{1}{2}\delta) & \text{sinon} \end{cases}$$

Elle passe de quadratique à linéaire après un seuil δ afin de réduire l'influence des grands gradients au sein d'un lot. Nous utilisons également le clipping des gradients pour réduire l'influence des grands gradients entre les lots.

4 Résultats expérimentaux

Pour les expériences, nous avons utilisé le jeu de données SATLIB [9], qui contient diverses instances 3-SAT. En particulier, nous avons utilisé 10000 problèmes 3-SAT uniformes ayant 20 à 250 variables et 91 à 1065 clauses, et 40000 problèmes ayant 100 variables et 403 à 449 clauses avec des backbone contrôlées (littéraux qui sont vrais dans chaque solution réalisable) de taille 10 à 90. Les séries de backbone avec 70 et 90 variables sont respectivement appelées `b70` et `b90`. Comme le benchmark comporte 50000 problèmes, nous les avons répartis aléatoirement en 3 catégories : un ensemble d'entraînement (80%), qui est utilisé pendant la phase d'entraînement, un ensemble de validation (10%) qui est utilisé pour évaluer le modèle de l'agent actuel et un ensemble de test (10%) qui est inconnu pendant la phase d'entraînement et qui est utilisé pour rapporter ces résultats.

Notre système est expérimental et souffre de plusieurs sources d'inefficacité qui nous empêchent de fournir des informations intéressantes sur les temps de calcul. Premièrement, notre système utilise un solveur de contraintes "maison" en C++ qui n'est pas optimisé pour les problèmes SAT. Ensuite, chaque état est extrait du solveur et transféré via une interface Python vers le framework GNN GraphNet [2], puis de façon interne vers TensorFlow [1]. L'inférence est réalisée sur une carte graphique NVIDIA GTX 1080Ti et le résultat est renvoyé au solveur via cette même interface Python pour être utilisé dans une heuristique dédiée. En moyenne, une seule inférence prend $0,14$ seconde alors que MiniSat peut souvent résoudre le problème entier en $0,01$ seconde. Pour cette raison, nous nous concentrons uniquement sur l'évaluation de l'efficacité de l'apprentissage. Une amélioration ultérieure de l'architecture et de l'inférence n'étant bénéfique que si l'apprentissage s'avère utile à l'heuristique résultante.

Heuristique	Conflits b90	Conflits b70
DomDeg + MaxVal	906	715
DomDeg + MinVal	1963	1771
Minisat	263	229
MiniSat sans redémarrage	283	239
Single agent MNSC	188	159
AGNN Antagoniste	133	106
AGNN Antagoniste Small	**116**	**99**

TABLE 1 – Performances sur l'ensemble de test

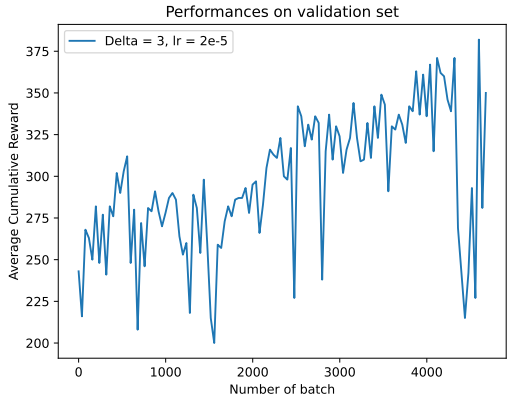
Efficacité de l'entraînement

Nous utilisons comme métrique le nombre d'échecs rencontrés avant la première solution. Notre objectif est de montrer que l'apprentissage à partir de problèmes offline est efficace et que notre architecture hétérogène antagoniste est supérieure aux autres techniques. Pour ce faire, nous comparons les heuristiques suivantes : `Dom/Deg + MaxVal`, `Dom/Deg + MinVal`, le solveur standard `MiniSat`, qui peut être considéré comme la base de référence, `MiniSat sans redémarrage`, pour faire des comparaisons sans la partie apprentissage de VSIDS [14], `Single agent MNSC`, où un seul agent est entraîné pour maximiser le nombre de clauses satisfaites (MNSC) et `AGNN Antagoniste`. Nous avons ajouté `AGNN Antagoniste Small` dans lequel l'heuristique GNN antagoniste (AGNN) est entraînée sur des problèmes SATLIB uniformes de petite taille, c'est-à-dire les 2100 problèmes ayant 20 à 75 variables.

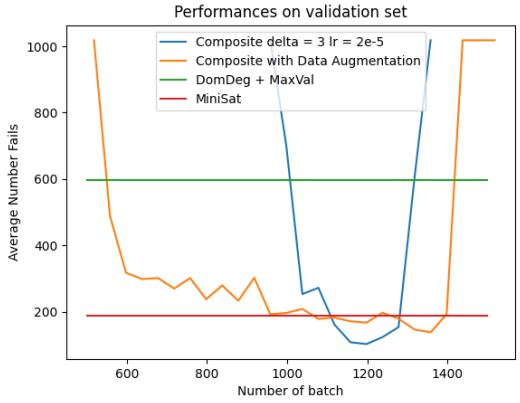
Les résultats sont présentés dans le Tableau 1. Nous observons que les deux agents antagonistes entraînés sur des objectifs opposés ont les meilleures performances, réduisant d'un facteur de respectivement $2,26$ et $2,31$ le nombre d'échecs nécessaires pour trouver la première solution sur `b90` et `b70` par rapport à MiniSat avec redémarrages. Comme les redémarrages sont une caractéristique orthogonale de la stratégie de recherche et ne sont pas utilisés par l'heuristique AGNN, nous indiquons également l'amélioration par rapport à MiniSat sans redémarrage, de respectivement $2,43$ et $2,31$. Nous observons que l'entraînement sur des petites instances augmente d'environ 10% la capacité de généralisation du GNN (avec un facteur de réduction passant de $1,97$ à $2,26$ sur `b90` et de $2,16$ à $2,31$ sur `b70`). Une explication possible est que plus de solutions sont rencontrées sur les petits problèmes que sur les grands au moment de l'apprentissage : le nombre de déroulements qui mènent à une solution sur les petits problèmes est de 19% alors qu'il n'est que de 0.19% pour les grands problèmes.

Apprentissage guidé par une satisfiabilité maximale

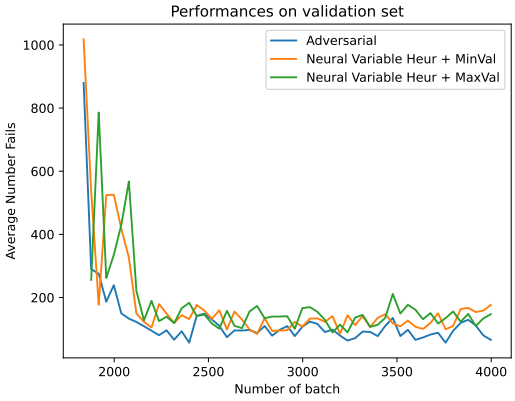
Nous évaluons l'impact de l'utilisation comme récompense du nombre de contraintes satisfaites, puisque nous supposons qu'une telle récompense devrait aider à rapprocher un agent d'une solution. Nous réalisons cette expérience en utilisant l'agent MNSC. La figure 3 montre, sur la même échelle, l'évolution conjointe du nombre d'échecs avant d'atteindre la première solution, et de la récompense cu-



(a) Agent unique (MNSC)



(b) Agent unique (échecs)



(c) Antagoniste (échecs)

FIGURE 3 – Influence de la fonction objectif

mulative pour l'agent (le nombre des contraintes satisfaites avant le premier retour en arrière). Comme on peut le voir sur la figure 3(a), le nombre de contraintes satisfaites augmente au fur et à mesure de l'apprentissage. Cependant, le nombre d'échecs avant la première solution, augmente brusquement après une diminution initiale (Figure 3(b)), ce qui montre que le processus d'apprentissage est instable. Nous pensons que cela est dû au fait qu'un agent entrainé avec cette récompense va guider la recherche vers un optimum local, où un grand nombre de contraintes sont satisfaites, mais qui n'est pas nécessairement proche d'une solution complète.

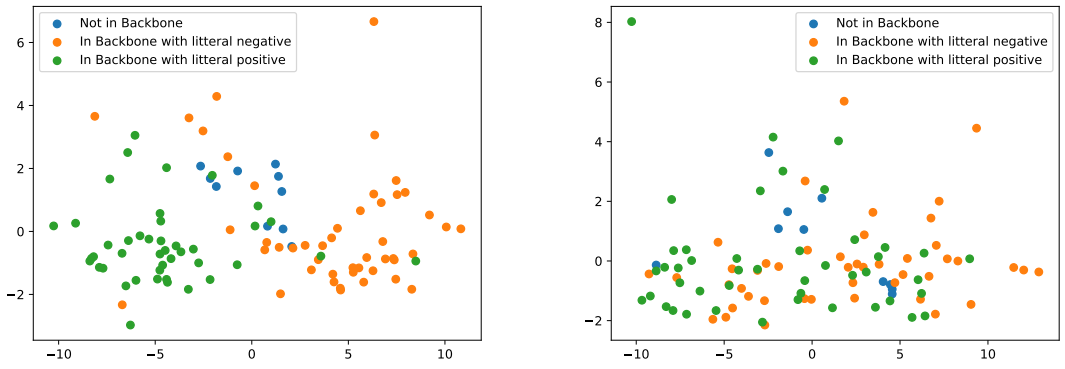
C'est probablement aussi pourquoi dans [11] le réseau est évalué après chaque épisode par rapport aux données de validation et le meilleur réseau est renvoyé à la fin. Nous avons également effectué une augmentation des données en considérant les valeurs opposées des littéraux dans un problème où tous les arcs sont inversés. La figure 3(b) montre que cela n'apporte aucune amélioration. En revanche, la même expérience avec l'approche antagoniste montre une bien meilleure stabilité. La figure 3(c) montre les performances de l'heuristique AGNN au fur et à mesure de l'apprentissage. En comparant l'entraînement antagoniste où la récompense pour le réseau des valeurs est le nombre des contraintes satisfaites (simplement appelé Adversarial dans la figure) avec les récompenses données par les heuristiques MinVal et MaxVal, nous observons que la performance est principalement obtenue par le réseau de l'heuristique des variables.

Structure du GNN et paramètres d'apprentissage

Nous avons comparé l'approche homogène et l'approche hétérogène pour différentes valeurs des hyperparamètres de l'apprentissage en Table 2 où δ est le paramère de la loss de Huber et lr la vitesse d'apprentissage. Nous pouvons observer que ces paramètres d'apprentissage ont une plus grande influence que le simple découpage des réseaux de neurones. Toutefois, l'approche hétérogène procure une légère amélioration dans tous les cas.

	Homogène			Heterogène		
δ	$+\infty$	3	10	$+\infty$	3	10
$lr = 2e^{-4}$	366	285	318	299	245	255
$lr = 2e^{-5}$	295	**114**	225	268	**103**	153

TABLE 2 – Performance sur l'ensemble de validation



(a) Problem 0 issu de b90

(b) Problem 2 issu de b90

FIGURE 4 – Visualisation du backbone par PCA sur les embeddings du GNN

L'importance des premières inférences
Puisque l'inférence du GNN est coûteuse, et que cela restera probablement le cas avec une meilleure implémentation, il est utile d'évaluer l'importance des premiers choix. En effet, c'est là qu'ils peuvent apporter beaucoup d'améliorations puisque tout apprentissage effectué par le solveur doit s'échauffer en commençant à résoudre le problème. La figure 5 illustre l'impact sur le nombre d'échecs lorsque les premières inférences sont faites par le GNN avant que le contrôle ne soit entièrement transféré à l'heuristique du solveur classique. On peut voir qu'il y a un impact majeur des 5-6 premières décisions. Ceci n'est pas surprenant car on s'attend à ce que le GNN puisse capturer une partie de la structure du problème à l'avance.

Apprentissage supervisé
Puisque les choix initiaux sont de la plus haute importance pour réduire le nombre de conflit avant la première solution, nous avons essayé d'apprendre de manière supervisée les Q-values afin de voir si le réseau réussit à les apprendre. Pour cela, nous avons considéré les problèmes avec backbone `b70` et `b90`. Considérant le nombre limité de variables qui sont à l'extérieur du backbone, nous pouvons directement leur donner un label vrai ou faux pour chacune de leur polarité sat ou unsat. Notons que dans ce cas il y a trois possibilités : soit la variable appartient au backbone et seule l'une de ses polarité est assignée à 1, l'autre à 0, soit elle est en dehors et on attend que le réseau trouve le label 1 pour les deux polarités. L'usage lors de la recherche est fait par une seule inférence du GNN qui prédit d'un seul coup l'ensemble des labels.

Nous avons évalué sur un ensemble de problèmes la pertinence de la prédiction des affectations pour l'appartenance au backbone. Nous avons obtenu un résultat de 73% dans le meilleur cas, alors qu'un classifieur de référence qui prédit la classe majoritaire de la polarité de la variable dans les clauses obtient un résultat de 60%. En effectuant une PCA sur la dernière couche du réseau juste avant la prédiction, on peut obtenir une projection des embeddings extraits par le réseau. La Figure 4 montre deux résultats représentatifs sur l'ensemble de test. Dans le premier cas, en Figure 4(a), la séparation de l'espace latent est bien visible, montrant que le modèle est parfaitement capable d'identifier l'emplacement du backbone et son affectation. Toutefois, il existe des cas où cela ne fonctionne pas, comme le cas présenté en Figure 4(b) où l'on constate une classification pour le moins confuse.

Influence du nombre d'itérations du coeur du GNN
La Figure 6 montre les résultats sur `b90` en fonction du nombre d'itérations du coeur du GNN. Il s'agit d'une expérience d'un intérêt pratique discutable puisque les temps d'inférences peuvent prendre plusieurs heures. Nous constatons toutefois un phénomène intéressant. D'une part le nombre de problèmes résolus sans backtrack augmente grandement, jusqu'à atteindre 20% (Figure 6(a)). A titre de comparaison, Minisat obtient un score de 0% et `AGNN Antagoniste Small` 0.8%. Mais la Figure 6(b) montre qu'en moyenne le nombre de conflits remonte quand on itère plus avec des résultats dégradés sur les autres problèmes.

5 Discussion et conclusion

Dans cet article, nous avons présenté une heuristique pour SAT appelée AGNN, basée sur les GNN, qui utilise deux agents antagonistes et des réseaux hétérogènes entraînés par l'apprentissage par renforcement. Nous avons observé une réduction de 50% du nombre de retours arrière sur les problèmes SATLIB par rapport à MiniSAT, même en autorisant les redémarrages. Ces résultats montrent que l'apprentissage est efficace et a le potentiel de guider la recherche. Cependant, notre implémentation ne réduit pas le temps de résolution effectif. Notre objectif initial est de montrer l'intérêt scientifique de l'apprentissage par renforcement profond pour guider la recherche. Il reste maintenant à s'intéresser à rendre l'approche efficace en pratique. En effet, les grands problèmes nécessitent de construire des grands réseaux de neurones et de nombreux embeddings qui doivent

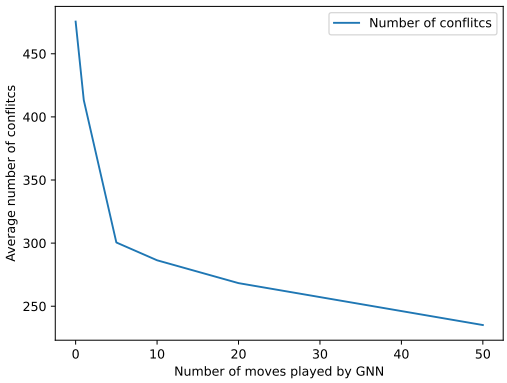
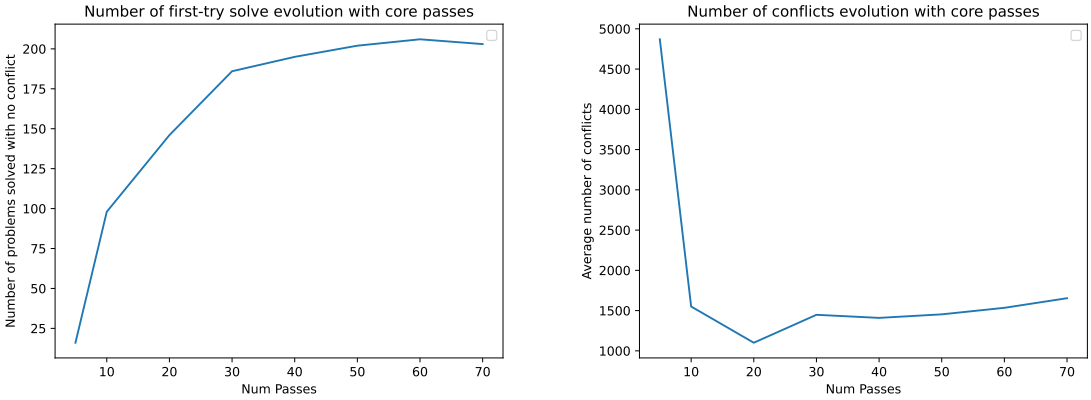


FIGURE 5 – Impact d'un nombre limité de coups joués par le GNN sur `b70`



(a) Problèmes résolus sans backtrack

(b) Nombre moyen de conflits

FIGURE 6 – Evolution des performances en fonction du nombre d'itérations du coeur

être recalculés à chaque fois qu'une décision est prise. Malgré l'utilisation d'un GPU pour les calculer, appeler l'heuristique AGNN à chaque fois qu'une décision est prise est plus coûteux que d'utiliser l'heuristique intégrée très rapide de MiniSAT. Nous avons vu, et ce résultat a également été observé dans [11] que l'utilisation de l'heuristique uniquement pour les premiers coups peut améliorer la vitesse de résolution et conserver le bénéfice de l'heuristique d'apprentissage. Cette direction mérite clairement une étude plus approfondie.

Une autre idée que nous avons exploré est d'effectuer une seule exécution du GNN et de prédire la polarité de toutes les variables afin de modifier l'initialisation de VSIDS. Cette approche est intéressante du point de vue de l'inférence mais semble difficile à fiabiliser en pratique. Nous avons choisi de ne pas utiliser de features créées à la main comme dans [18]. Des informations comme l'arité, la densité ou même les présolutions pourraient être très précieuses et devraient certainement être exploitées dans un système réel. Malheureusement, leurs utilisation ne permet pas d'isoler correctement l'impact de l'apprentissage sur les caractéristiques élaborées. Enfin, la manière dont l'embedding est calculée peut certainement faire l'objet d'une étude plus approfondie. En particulier, les contraintes n-aires définissent un hypergraphe, qui est imparfaitement représenté par un graphe avec des nœuds de contrainte. Le passage de SAT à CP nécessitera plus de travail dans cette direction, ainsi que dans la représentation des domaines.

Références

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow : Large-scale machine learning on hetero-

geneous systems, 2015. Software available from tensorflow.org.

[2] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinícius Flores Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Çaglar Gülçehre, H. Francis Song, Andrew J. Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey R. Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matthew Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *CoRR*, abs/1806.01261, 2018.

[3] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[4] Quentin Cappart, Thierry Moisan, Louis-Martin Rousseau, Isabeau Prémont-Schwarz, and André A. Ciré. Combining reinforcement learning and constraint programming for combinatorial optimization. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 3677–3687. AAAI Press, 2021.

[5] Félix Chalumeau, Ilan Coulon, Quentin Cappart, and Louis-Martin Rousseau. Seapearl : A constraint programming solver guided by reinforcement learning. In Peter J. Stuckey, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 18th International Conference, CPAIOR 2021, Vienna, Austria, July 5-8, 2021, Proceedings*, volume 12735 of *Lecture Notes in Computer Science*, pages 392–409. Springer, 2021.

[6] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

[7] Youssef Hamadi and Lakhdar Sais, editors. *Handbook of Parallel Constraint Reasoning*. Springer, 2018.

[8] John N. Hooker. Solving the incremental satisfiability problem. *J. Log. Program.*, 15(1&2) :177–186, 1993.

[9] Holger H. Hoos and Thomas Stützle. Satlib : An online resource for research on sat. pages 283–292. IOS Press, 2000.

[10] Peter J. Huber. Robust Estimation of a Location Parameter. *The Annals of Mathematical Statistics*, 35(1) :73 – 101, 1964.

[11] Vitaly Kurin, Saad Godil, Shimon Whiteson, and Bryan Catanzaro. Can Q-learning with graph networks learn a generalizable branching heuristic for a SAT solver ? In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33 : Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[12] Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. Exponential recency weighted average branching heuristic for SAT solvers. In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 3434–3440. AAAI Press, 2016.

[13] Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. Learning rate based branching heuristic for SAT solvers. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 123–140. Springer, 2016.

[14] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff : Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*, pages 530–535. ACM, 2001.

[15] Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006.

[16] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Trans. Neural Networks*, 20(1) :61–80, 2009.

[17] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. Learning a SAT solver from single-bit supervision. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[18] Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. Learning variable ordering heuristics for solving constraint satisfaction problems. *CoRR*, abs/1912.10762, 2019.

[19] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press, 2nd edition, 2018.

[20] Fei Wang and Tiark Rompf. From gameplay to symbolic reasoning : Learning SAT solver heuristics in the style of alpha(go) zero. *CoRR*, abs/1802.05340, 2018.

[21] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Satzilla : Portfolio-based algorithm selection for SAT. *J. Artif. Intell. Res.*, 32 :565–606, 2008.

[22] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V. Chawla. Heterogeneous graph neural network. In Ankur Teredesai, Vipin Kumar, Ying Li, Rómer Rosales, Evimaria Terzi, and George Karypis, editors, *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, pages 793–803. ACM, 2019.

Évaluation des forêts optimales d'arbres de décision

Gaël Aglin*, Siegfried Nijssen, Pierre Schaus

{prenom.nom}@uclouvain.be
ICTEAM, UCLouvain
Louvain-la-Neuve, Belgique

Résumé

Les forêts de décision sont des modèles d'apprentissage automatique très utilisés en pratique pour leur performance. Malheureusement, il est difficile de dire à quoi est dû leur performance parce que les algorithmes pour trouver des forêts de décision sont heuristiques. Ce papier résume notre papier "Assessing Optimal Forests of Decision Trees" accepté à ICTAI 2021 [3]. Dans ce papier, nous proposons une technique d'optimisation pour trouver des forêts de décision optimales. Cela permet de pouvoir facilement évaluer la pertinence des critères qui pourraient expliquer la performance des forêts de décision.

Mots-clés

Forêts optimales, boosting, génération de colonnes.

Abstract

Decision forests are machine learning models that are widely used in practice for their performance. Unfortunately, it is difficult to find what their performance is due to because the algorithms for finding decision forests are heuristic. This paper summarizes our paper "Assessing Optimal Forests of Decision Trees" accepted at ICTAI 2021 [3]. In this paper, we propose an optimization technique to find optimal decision forests. This allows to easily assess the relevance of criteria that could explain the performance of decision forests.

Keywords

Optimal forests, boosting, column generation.

1 Introduction

Les forêts de décision sont des modèles d'apprentissage très utilisés en pratique pour résoudre des problèmes difficiles. Elles sont beaucoup utilisées en raison de leur performance. Le *boosting* est l'une des techniques les plus connues pour construire des forêts de décision. Son principe est de construire successivement des arbres de décision, chacun ayant pour objectif de corriger les erreurs des précédents. Adaboost est l'un des algorithmes de *boosting* les plus populaires. Il combine plusieurs arbres de décision, chacun ayant un poids. Adaboost fonctionne bien en pratique mais il est difficile d'identifier à quoi ses performances sont dues. En effet, Adaboost est un algorithme heuristique. Contrairement à beaucoup d'autres algorithmes d'apprentissage automatique ou d'apprentissage profond dont le succès vient explicitement d'une fonction perte connue, Adaboost n'optimise pas un critère concret. Afin de mieux cerner le fonctionnemnet des forêts de décision, certains chercheurs ont proposé des modèles mathématiques qui reposent sur la définition d'un critère d'optimisation global et dont la résolution aboutirait à des forêts de décision optimales. Ces forêts optimales permettraient de mieux apprécier les différentes intuitions qui expliqueraient la performance des forêts de décision. Malheureusement, l'algorithme proposé pour résoudre les modèles n'aboutit pas à une solution optimale. Il n'est donc pas clair si les critères proposés peuvent vraiment justifier les performances des forêts de décision ou non. Dans cet article, nous nous appesantissons sur les modèles mathématiques LPBoost [4] et MDBoost [6]. Nous montrons qu'avec les algorithmes proposés récemment pour apprendre des arbres de décision optimaux, il est à présent possible de résoudre de façon optimale les modèles LPBoost et MDBoost. Finalement, nous évaluons avec exactitude la pertinence des critères proposés par ces modèles.

*Papier doctorant: Gaël Aglin est l'auteur principal

2 LPBoost et MDBoost

LPBoost et MDBoost sont des modèles mathématiques définissant un critère global à optimiser pour la construction de forêts de décision. L'idée est de trouver l'ensemble des arbres de décision et des poids dont la combinaison permettrait d'optimiser un critère spécifique, tout en étant performant. Afin de déterminer le critère à optimiser, ces deux modèles se basent sur un concept introduit par Schapire et al. [5]. Ce concept s'appelle la *marge*. La marge est un score donné par un ensemble d'arbres à la prédiction d'une instance d'entraînement. Le signe de la marge indique si la prédiction est correcte ou non. Si elle négative, la prédiction est incorrecte. Sinon, la prédiction est bonne. De même, la valeur absolue de la marge indique à quel point l'ensemble des arbres est unanime sur la prédiction, qu'elle soit correcte ou incorrecte. Schapire et al. ont montré que les algorithmes heuristiques de boosting maximisent la marge des instances au fur et à mesure qu'ils construisent des forêts. Cependant, ils n'ont pas donné une définition concrète d'un critère d'optimisation basé sur la marge.

Afin de trouver des forêts optimales performantes, LPBoost

identifie l'ensemble des arbres qui maximise la plus petite des marges des données d'entraînement. MDBoost, quant à lui, maximise la moyenne de toutes les marges tout en minimisant leur variance. Pour résoudre les modèles, un même algorithme de génération de colonnes a été utilisé. Le primal permet d'identifier les poids des arbres obtenus pendant que le dual permet d'identifier à chaque itération, les poids à affecter aux instances afin d'améliorer leur marge. Concrètement, les poids sont utilisés à chaque itération pour apprendre un nouvel arbre qui améliore les marges.

3 Forêts optimales

Lors de la résolution des modèles LPBoost et MDBoost, les chercheurs ont utilisé un algorithme heuristique afin de trouver chaque nouvel arbre à ajouter à l'ensemble, sur base des poids fournis par le dual. De ce fait, les arbres obtenus n'ont pas la garantie d'optimalité. Par conséquent, l'algorithme de génération de colonnes n'a aucune assurance de l'optimalité du résultat final puisqu'il n'est pas possible de prouver qu'il n'existe plus aucun arbre à rajouter pour améliorer le critère d'optimisation.

Afin de résoudre efficacement les différents modèles et obtenir des forêts réellement optimales, nous proposons pour la première fois, dans ce papier, un algorithme pour trouver des arbres optimaux supportant des coûts variables pour les instances. Afin de pouvoir obtenir ces arbres, nous nous sommes appuyés sur l'algorithme DL8.5 [1] rendu disponible par la librairie PyDL8.5 [2]. Le changement majeur effectué dans cet algorithme pour supporter les coûts des instances est le changement de la fonction objectif minimisée dans chaque feuille de l'algorithme. Pour un jeu de données avec un ensemble de classes $\mathcal{C}$, l'erreur minimisée dans une feuille ℓ contenant des instances, chacune associée à un coût w_i et une classe y_i est $erreur(\ell) = \sum_{i \in \ell} w_i - \max_{c \in \mathcal{C}} \sum_{i \in \ell \wedge y_i = c} w_i$. La nouvelle fonction que nous avons introduite n'est pas adaptée à l'implémentation de DL8.5 proposée par PyDL8.5 parce qu'il utilise une optimisation algorithmique basée sur des vecteurs binaires et qui considère les instances par bloc de 64. Cette optimisation, permet d'accélérer le calcul des erreurs dans les feuilles. Cependant, elle ne convient pas à des opérations sur des coûts, qui ne peuvent pas être binarisées. Au contraire, l'utilisation des vecteurs binaires contraint le calcul de l'erreur à parcourir toutes les instances du jeu de données. Nous proposons donc une adaptation de cette optimisation qui permet de ne parcourir dans une feuille, que les instances concernées par la feuille.

4 Résultats

Nous avons réalisé différentes expériences. Dans un premier temps, nous avons exécuté notre algorithme et l'avons comparé à l'algorithme qui utilise des arbres heuristiques. Les valeurs objectives de nos forêts sont toujours meilleures que celles de l'algorithme existant. Cela prouve que les forêts trouvées avec l'algorithme existant n'étaient pas optimales. Pour évaluer la pertinence de la forêt optimale, nous avons effectué une validation croisée et comparé la précision de nos forêts à celle des forêts trouvées avec l'algorithme existant. Il apparaît que les forêts optimales sont plus performantes que les autres sur la plupart des données. Cependant, il y a quelques jeux de données sur lesquels les forêts de l'algorithme existant sont plus performantes que les nôtres. On peut en conclure que les forêts optimales peuvent augmenter les performances, mais pas systématiquement sur tous les jeux de données. De plus, les résultats des forêts optimales avec le modèle MDBoost sont meilleurs que ceux des forêts qui utilisent le modèle LPBoost. Cela implique que la qualité des forêts optimales dépend aussi de la fonction objectif optimisée. Il faut aussi noter que, la forêt optimale avec MDBoost surpasse Adaboost sur de nombreux jeux de données.

5 Conclusion

Ce travail a intérêt à évaluer les forêts de décision optimales afin d'évaluer l'intuition du succès des forêts de décision. Dans un premier temps, nous avons montré que l'algorithme existant ne permet pas de trouver des forêts optimales. Ensuite, nous avons proposé un nouvel algorithme pour trouver des arbres de décision pondérés optimaux capables d'être utilisés pour construire des forêts optimales. Nous avons montré que les forêts optimales obtiennent toujours une meilleure valeur d'optimisation que les forêts de l'algorithme existant. En termes de généralisation, il apparait que les forêts optimales ont une bonne capacité de généralisation mais cela dépend fondamentalement du critère optimisé. Notre travail ouvre la porte à l'exploration de nouvelles fonctions pouvant permettre d'améliorer la généralisation des forêts de décision.

Références

[1] Gaël Aglin, Siegfried Nijssen, and Pierre Schaus. Learning optimal decision trees using caching branch-and-bound search. In *Proceedings of the AAAI Conference*, volume 34, pages 3146–3153, 2020.

[2] Gaël Aglin, Siegfried Nijssen, and Pierre Schaus. PyDL8.5 : a library for learning optimal decision trees. In *Proceedings of 29th IJCAI*, pages 5222–5224, 2021.

[3] Gaël Aglin, Siegfried Nijssen, and Pierre Schaus. Assessing optimal forests of decision trees. In *2021 IEEE 33rd ICTAI*. IEEE, 2021.

[4] Ayhan Demiriz, Kristin P Bennett, and John Shawe-Taylor. Linear programming boosting via column generation. *Machine Learning*, 46(1-3) :225–254, 2002.

[5] Robert E Schapire, Yoav Freund, Peter Bartlett, Wee Sun Lee, et al. Boosting the margin : A new explanation for the effectiveness of voting methods. *The annals of statistics*, 26(5) :1651–1686, 1998.

[6] Chunhua Shen and Hanxi Li. Boosting through optimization of margin distributions. *IEEE Transactions on Neural Networks*, 21(4) :659–666, 2010.

Fouille des motifs High Utility via SAT

Amel Hidouri[1,2], Said Jabbour[1], Badran Raddaoui[3], Boutheina Ben Yaghlane[2]

[1] Université d'Artois, CRIL
[2] Université de Tunis, LARODEC
[3] Télécom SudParis, Institut Polytechnique de Paris, SAMOVAR

Résumé

L'extraction des motifs dits high utility est un problème clé de la fouille de données visant à découvrir les motifs qui génèrent une utilité élevée. La fonction d'utilité peut être quantifiée en termes de profit, de coût, ou toute autre mesure de préférence définie par l'utilisateur. D'autres part, la fouille de données à base d'Intelligence Artificielle (AI) symbolique est un cadre prometteur permettant de spécifier les propriétés des motifs à découvrir en terme de contraintes. Deux modèles de représentation et de résolution en IA ont été utilisés pour représenterr et résoudre différent problèmes en fouille de données, à savoir la programmation par contrainte (CP) et la satisfiabilité propositionnelle (SAT). Dans ce travail [1], nous proposons une approche basée sur la logique propositionnelle pour résoudre le problème de la fouille des itemsets high utility.

Mots-clés

Fouille de données, High Utility, Intelligence Artificielle Symbolique, Satisfiabilté Propositionelle

1 Introduction

La fouille de données (data mining, en anglais) est un domaine de recherche multi-disciplinaire qui exploite l'apprentissage automatique, les statistiques ainsi que l'Intelligence Artificielle symbolique afin de d'inférer des relations nouvelles et valides entre les données. La fouille des itemsets "profitables" ou high utility (High Utility Itemsets Mining, en anglais ou HUIM) est un problème clé dans l'extraction des motifs. Ce problème a émergé pour pallier les limites du problème classique de l'extraction des motifs fréquents (Frequent Itemset Mining en anglais ou FIM). HUIM consiste à découvrir l'ensemble des articles/items apparaissant ensemble et générant un profit/gain supérieur à un seuil d'utilité minimum. Chaque item est caractérisé par une quantité dans les transactions ainsi qu'un profit unitaire. En fait, l'utilité peut être quantifiée en fonction de profit, du coût, ou toute autre mesure de préférence définie par l'utilisateur.

La fouille de données à base d'Intelligence Artificielle (AI) symbolique est un cadre prometteur permettant de spécifier les propriétés des motifs à découvrir en terme de contraintes. Ainsi, les contraintes font souvent partie de la spécification de plusieurs problèmes en fouille de données. Ce constat a permis la création d'un nouvel axe de recherche multidisciplinaire combinant les techniques de l'Intelligence artificielle symbolique et le domaine de la fouille de données. Encouragé par les travaux récents sur la reformulation de problèmes d'énumération d'itemsets classiques en programmation par contraintes, l'objectif de cet article est de représenter et de résoudre en SAT le problème de la fouille des motifs high utility dans les bases de données transactionnelles.

1. La version anglaise de cet article publiés dans la revue (Data Knowl. Eng. 136 : 101927 (2021)), le premier auteur est un étudiant.

2 Travaux antérieurs

Les approches de la fouille des motifs high utility peuvent être scindés en deux grandes classes : celles qui opèrent en deux phases et ceux qui permettent d'extraire en une seule phase des motifs high utility. Les algorithmes de la première catégorie utilisent généralement une sur-estimisation de la mesure d'utilité afin d'élaguer l'espace de recherche. Ils génèrent d'abord un ensemble de motifs candidats en surestimant leur utilité dans la phase 1. Ensuite, dans la phase 2, ils effectuent une analyse supplémentaire de la base de données pour calculer l'utilité exacte des candidats et filtrer les ensembles de motifs qui ne peuvent pas être des HUIs. Par exemple, les algorithmes Two-Phase, IHUP et Up-Growth font partie de cette première classe d'approches.

Les approches de la seconde famille appliquent des stratégies d'élagage pour réduire l'espace de recherche en fonction des bornes supérieures de la mesure d'utilité. Parmi ces algorithmes, nous pouvons citer EFIM, D2HUP et FHM, etc.

3 Préliminaires

3.1 Motifs profitables

Étant donné l'ensemble $\Omega = \{a_1, a_2, \ldots, a_n\}$ de n items distincts. Une base données de transactions $D = \{(i_1, T_1), (i_2, T_2), \ldots, (i_m, T_m)\}$ est un ensemble de transactions où chaque transaction T est définie par $(i, T_i) \in D$ et est composée d'un identifiant i et un ensemble d'items T_i.

Un motif X est un sous-ensemble non vide de Ω, i.e., $X \subseteq \Omega$. Chaque item dans une transaction est associé à une utilité *interne* $w_{int}(a, T_i)$ et une utilité *externe* $w_{ext}(a)$. Ainsi, l'utilité d'un item a dans une transaction $(i, T_i) \in D$ est $u(a, T_i) = w_{int}(a, T_i) \times$

$w_{ext}(a)$. L'utilité d'un itemset X dans une transaction est $u(X, T_i) = \sum_{a \in X} u(a, T_i)$, l'utilité d'un itemset X dans D est $u(X) = \sum_{(i,T_i) \in D \mid X \subseteq T_i} u(X, T_i)$. Un itemset X est dit high utility (HUI) dans une base de données D s'il a une valeur d'utilité supérieure à un seuil d'utilité minimum donné θ.

3.2 Logique propositionnelle et SAT

Soit $\mathcal{L}$ un langage propositionnel construit inductivement à partir d'un ensemble fini PS de variables propositionnelles, les constantes booléennes $\top$ (*true* ou 1) et $\bot$ (*false* ou 0) ainsi que les connecteurs logiques classiques $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$. Nous utilisons les lettres x, y, z, etc. pour désigner les éléments de PS. Les formules propositionnelles de $\mathcal{L}$ sont notées Φ, Ψ, etc. Un littéral est une variable propositionnelle (x) de PS ou sa négation ($\neg x$). Une clause est une disjonction (finie) de littéraux. Pour toute formule propositionnelle Φ du langage $\mathcal{L}$, $\mathcal{P}(\Phi)$ dénote les symboles de PS apparaissant dans Φ.

4 Modélisation de HUIM en SAT

Dans ce travail, un encodage vers la logique propositionnelle du problème de la fouille des motifs high utility est introduit. Notre formulation est constituée d'un ensemble de contraintes comme résumé dans la Figure 1. Plus précisément, étant donnée une table de transactions, l'encodage proposé associe à chaque item a (resp. identifiant de transaction i), une variable propositionnelle dénommée p_a (resp. q_i). La première formule propositionnelle (1) représente la couverture de l'itemset candidat. Cette contrainte (1), appelée contrainte de couverture, permet d'exprimer qu'une transaction T_i ne contient pas l'itemset X, si un des items de X n'est pas dans T_i. La contrainte (2) permet de forcer les itemsets recherchés à être fermés. Cette contrainte est la même que l'encodage classique des itemets fréquents. Elle stipule que pour un item a, si toutes les transactions qui ne contienent pas a n'appartiennent pas à la couverture de l'itemset recherché X, alors a doit être dans X. Finalement, la contrainte sur l'utilité d'un itemset X dans D est exprimée en utilisant l'inéquation linéaire (3). Cette contrainte exige que l'utilité de l'itemset recherché doit être supérieure ou égale à un seuil fixé θ. En utilisant une variable supplémentaire (r_{ai}), la contrainte (3) peut être réecrite comme la conjonction des contraintes (4) et (5).

Par conséquent, le problème HUIM peut être modélisé en logique avec la formule propositionnelle $\Phi_{huim} = (1) \wedge (3)$. Pour encoder le probleme de la fouille des HUIs fermés, la formule résultante de l'encodage est $\Phi_{chuim} = (1) \wedge (2) \wedge (3)$

Afin d'énumérer tous les modèles de la formule Φ_{huim} (resp.Φ_{chuim}), nous proposons une nouvelle approche fondée sur DPLL légèrement modifié pour l'énumération de tous les modèles.

Dans cet article, nous définissons une stratégie de décomposition pour améliorer l'efficacité de notre algorithme. L'idée principale est d'éviter d'encoder toute la base de données en faveur de la résolution de nombreux sous-problèmes indépendants de petite taille plutôt qu'un seul gros problème.

$$\bigwedge_{i=1}^{m} (\neg q_i \leftrightarrow \bigvee_{a \in \Omega \setminus T_i} p_a) \tag{1}$$

$$\bigwedge_{a \in \Omega} (p_a \vee \bigvee_{T_i \in D \mid a \notin T_i} q_i) \tag{2}$$

$$\sum_{i=1}^{m} \sum_{a \in T_i} u(a, T_i) \times (p_a \wedge q_i) \geq \theta \tag{3}$$

$$\sum_{i=1}^{m} \sum_{a \in T_i} u(a, T_i) \times r_{ai} \geq \theta \tag{4}$$

$$\bigwedge_{i=1}^{m} \bigwedge_{a \in T_i} (r_{ai} \leftrightarrow p_a \wedge q_i) \tag{5}$$

FIGURE 1 – Encodage de HUIM en SAT

Formellement, étant donné l'ensemble des items $\Omega = \{a_1, \ldots, a_n\}$, l'ensemble de modèles de la formule Φ_{huim} peut être divisé en $\{E_1, \ldots, E_n\}$ avec E_1 est un sous-ensemble contenant l'item a_1, E_2 est un sous-ensemble contenant l'item a_2 mais pas l'item a_1, E_n contient a_n mais pas les items $a_1, \ldots, a_{n-1}$. Plus généralement, E_i est obtenu en enumérant les modèles de la sous-formule Φ_i en propageant $p_{ai} = \top$ et $p_{ak} = \bot \; \forall \; 1 \leq k < i$. En d'autres termes, $\Phi_i = \Phi \wedge p_{a_i} \wedge \bigwedge_{1 \leq k < i} \neg p_{a_k}$.

5 Résultats expérimentaux

Nous comparons la performance de notre algorithme SATHUIM avec les algorithmes spécialisés EFIM [1] et D2HUP [3]. Son extension appelée SATCHUIM est comparée avec EFIM-Closed [4], CHUD [2] et CHUI-Miner. Les tests sont éffectués sur des instances réelles prises de SPMF. Les expérimentations montrent que nos méthodes sont très compétitives avec l'état de l'art et passent à l'échelle sur des grandes instances. Notons que les performances de tous les algorithmes dépendent des caractéristiques de l'instance ainsi que les valeurs du seuil d'utilité.

Références

[1] Zida, S., Fournier-Viger, P., Lin, J. C. W., Wu, C. W., & Tseng, V. S. (2017). EFIM : a fast and memory efficient algorithm for high-utility itemset mining. Knowledge and Information Systems, 595-625.

[2] Tseng, V. S., Wu, C. W., Fournier-Viger, P., & Philip, S. Y. (2014). Efficient algorithms for mining the concise and lossless representation of high utility itemsets. IEEE TKDE, 27(3), 726-739.

[3] Liu, J., Wang, K., & Fung, B. C. (2012, December). Direct discovery of high utility itemsets without candidate generation. In 2012 ICDM (pp. 984-989).

[4] Fournier-Viger, P., Zida, S., Lin, J. C. W., Wu, C. W., & Tseng, V. S. EFIM-closed : fast and memory efficient discovery of closed high-utility itemsets. ICML 2016 (pp. 199-213).

Intégration incrémentale de contraintes pour le clustering avec la programmation par contraintes

Aymeric Beauchamp[1], Thi-Bich-Hanh Dao[1], Samir Loudni[2], Christel Vrain[1]

[1]Université d'Orléans, LIFO
[2]IMT Atlantique Bretagne-Pays de la Loire

[1]{aymeric.beauchamp, thi-bich-hanh.dao, christel.vrain}@univ-orleans.fr
[2] samir.loudni@imt-atlantique.fr

Résumé

L'implication active d'un expert dans un processus de clustering sous contraintes se traduit par une démarche incrémentale où les contraintes expertes sont ajoutées à la volée. Toutefois, les clusterings intermédiaires produits devront être relativement similaires pour ne pas dérouter l'expert. Nous proposons un modèle en PPC permettant d'obtenir une partition similaire à la partition existante tout en tenant compte des contraintes. Notre modèle peut traiter plusieurs types de contraintes, relâcher les contraintes et créer un nouveau cluster. Des expériences menées sur des jeux de données de référence ainsi que dans un cas d'usage réel en télémétrie montrent l'intérêt de notre modèle.

Mots-clés

Clustering sous contraintes, programmation par contraintes, optimisation combinatoire

Abstract

Actively involving an expert in a constrained clustering loop translates into an incremental process where expert constraints are added on the fly. However, intermediate clusterings must be somewhat similar to one another in order not to confuse the expert. We propose a CP model to build a partition that is similar to an existing partition while taking constraints into account. Our model can support several types of constraints and features constraint relaxation and cluster creation. Experiments performed on popular datasets as well as a use case in remote sensing denote the relevance of our model.

Keywords

Constrained clustering, constraint programming, combinatorial optimization

1 Introduction

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 même cluster soient similaires mais différents de ceux appartenant aux autres clusters. Le clustering sous contraintes [6] 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 imposant que des objets appartiennent (ou non) à un même cluster [14], ou que les clusters aient une taille minimale et/ou maximale [1]. 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 incorporer dès le début du processus toutes les connaissances nécessaires à l'élaboration d'une partition intéressante. L'idée est donc de lui proposer une partition qui lui permettra de suggérer des connaissances intéressantes à intégrer, conduisant ainsi à une nouvelle partition qui pourra être encore itérativement améliorée. Ceci constitue un processus interactif qui est bien connu en fouille de données.

Comme l'illustre la figure 1, l'implication active de l'expert dans le processus de clustering se traduit par l'intégration incrémentale de contraintes représentant des connaissances de l'expert à une partition existante. Ces contraintes peuvent prendre des formes diverses et doivent être satisfaites dans la nouvelle partition modifiée. Par exemple, l'expert peut ainsi indiquer que deux objets appartenant à un même cluster devraient être séparés car il les juge dissimilaires. Les nouvelles contraintes données par l'expert doivent être prises en compte pour construire une nouvelle partition. Toutefois, afin de ne pas trop dérouter l'expert, la partition résultante doit être relativement similaire à la partition initiale.

Une première approche naïve pour tenir compte des contraintes données par l'expert consiste à relancer un algorithme de clustering sous contraintes sur les données avec les nouvelles contraintes. Cependant, cette approche présente au moins deux inconvénients majeurs : (1) elle relance le clustering sur les données initiales à partir de zéro sans tenir compte des résultats intermédiaires ; (2) elle peut produire des résultats très différents de la partition qui a été présentée à l'expert. Ainsi, l'enjeu dans une approche incrémentale est de réduire le fossé entre les résultats produits par l'algorithme du clustering et les intuitions de l'expert durant la boucle d'interaction.

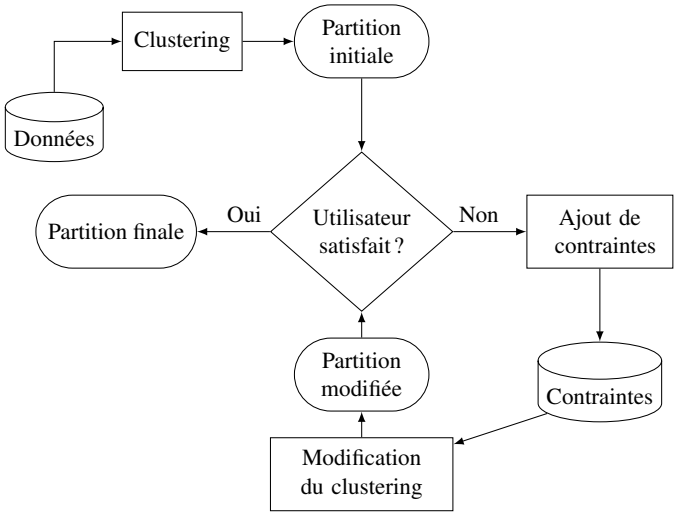


FIGURE 1 – Représentation du processus de clustering par contraintes incrémentales.

Dans cet article, nous proposons une nouvelle approche incrémentale basée sur la programmation par contraintes (PPC) permettant d'intégrer des contraintes utilisateur à une partition existante, et de produire une nouvelle partition qui soit proche de la partition existante. L'originalité de notre méthode est d'intégrer dans le critère d'optimisation, pour les points changeant de cluster, la distance au centre de leur nouveau cluster, pour limiter les modifications par rapport à une partition existante. À la différence des méthodes existantes à base de la programmation linéaire en nombres entiers (PLNE) [11], notre méthode permet de trouver une solution satisfaisant le plus grand nombre de contraintes dans le cas de conflits entre les contraintes, alors que les autres approches échouent. Des expériences menées sur des jeux de données de référence ainsi que sur un cas d'usage réel en télémétrie montrent l'intérêt de notre modèle. Nous discutons également des inconvénients d'une modification minimale et présentons une solution de compromis entre deux critères : l'amélioration du clustering par les contraintes et la nécessité de ne pas produire de résultats perturbants pour l'expert.

Dans la section 2, nous discutons des travaux antérieurs liés au clustering sous contraintes, ou à la modification minimale de clustering. Dans la section 3, nous décrivons notre modèle en programmation par contraintes pour la modification de clustering et détaillons des méthodes pour augmenter l'efficacité des contraintes. Dans la section 4, nous présentons les résultats du modèle sur des jeux de données de référence ainsi que sur des séries temporelles en télémétrie.

2 Travaux antérieurs

Les premiers travaux développés sur l'intégration de contraintes dans des tâches de clustering sont des extensions d'algorithmes de clustering classiques pour intégrer des contraintes *must-link/cannot-link* [4, 14, 16]. Ces algorithmes peuvent chercher à satisfaire toutes les contraintes comme COP-KMEANS [14] ou cherchent une solution qui est un compromis entre la satisfaction de contraintes et la qualité du clustering [2]. Par la suite, des approches fondées sur des formalismes déclaratifs ont été proposées, permettant d'intégrer des contraintes plus générales que des contraintes *must-link/cannot-link*, avec un solveur SAT [5], avec la PLNE [12] ou encore avec la PPC [3]. Toutes ces méthodes cependant cherchent à construire un clustering à partir des données en présence des contraintes. Elles ne sont par conséquent pas appropriées dans une démarche incrémentale, car l'apparition de nouvelles contraintes nécessite de relancer l'algorithme de nouveau à partir des données, sans garantie que la nouvelle partition soit similaire à l'ancienne.

Le problème de modification minimale d'une partition a été posé dans [9]. Le modèle proposé dans ce travail est basé sur la PLNE et intègre des contraintes sur les diamètres des clusters relativement à un descripteur. L'objectif recherché est d'enlever des propriétés indésirables d'une partition. Cependant dans ce travail, la différence avec la partition initiale est mesurée en nombre de points qui changent de cluster, sans tenir compte de l'homogénéité des clusters. Une autre approche utilisant la PLNE [11] exploite la partition initiale en mesurant le score d'appartenance d'un point à chaque cluster. Ce travail cherche ensuite à construire un clustering optimisant un critère fondé sur ce score ; toutes les données sont prises en compte, même celles qui ne sont pas liées aux contraintes. Notons enfin que ces approches ne permettent pas de gérer le cas de conflits entre contraintes.

Notre approche est basée sur la PPC, et vise à conserver l'homogénéité des clusters d'une partition existante tout en tenant compte de nouvelles contraintes. Elle permet d'intégrer différents types de contraintes et de relâcher des contraintes en cas de conflit.

3 Modèle de clustering par contraintes incrémentales

Nous présentons dans cette section un modèle en programmation par contraintes pour le problème de modification minimale de partition : nous utilisons un critère d'optimisation basé sur la distance qui tend à limiter la portée des changements apportés à une partition existante – produite par un algorithme de clustering ou issue d'une précédente itération du modèle – pour rester relativement similaire à cette dernière tout en améliorant sa qualité par l'intégration du retour expert sous forme de contraintes.

3.1 Problème d'optimisation sous contraintes pour la modification de clustering

Nous considérons un ensemble de N instances $\mathcal{X}$ et une partition existante $\mathcal{P}$ de $\mathcal{X}$ en K clusters, où $\mathcal{P}[i]$ indique le numéro du cluster contenant l'instance i. Nous proposons de représenter la structure de la partition $\mathcal{P}$ par une matrice de distances $\mathcal{D}$ de taille $N \times K$, où $\mathcal{D}[i, c]$ représente la distance d'une instance i au centre du cluster c.

Variables et fonction objectif. Pour représenter la partition modifiée, nous utilisons N variables $X_1, \ldots, X_N$ ayant pour domaines $\{1, \ldots, K\}$, où $X_i = c$ indique que l'instance i est affectée au cluster c. Comme la matrice $\mathcal{D}$ représente la structure de $\mathcal{P}$, afin d'obtenir une partition proche à $\mathcal{P}$, nous proposons une fonction objectif qui minimise, pour les instances réaffectées dans la partition modifiée, la distance au centroïde de leur nouveau cluster :

$$\arg\min \sum_{i=0}^{N} \mathbb{I}(\mathcal{P}[i] \neq X_i)\ \mathcal{D}[i, X_i]$$

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.

Contraintes expert. Différents types de contraintes d'expert peuvent être intégrés dans le modèle. Une contrainte *must-link* (resp. *cannot-link*) sur deux instances i, j indique que ces instances doivent être (resp. ne doivent pas être) dans le même cluster. Ces contraintes sont représentées par $X_i = X_j$ pour la contrainte *must-link* et $X_i \neq X_j$ pour la contrainte *cannot-link*.
Une contrainte triplet (a, p, n) signifie que l'instance a *est plus proche de* p *que de* n. Elle est traduite par :

$$X_a = X_n \implies X_a = X_p$$
$$X_a \neq X_p \implies X_a \neq X_n$$

Pour inclure des retours expert plus thématiques, des contraintes logiques peuvent être intégrées. Elles peuvent être de la forme $P \implies Q$, où P et Q sont des conjonctions de contraintes élémentaires de type $X_i = X_j$ ou $X_i \neq X_j$.

Création de nouveaux clusters. Notre modèle permet de spécifier un nombre de clusters plus grand que K, en définissant le domaine des X_i à $\{1, \ldots, K'\}$, avec $K' > K$. Il permet ainsi d'affecter une instance à un nouveau cluster en cas de non satisfaction de toutes les contraintes. Par exemple si $K = 2$ et que l'expert impose les trois contraintes suivantes : $X_i \neq X_j, X_j \neq X_l$ et $X_l \neq X_i$, alors il sera nécessaire de former un nouveau cluster pour trouver une solution. Par rapport à la partition existante $\mathcal{P}$, nous fixons la distance $\mathcal{D}[i, k']$ à une valeur plus grande que la distance au centre le plus éloigné de i pour $K < k' \leq K'$, et ce afin d'éviter que le modèle ne crée des clusters dont le seul intérêt est d'optimiser la fonction objectif.

Relâchement de contraintes expert. L'expert pourrait faire des erreurs, ce qui se traduirait par le fait que les contraintes ajoutées sont contradictoires. Il n'existe pas dans ce cas de solution satisfaisant toutes les contraintes. Nous autorisons alors à satisfaire un certain nombre de contraintes. Chaque contrainte expert c est associée à une variable booléenne S_c de sorte que $S_c = 1$ si et seulement si c est satisfaite. La somme des variables S_c donne le nombre de contraintes satisfaites par une solution donnée. On peut ainsi imposer une contrainte établissant un seuil de satisfaction des contraintes expert :

$$\sum_{c} \mathcal{S}_c \gtreqless \delta$$

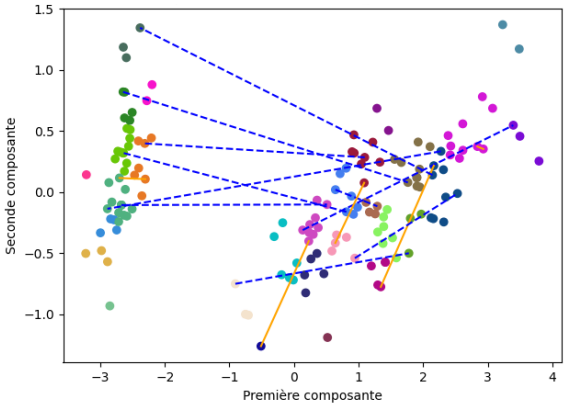
L'introduction de la relaxation de contraintes permet à notre modèle de résoudre des problèmes contenant des contraintes contradictoires et d'ignorer des contraintes nécessitant la modification d'instances éloignées de leur nouveau cluster. Dans un contexte incrémental, on peut envisager que l'expert fasse varier le taux de satisfaction des contraintes. Cette fonctionnalité engendre toutefois une combinatoire plus importante qui se répercute sur le temps de calcul (cf. section 4.1.2 pour plus de détails).

Restriction du modèle. Étant données de nouvelles contraintes, l'objectif est de modifier la partition existante en préservant le plus possible la structure de $\mathcal{P}$ tout en satisfaisant ces nouvelles contraintes. Ainsi, si une instance n'apparaît dans aucune contrainte, son affectation ne sera pas modifiée afin de conserver la structure de $\mathcal{P}$. Par conséquent il suffit de n'inclure dans le modèle de PPC que les instances concernées par au moins une contrainte.

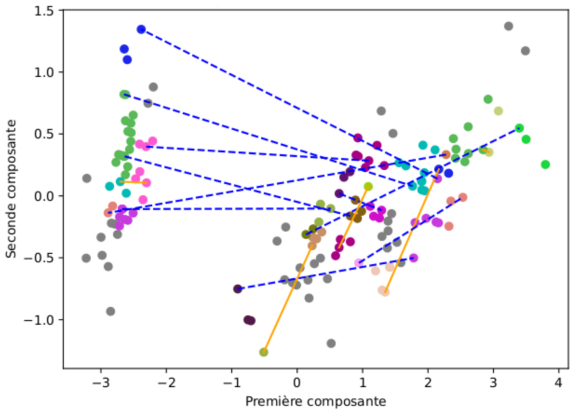
3.2 Propagation des modifications

Dans une application réelle, nous supposons que l'expert ne va réagir que sur un petit nombre d'instances par itération. Il en découle que la modification du clustering est tellement minimale qu'elle en devient imperceptible. Il est donc important d'exploiter les retours pour propager les informations/modifications et ainsi éviter de poser à l'expert un nombre prohibitif de requêtes. Il faut dès lors trouver un moyen de permettre la propagation des modifications à des instances non contraintes, tout en la contrôlant pour ne pas dérouter l'expert. Notre intuition est que les contraintes sur des instances ne concernent pas uniquement les instances elles-mêmes, mais aussi leur voisinage. Dans cette optique, nous présentons deux façons de transmettre localement les modifications réalisées par notre modèle.

Propagation par super-instances. Notre première approche consiste à construire des super-instances [13] : des instances virtuelles regroupant plusieurs points de données réels. Ainsi, la modification des super-instances se traduit par des changements sur toutes les instances qui la composent, contraints ou non. La création des super-instances est réalisée en décomposant chaque cluster de la partition initiale en petits groupes, chaque groupe représentant une super-instance. On calcule ensuite une matrice de distance $\mathcal{D}'$ de taille $N_s \times K$ avec N_s le nombre de super-instances, où $\mathcal{D}'[i, c]$ représente la distance moyenne des instances de la super-instance i au centre du cluster c. Le nombre de super-instances par cluster est déterminé arbitrairement et a une utilité indirecte pour définir la granularité de la propagation des contraintes : quand N_s tend vers le nombre d'instances du cluster, le nombre moyen d'instances dans une super-instance tend vers 1.
La décomposition des clusters en super-instances est effectuée par un algorithme de clustering exécuté sur chacun des clusters de la partition existante. Conformément à notre idée que les contraintes de l'expert sur des instances symbolisent en fait un ensemble de modifications à effectuer sur ces instances et leur voisinage, nous recherchons des super-instances compactes, ce qui oriente notre choix vers des algorithmes comme le clustering hiérarchique *complete-link*,



(a) Résultat d'une génération de super-instances via un algorithme de clustering hiérarchique ascendant *complete-link* lancé sur chaque cluster. Les instances d'une même couleur sont représentées par la même super-instance.



(b) Super-instances obtenues après division des instances contraintes pour éviter l'émergence de contraintes indésirables. Les instances en gris font partie de super-instances non contraintes et ne sont donc pas pris en compte par le modèle.

FIGURE 2 – Exemple de prétraitement pour la génération de super-instances sur le jeu de données Iris.

FPF [7], ou KMEANS. Le processus est décrit dans la figure 2.

Les contraintes de l'utilisateur liant des instances doivent être transférées aux super-instances afin que notre modèle puisse les traiter. La transmission des contraintes peut avoir comme effet de bord l'émergence de contraintes absentes du jeu de contraintes fourni, et qui peuvent rendre le problème insoluble. La figure 3 présente une illustration de ce problème. Pour éviter cet écueil, nous avons décidé d'ajouter un pré-traitement supplémentaire consistant à diviser les super-instances contenant plusieurs instances contraints. Nous nous servons des instances contraintes comme centres des nouvelles super-instances et nous assignons les instances restantes au centre le plus proche. Ainsi, le problème de la figure 3 est résolu en divisant les super-instances en deux.

Propagation aux plus proches voisins. Notre seconde approche consiste à propager les modifications du modèle en post-traitement aux plus proches voisins des instances modifiées. Cette méthode permet de faire varier facilement la propagation des modifications à partir d'une solution optimale du modèle et est facilement applicable quel que soit le jeu de données. C'est également une méthode qui fait des modifications locales et qui n'est pas sujette aux problèmes de temps et/ou de mémoire qui peuvent arriver avec la méthode des super-instances. Cependant la portée de la propagation par cette voie est moins prévisible, alors que la génération de super-instances définit en amont les instances ciblées par la propagation de modifications ; l'expert peut ainsi être informé à l'avance de la portée des changements.

4 Expérimentations

Les expérimentations visent à :

— estimer l'impact des contraintes sur la qualité du clustering, tout en gardant la structure générale des clusters existants.

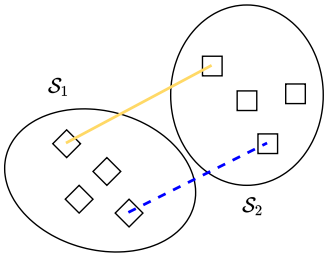
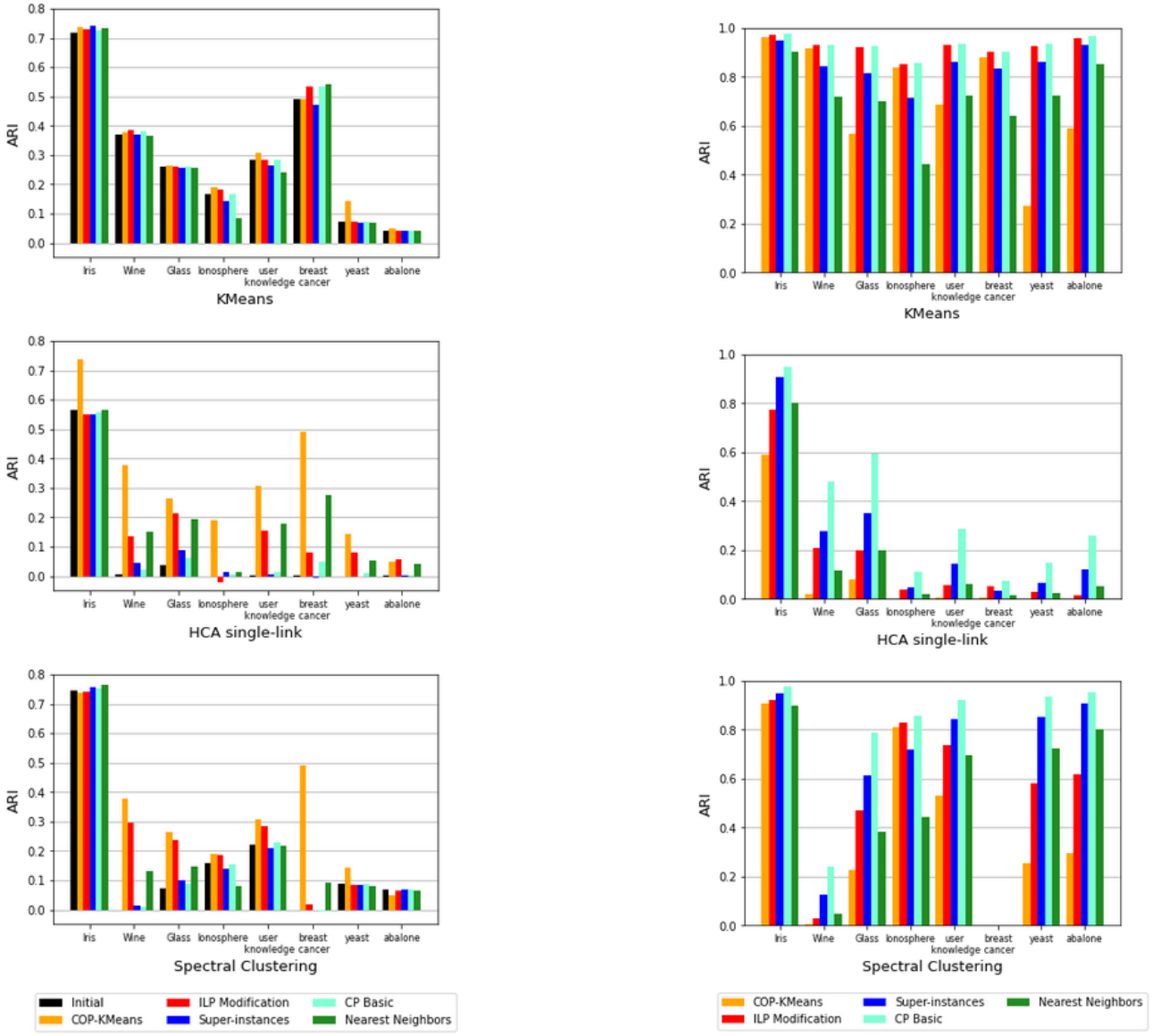


FIGURE 3 – Exemple de contraintes émergentes : une paire d'instances appartenant à deux super-instances $\mathcal{S}_1$ et $\mathcal{S}_2$ sont soumises à une contrainte *must-link* (en orange), tandis qu'une autre paire est soumise à une contrainte *cannot-link* (en bleu pointillés). Il se trouve alors que $\mathcal{S}_1$ et $\mathcal{S}_2$ sont en même temps soumis à une contrainte *must-link* et une contrainte *cannot-link*, ce qui rend le problème insoluble.

— comparer les performances de notre approche par rapport à d'autres méthodes similaires.
— étudier la capacité de relâcher des contraintes.

Implémentation. Le modèle a été implémenté dans un premier temps en C++ avec la librairie *open source* Gecode 6.2.0 [1], puis en Python avec CPMpy [2]. Cette nouvelle librairie fait l'interface avec différents solveurs connus, tels que CP-SAT d'`or-tools` [3], que nous utilisons. Nous avons constaté un *speedup* allant jusqu'à deux ordres de grandeur par rapport à Gecode. En termes de stratégie de recherche, nous choisissons en premier les variables X de plus haut degré puisque leur affectation peut permettre de satisfaire plusieurs contraintes rapidement et avec peu de modifications. Nous comparons notre approche avec une méthode de

1. https://www.gecode.org
2. https://github.com/CPMpy/cpmpy
3. https://developers.google.com/optimization



(a) Valeurs d'ARI entre la vérité terrain et les méthodes

(b) ARI entre la partition existante et les méthodes.

FIGURE 4 – Évolution de l'ARI des partitions calculées avec les différentes méthodes comparées à (a) la vérité des données et (b) les partitions existantes. **Initial** est la partition générée par l'algorithme indiqué en abscisse ; **ILP Modification** est la méthode de post-processing en PLNE présentée dans [11] ; **CP Basic** est le résultat de notre modèle sans propagation ; les meilleurs résultats des méthodes de propagation sont également indiqués par **Super-instances** et **Nearest Neighbors.**

post-processing en PLNE avec Gurobi [4] [11] ainsi qu'une implémentation Python de COP-KMEANS [5] [15].

4.1 Jeux de données de référence

Nous utilisons des jeux de données issus du dépôt pour l'apprentissage automatique de l'UC Irvine [6] décrits en table 1.

4.1.1 Comparaison de performances

Pour chaque jeu de données, nous générons 15 jeux de contraintes *must-link/cannot-link* par une procédure aléatoire : on tire au hasard deux points du jeu de données ; si les points sont dans le même cluster du point de vue de la vérité terrain, on crée une contrainte *must-link*, sinon on crée une contrainte *cannot-link*. Nous simulons ainsi un retour expert sur les données. Chaque jeu de contraintes ainsi créé contient un nombre de contraintes équivalent à 10% de la taille du jeu de données.

Nous avons généré une partition de départ avec trois méthodes : KMEANS, clustering hiérarchique ascendant *single-link* (HCA) et clustering spectral. Lors de l'intégration de contraintes, COP-KMEANS n'utilise pas la partition de départ et est relancé pour chaque jeu de contraintes.

4. https://github.com/dung321046/ConstrainedClusteringViaPostProcessing
5. https://github.com/Behrouz-Babaki/COP-Kmeans
6. https://archive-beta.ics.uci.edu/ml/datasets

Nom	Objets	Attributs	Classes
iris	150	4	3
wine	178	13	3
glass	214	9	7
ionosphere	351	34	2
user knowledge	403	5	4
breast cancer	569	30	2
yeast	1484	8	10
abalone	4177	7	28

TABLE 1 – Jeux de données utilisés, triés par ordre croissant du nombre d'objets. Le nombre d'attributs que possède chaque objet, ainsi que le nombre de classes (et donc de clusters à trouver) sont également indiqués.

Nous effectuons donc une seule itération du processus de clustering par contraintes incrémentales. Dans le cas de plusieurs itérations, la partition produite après une itération deviendrait le point de départ des modifications suivantes.
Nous comparons également les méthodes de propagation avec différents paramètres : pour la propagation par super-instances, nous faisons varier le nombre de super-instances créés selon une proportion de 30%, 40% et 50% du nombre de points total ; pour la seconde méthode, nous avons aussi fait les expériences pour 5, 10, 15 et 20 voisins.
Pour déterminer la performance du clustering, nous utilisons l'indice de Rand ajusté (ARI [8]) entre deux partitions P et P' qui s'exprime ainsi :

$$\frac{2(ab - cd)}{(a+d)(d+b) + (a+c)(c+b)}$$

où a est le nombre de paires de points qui sont dans le même cluster dans les deux partitions, b est le nombre de paires de points dans des clusters différents dans les deux partitions, c est le nombre de paires de points dans le même cluster dans P et dans des clusters différents dans P' et d est le nombre de paires de points dans des clusters différents dans P et dans le même cluster dans P'. L'ARI est compris entre -1 et 1, une valeur de 1 dénotant des partitions identiques, et une valeur de 0 ou inférieure correspondant à des partitions indépendantes entre elles.

Qualité du clustering. Les résultats des comparaisons sont présentés en figure 4. La figure 4a présente les comparaisons des partitions trouvées par les différentes méthodes par rapport à la vérité des données. L'ARI entre la partition initiale et la vérité sert de référence. On peut constater que la partition initiale influe beaucoup sur les résultats car les méthodes ILP et CP se basent sur cette partition pour intégrer les contraintes. La partition trouvée par KMEANS possède en général un meilleur ARI que celles produites par HCA ou clustering spectral. On peut constater qu'en général, la modification produite par ILP ou CP améliore l'ARI quelle que soit la partition initiale. A noter que COP-KMEANS se comporte toujours de la même façon indépendamment de la partition initiale, puisque cet algorithme calcule une nouvelle partition à partir des données et des contraintes, sans tenir compte de la partition initiale.

La figure 4b présente les comparaisons des partitions produites par les méthodes par rapport à la partition initiale. On peut constater que COP-KMEANS produit une partition assez similaire à la partition initiale produite par KMEANS, car les deux méthodes visent la même fonction objectif. Cependant pour les partitions initiales produites par HCA ou le clustering spectral, COP-KMEANS ne peut pas garantir une partition similaire. On constate également que les méthodes ILP et CP produisent une partition très similaire à la partition initiale produite par KMEANS. Cependant, ILP s'écarte plus de la partition initiale dans les cas de HCA et du clustering spectral, alors que notre modèle est celui qui reste le plus proche de la partition de départ dans tous les cas sans propagation, et aussi dans bon nombre de cas où l'on effectue une propagation par les super-instances.

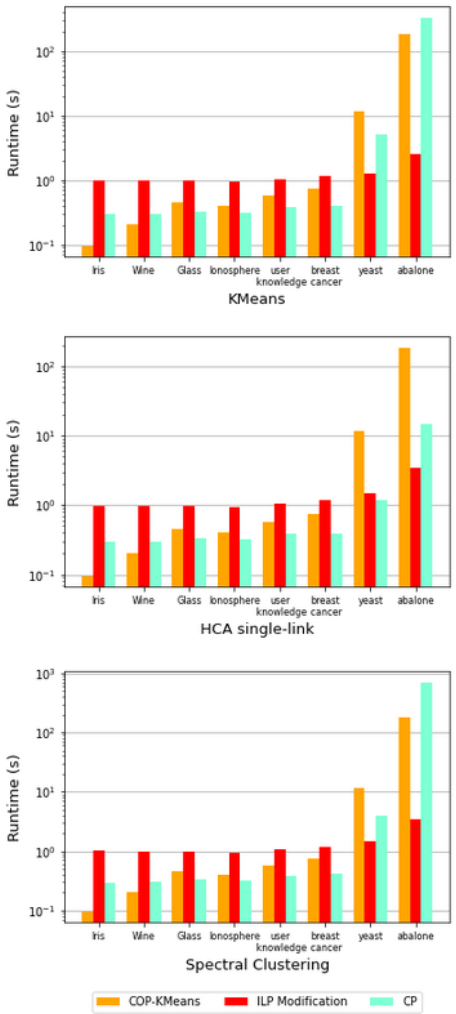


FIGURE 5 – Temps d'exécution en secondes des différentes méthodes (échelle logarithmique)

En effet le modèle ILP utilise le score d'allocation des points aux clusters pour réarranger tous les points, ce qui peut produire d'importantes modifications sur la structure des clusters, y compris sur les points non contraints. Notre modèle, en se restreignant sur les points ou super-instances

contraints, assure des modifications moins importantes tout en améliorant les résultats. De plus les propagations sont localisées au niveau des points contraints, ce qui les rend plus prévisibles pour l'expert.

Temps d'exécution. Nous avons mesuré l'efficacité des méthodes en termes de temps d'exécution. La figure 5 représente le temps d'exécution des méthodes en secondes. On peut constater que l'approche ILP est assez stable en temps alors que l'approche CP peut demander un temps d'exécution important lorsque le nombre de contraintes devient grand. Pour améliorer l'efficacité, une perspective est d'améliorer la stratégie de recherche pour atteindre rapidement des solutions proches de l'optimum.

4.1.2 Relâchement de contraintes

Le modèle ILP [11] ne permet pas de gérer les cas sur-contraints, par exemple lorsque les contraintes utilisateur sont conflictuelles. Notre modèle permet de gérer ces cas grâce à l'utilisation de la PPC. Afin de mettre en évidence l'intérêt de notre modèle dans le cas de problèmes sur-contraints, nous avons utilisé un problème de résolution de 40 contraintes sur Iris avec une partition de départ produite par KMEANS, dans lequel nous avons intentionnellement créé un réseau de trois contraintes rendant le problème insoluble ainsi que deux contraintes *must-link* contradictoires avec la vérité terrain entre des points très éloignés l'un de l'autre. Nous simulons ainsi le cas où des erreurs sont introduites dans le retour expert.

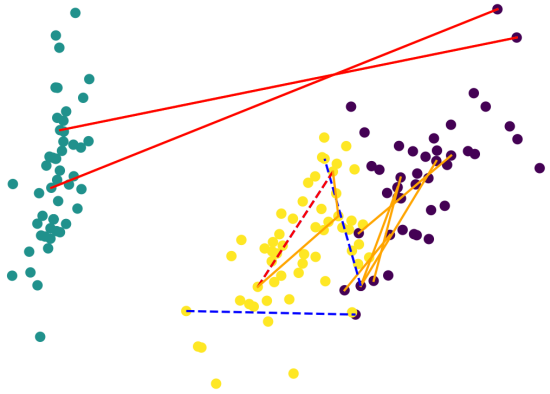


FIGURE 6 – Solution du problème sur-contraint ; les contraintes affichées ne sont pas satisfaites au lancement du modèle ; les contraintes en rouge sont relâchées par le modèle (*must-link* en trait plein, *cannot-link* en pointillés).

La solution obtenue en demandant la satisfaction de 37 contraintes sur les 40 est visible dans la figure 6. On constate que l'une des contraintes du réseau ainsi que les contraintes erronées sont relâchées. L'ARI entre la solution et la vérité terrain est de 0.802, alors qu'il est de 0.764 si l'on demande la satisfaction de 39 contraintes (toutes les contraintes ne pouvant pas être satisfaites à cause du réseau de contraintes contradictoires). Le choix par le modèle des contraintes à relâcher pour minimiser la fonction objectif permet donc dans une certaines mesure de corriger des erreurs dans le retour expert.

Par ailleurs, nous avons mesuré l'évolution du temps de calcul en fonction du taux de satisfaction pour estimer l'impact de la possibilité de relâcher les contraintes. Pour cette expérimentation, nous avons considéré le jeu de données *yeast* avec 10 jeux de 50 contraintes *must-link*/*cannot-link* en proportions égales. Les résultats sont présentés dans la figure 7. On constate que l'augmentation du temps de calcul est significative quand le taux de satisfaction est inférieur à 25% du nombre de contraintes. La variance du temps de calcul entre les jeux de contraintes est également très forte pour ces valeurs.

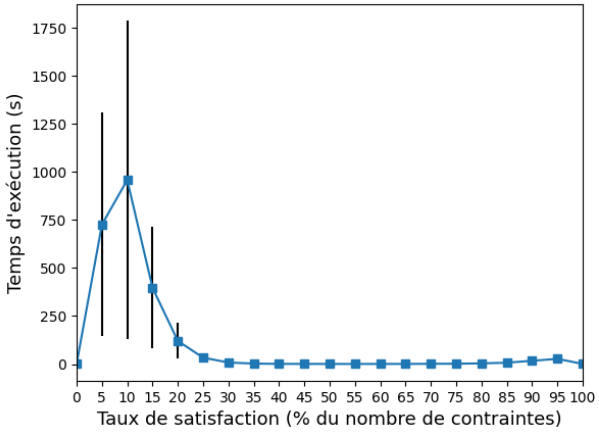


FIGURE 7 – Évolution du temps de calcul du modèle sur le jeu de données *yeast* en fonction de la proportion de contraintes à satisfaire. Les points sans barre d'erreur ont une variance trop faible pour être affichée.

4.2 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) [10]. 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.

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 pro-

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

gressif 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 8 présente la répartition en pixels des classes annotées et met en exergue la complexité de la tâche.

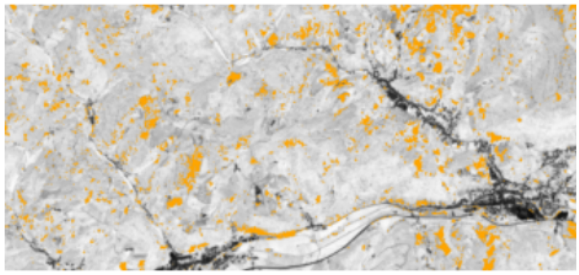
FIGURE 8 – Répartition des clusters labellisés sur la série d'images satellite du jeu de données *Coupes de bois*. La forêt est représentée en vert (156684 pixels), les zones d'activité humaine en bleu (86665 pixels) et les zones de coupes claires en jaune (639 pixels).

Présentation du problème. Étant donné que la seule classe pour laquelle nous avons des informations précises est celle des coupes de bois, nous nous donnons comme but de distinguer ce groupe du reste des données ; il s'agit ainsi d'un problème de clustering binaire.
Les données de départ décrivent l'évolution temporelle du NDVI des images satellites. Une donnée x_i est donc constituée de 11 valeurs NDVI $x_{i1}, \ldots, x_{i11}$. Pour chaque donnée, nous avons construit une série dérivée $x'_{i1}, \ldots, x'_{i10}$, en calculant la différence de NDVI entre chaque pas de temps $x'_{ij} = x_{ij+1} - x_{ij}$. Pour tenter de trouver un cluster proche des annotations d'experts, nous avons suivi la méthode décrite dans [10] consistant à effectuer un clustering avec $k = 15$ sur les données et à sélectionner le cluster avec la plus grande intersection avec la classe de coupes claires. Nous avons remarqué que le clustering sur les variations de NDVI permettait d'obtenir un cluster couvrant plus de points de coupes de bois que l'utilisation des données de base. Les données sélectionnées sont présentées en orange dans la figure 9a. Il s'agit d'un ensemble de 16183 pixels, dont 348 qui sont des coupes claires, soit plus de la moitié des points de coupe totaux.
On sépare une nouvelle fois ces données en deux en espérant distinguer ainsi les coupes de bois des autres évolutions similaires (fauchage, récoltes). Cette partition nous sert de point de départ. Elle contient un cluster de petite taille (561 points) contenant une majorité de points de coupe. C'est ce cluster qui est identifié comme les points de coupes. Nous représentons ce cluster dans la figure 9b, où les points de coupe réels (vrais positifs) sont indiqués en vert, et les points qui ne sont pas des coupes (faux positifs) sont indiqués en rouge. Le reste de la partition de départ reste affichée en orange. Les points de coupe non sélectionnés, qui sont donc absents de la partition de départ, sont indiqués en jaune. Ces zones sont présentées dans la figure 9c afin de mieux les positionner dans la zone géographique initiale.
Nous générons ensuite 25 contraintes sur des points de coupe de bois, sur la base des annotations d'experts. Ces contraintes servent de données d'entrée à notre modèle pour modifier la partition que nous avons construite. La figure 9b présente les contraintes générées, où les contraintes *must-link* sont en violet et *cannot-link* sont en bleu et pointillés.

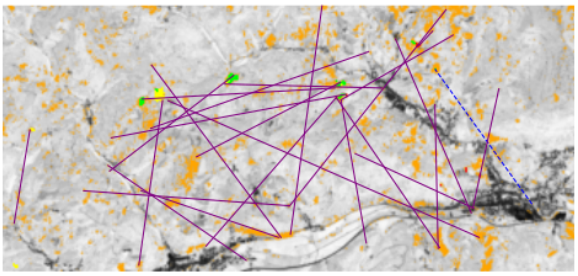
Résultats. Nous présentons dans la figure 9d le résultat obtenu avec la modification par notre modèle, sans utilisation de super-instances et avec la propagation aux 8 plus proches voisins, qui donne les meilleurs résultats d'après nos expériences. On peut constater que la modification en tenant compte de ces contraintes a permis de réduire le nombre de faux positifs. En particulier, la zone de faux positifs près de la zone de coupe au nord-est de la zone géographique est réduite après modification par notre modèle. La modification en elle-même est réalisée en 2.24 secondes en moyenne.
Nous avons analysé l'impact des paramètres sur le nombre de super-instances et le nombre de plus proches voisins, le résultat est présenté dans la figure 10. On peut constater qu'un très faible nombre de super-instances entraîne des modifications trop importantes, ce qui se traduit par une chute de l'ARI. Le meilleur résultat est obtenu pour un taux de super-instances équivalent à environ 50% du nombre de points de chaque cluster. La propagation par les plus proches voisins est cependant moins sensible à l'augmentation du nombre de voisins. Le choix d'un nombre de voisins entre 5 et 10 pour propager efficacement les modifications semble raisonnable. Enfin, le temps de génération des super-instances calculé en utilisant le clustering hiérarchique *complete-link* est de 8.73 secondes.

5 Conclusion

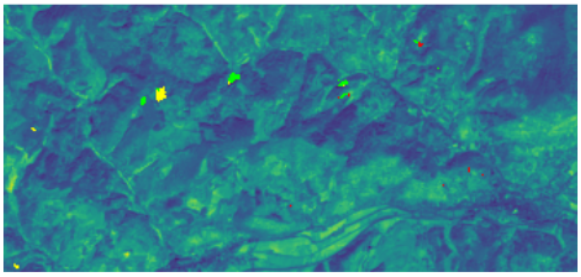
Nous avons développé un modèle en programmation par contraintes pour la modification d'un clustering, permettant d'une part l'intégration incrémentale de contraintes et d'autre part de préserver la structure des clusters. Il peut également traiter différents types de contraintes, dont des contraintes thématiques, et relâcher des contraintes pour ré-



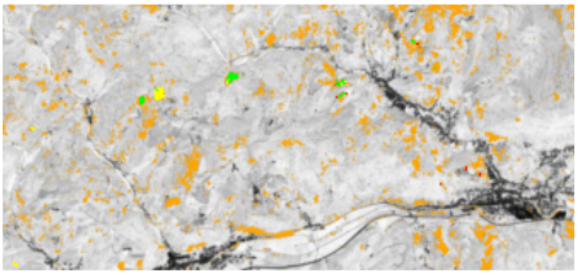
(a) En orange, le cluster couvrant le plus de coupes de bois.



(b) Mise en évidence des coupes de bois et des contraintes.

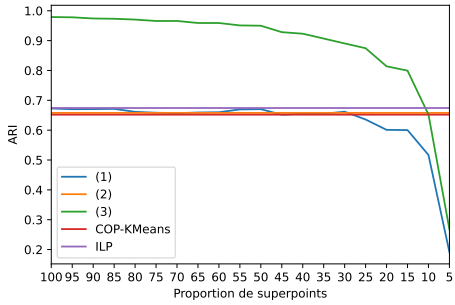


(c) Image d'origine.

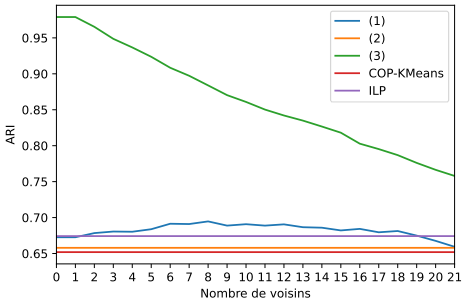


(d) Cluster modifié par notre modèle.

FIGURE 9 – Le cluster utilisé pour l'étude de cas est affiché en orange. La première image utilise les canaux d'origine, la seconde est en niveaux de gris selon la valeur NDVI. Les vrais positifs sont en vert, les faux positifs en rouge, les points de coupe absents des données considérées en jaune.



(a) Propagation par les super-instances



(b) Propagation par les proches voisins

FIGURE 10 – Évolution de l'ARI selon les méthodes de propagation sur la partition construite à partir des données de coupes de bois. On considère l'ARI entre la partition modifiée et la vérité terrain (1), entre la partition initiale et la vérité terrain (2), entre la partition initiale et la partition modifiée (3). En outre, l'ARI entre les résultats de COP-KMEANS (ou ILP) et la vérité terrain sont indiqués pour comparaison.

soudre des conflits ou corriger des erreurs dans le retour de l'expert. Ces atouts rendent ce modèle adapté à l'utilisation dans un contexte d'interaction avec un expert pour des tâches de clustering. L'efficacité des contraintes sur la qualité du clustering peut être améliorée par des processus d'apprentissage actif pour tirer au mieux avantage du retour expert. La contrainte du temps réel pour l'interaction signifie qu'il faut également étudier des stratégies de recherche adaptées pour obtenir rapidement des solutions proches de l'optimum.

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) [7].

Références

[1] Arindam Banerjee and Joydeep Ghosh. Scalable clustering algorithms with balancing constraints. *Data Min. Knowl. Discov.*, 13(3) :365–395, 2006.

7. https://anr.fr/Projet-ANR-20-CE23-0022

[2] M. Bilenko, S. Basu, and R. J. Mooney. Integrating constraints and metric learning in semi-supervised clustering. In *Proceedings of the 21st International Conference on Machine Learning*, pages 11–18, 2004.

[3] Thi-Bich-Hanh Dao, Khanh-Chuong Duong, and Christel Vrain. Constrained clustering by constraint programming. *Artificial Intelligence*, 244 :70–94, March 2017.

[4] Ian Davidson and S. S. Ravi. Agglomerative Hierarchical Clustering with Constraints : Theoretical and Empirical Results. In Alípio Mário Jorge, Luís Torgo, Pavel Brazdil, Rui Camacho, and João Gama, editors, *Knowledge Discovery in Databases : PKDD 2005*, Lecture Notes in Computer Science, pages 59–70, Berlin, Heidelberg, 2005. Springer.

[5] Ian Davidson, S. S. Ravi, and Leonid Shamis. A SAT-based Framework for Efficient Constrained Clustering. In *Proceedings of the 2010 SIAM International Conference on Data Mining*, pages 94–105, Columbus, OH, USA, April 2010. Society for Industrial and Applied Mathematics.

[6] 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.

[7] Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38 :293–306, 1985.

[8] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of Classification*, 2(1) :193–218, December 1985.

[9] Chia-Tung Kuo, S. S. Ravi, Thi-Bich-Hanh Dao, Christel Vrain, and Ian Davidson. A framework for minimal clustering modification via constraint programming. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, pages 1389–1395, San Francisco, CA, USA, February 2017. AAAI Press.

[10] Thomas Lampert, Baptiste Lafabregue, Thi-Bich-Hanh Dao, Nicolas Serrette, Christel Vrain, and Pierre Gancarski. 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.

[11] Nguyen-Viet-Dung Nghiem, Christel Vrain, Thi-Bich-Hanh Dao, and Ian Davidson. Constrained Clustering via Post-processing. In Annalisa Appice, Grigorios Tsoumakas, Yannis Manolopoulos, and Stan Matwin, editors, *Discovery Science*, Lecture Notes in Computer Science, pages 53–67, Thessaloniki, Greece, 2020. Springer International Publishing.

[12] Abdelkader Ouali, Samir Loudni, Yahia Lebbah, Patrice Boizumault, Albrecht Zimmermann, and Lakhdar Loukil. Efficiently finding conceptual clustering models with integer linear programming. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, pages 647–654, New York, NY, USA, July 2016. AAAI Press.

[13] Toon Van Craenendonck, Sebastijan Dumančić, Elia Van Wolputte, and Hendrik Blockeel. COBRAS : Fast, Iterative, Active Clustering with Pairwise Constraints. *arXiv :1803.11060 [cs, stat]*, March 2018. arXiv : 1803.11060.

[14] Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schrödl. Constrained k-means clustering with background knowledge. In Carla E. Brodley and Andrea Pohoreckyj Danyluk, editors, *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001), Williams College, Williamstown, MA, USA, June 28 - July 1, 2001*, pages 577–584. Morgan Kaufmann, 2001.

[15] Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schroedl. Constrained K-means Clustering with Background Knowledge. page 8, 2001.

[16] Xiang Wang and Ian Davidson. Flexible constrained spectral clustering. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '10, pages 563–572, New York, NY, USA, July 2010. Association for Computing Machinery.

Session 4 : Max-SAT

Certificats d'optimalité pour Max-SAT

Matthieu Py, Mohamed Sami Cherif, Djamal Habet

Aix-Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France

{ matthieu.py, mohamed-sami.cherif, djamal.habet}@univ-amu.fr

Résumé

Dans ce papier, on présente un outil, MS-Builder, qui génère des certificats pour le problème Max-SAT, en appelant itérativement un oracle SAT pour générer une réfutation par résolution de la formule qui est ensuite rendue valide pour Max-SAT et appliquée sur la formule. Ce procédé est répété jusqu'à ce que la formule restante devienne satisfiable. On propose également un outil, MS-Checker, capable de vérifier les certificats Max-SAT basés sur l'application de règles d'inférence Max-SAT. Cet article résume les travaux publiés à la conférence SAT 2021 [4].

Mots-clés

Optimisation Combinatoire, Max-SAT, Certificats d'optimalité.

1 Introduction

Étant donné une formule sous Forme Normale Conjonctive (FNC), le problème Max-SAT consiste à déterminer le nombre maximum de clauses qu'il est possible de satisfaire par une affectation des variables alors que le problème SAT consiste simplement à déterminer si la formule est satisfiable. Dans le contexte du problème SAT, une formule peut être démontrée insatisfiable à l'aide d'une séquence de résolutions [5], appelée réfutation par résolution, qui déduit de la formule initiale de nouvelles clauses jusqu'à en déduire la clause vide qui, par définition, est impossible à satisfaire. Ainsi, les solveurs SAT modernes, en plus de proposer la résolution du problème SAT, fournissent aussi la preuve de la réponse affirmée, soit sous forme de modèle (si la formule est satisfiable), soit sous forme de réfutation (si elle ne l'est pas). En revanche, si les solveurs Max-SAT sont de plus en plus efficaces chaque année, ils ne produisent pas encore de certificats d'optimalité sur la solution trouvée. Les systèmes de preuve pour Max-SAT ont en effet principalement été étudiés théoriquement ou pour leur apport à la résolution (sans preuve) du problème Max-SAT.

Dans cet article, on propose un outil indépendant, MS-Builder, capable de produire des certificats d'optimalité pour le problème Max-SAT. Cet outil se base sur l'adaptation des réfutations par résolution pour Max-SAT dans [3] auquel on ajoute un cas particulier détaillé dans [4]. Itérativement, l'outil fait appel à un oracle SAT pour obtenir une réfutation par résolution de la formule courante. Cette réfutation est adaptée en max-réfutation (qui utilise les règles de la max-résolution [2] et du split [3]), puis appliquée sur la formule. Ce procédé est répété jusqu'à ce que l'oracle SAT détecte que la formule est devenue satisfiable. L'ensemble des transformations calculées, plus le modèle pour la formule résiduelle, forme un certificat d'optimalité pour le problème Max-SAT. On propose aussi un outil associé, MS-Checker, pour vérifier les certificats d'optimalité basés sur les règles d'inférence Max-SAT. Ces deux outils ont été testés expérimentalement sur les instances partielles non pondérées de la Compétition Max-SAT 2020 [1].

2 Fonctionnement de MS-Builder

Exemple 1. *On considère la formule* $\phi = (\overline{x_1} \vee x_3) \wedge (x_1) \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_2}) \wedge (\overline{x_3}) \wedge (x_2 \vee x_3)$.

Lecture du fichier : *MS-Builder récupère la formule* ϕ *encodée dans le fichier représenté dans la figure 1.*

```
c formule avec optimum = 2
1 -1 3 0
1 1 0
1 -1 2 0
1 -2 -3 0
1 -2 0
1 -3 0
1 2 3 0
```

FIGURE 1 – Fichier avec formule

Première itération : *MS-Builder fait appel à un oracle SAT, qui retourne que la formule initiale est insatisfiable, avec la réfutation par résolution de la figure 2, qui est adaptée en la max-réfutation de la figure 3 puis appliquée sur la formule. La formule courante est maintenant* $\phi = (\overline{x_1} \vee x_3) \wedge (x_1) \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_2} \vee \overline{x_3}) \wedge \square$.

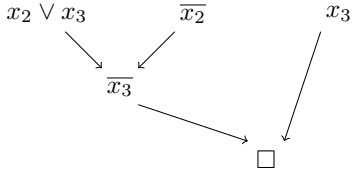


FIGURE 2 – Première réfutation par résolution

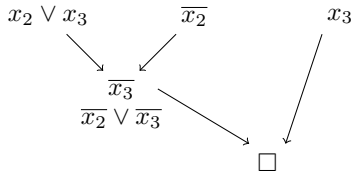


FIGURE 3 – Adaptation de la première réfutation

Deuxième itération : *MS-Builder fait appel à un oracle SAT (en laissant de côté la clause vide), qui retourne que la formule initiale est insatisfiable, avec la réfutation par résolution de la figure 4, qui est adaptée en la max-réfutation de la figure 5 puis appliquée sur la formule. La formule courante est maintenant* $\phi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge \square \wedge \square$.

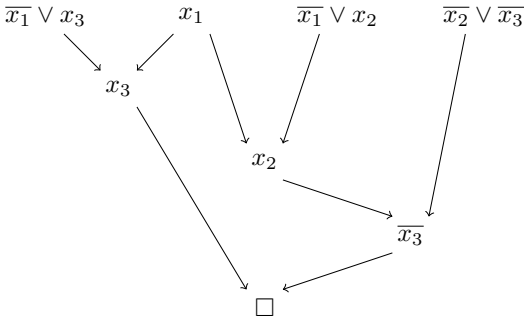


FIGURE 4 – Deuxième réfutation par résolution

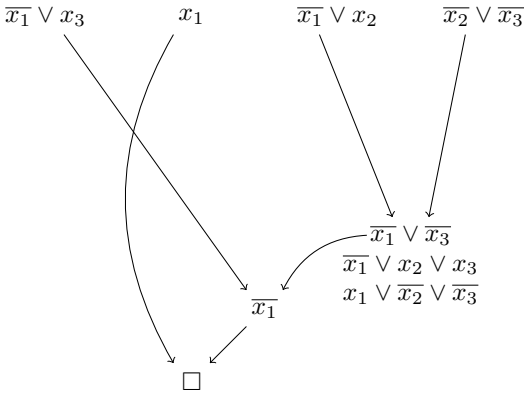


FIGURE 5 – Adaptation de la deuxième réfutation

Troisième itération : *MS-Builder fait appel à un oracle SAT (en laissant de côté les deux clauses vides), qui retourne que la formule initiale est satisfiable, avec l'affectation* $x_1 = x_2 = x_3 = 0$. *Il retourne donc le certificat Max-SAT représenté dans la figure 6, constitué de cinq étapes de max-résolution, plus l'affectation qui permet de satisfaire les clauses non vides restantes après transformation de la formule.*

```
c read-once refutation
t msres < 1 2 3 | 1 -2 >
t msres < 1 3 | 1 -3 >
c semi-read-once refutation
t msres < 1 -1 2 | 1 -2 -3 >
t msres < 1 -1 3 | 1 -1 -3 >
t msres < 1 1 | 1 -1 >
o 2
v 000
```

FIGURE 6 – Certificat d'optimalité Max-SAT

3 Expérimentations

MS-Builder et MS-Checker sont mis à disposition de la communauté scientifique (`https://pageperso.lis-lab.fr/matthieu.py/en/software.html`) et ont été testés expérimentalement sur les instances partielles non pondérées de la Compétition Max-SAT 2020 [1]. En particulier, MS-Builder a été capable de construire des certificats d'optimalité complets pour 163 instances sur 576 instances (figure 7).

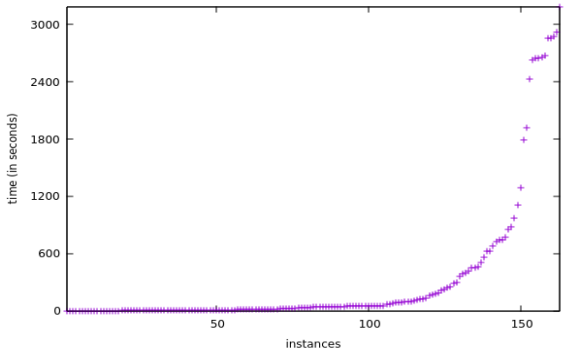


FIGURE 7 – Temps d'exécution (en secondes) pour la construction de certificats complets

Références

[1] Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. MaxSAT Evaluation, 2020.

[2] María Luisa Bonet, Jordi Levy, and Felip Manyà. A complete calculus for MAX-SAT. In *Theory and Applications of Satisfiability Testing - SAT 2006*, volume 4121, pages 240–251, 08 2006.

[3] Matthieu Py, Mohamed Sami Cherif, and Djamal Habet. Towards Bridging the Gap Between SAT and Max-SAT Refutations. In *32nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2020*, pages 137–144. IEEE, 2020.

[4] Matthieu Py, Mohamed Sami Cherif, and Djamal Habet. A Proof Builder for Max-SAT. In *Proceedings of the 24th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 2021.

[5] John Alan Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *Journal of the Association for Computing Machinery*, 12 :23–41, 1965.

Max-réfutations et oracles SAT

Matthieu Py, Mohamed Sami Cherif, Djamal Habet

Aix-Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France

{ matthieu.py, mohamed-sami.cherif, djamal.habet}@univ-amu.fr

Résumé

Adapter une réfutation par résolution en max-réfutation sans en augmenter considérablement sa taille est une question ouverte depuis l'introduction de la max-résolution. Cet article contribue à cette problématique en proposant un algorithme nommé algorithme de génération de remplaçants, capable d'adapter n'importe quelle réfutation par résolution en max-réfutation grâce à des appels à un oracle SAT. En particulier, on démontre que cet algorithme adapte efficacement les motifs en diamant, dont l'adaptation est exponentielle dans la littérature. Cet article résume les travaux publiés à la conférence ICTAI 2021 [4].

Mots-clés

Max-SAT, Réfutation par résolution, Max-Réfutation.

1 Introduction

Étant donné une formule sous Forme Normale Conjonctive, le problème Max-SAT consiste à déterminer le nombre maximum de clauses qu'il est possible de satisfaire par une affectation des variables alors que le problème SAT consiste simplement à déterminer si la formule est satisfiable. Dans le contexte du problème SAT, une formule peut être démontrée insatisfiable à l'aide d'une séquence de résolutions [5], appelée réfutation par résolution, qui déduit de la formule initiale de nouvelles clauses jusqu'à en déduire la clause vide. Un exemple de réfutation par résolution est présenté dans l'exemple 1. De même, pour Max-SAT, on utilise un système de preuve bien connu basé sur la règle de la max-résolution (définition 1) [1], qui étend la règle de résolution utilisée dans SAT. Les séquences de max-résolutions sont plus contraintes que les séquences de résolutions car la max-résolution remplace les prémisses par les conclusions, ce qui consomme les prémisses et empêche de les utiliser à nouveau. Adapter une réfutation par résolution utilisée dans le cadre du problème SAT en une réfutation valide pour Max-SAT est par conséquent un problème difficile et jusqu'à il y a peu, on ne savait le faire que si la formule en entrée est *read-once* (comme dans la figure 1), c'est à dire si chaque clause est utilisée une seule fois comme prémisse d'une résolution. Des travaux récents proposent une adaptation de n'importe quelle réfutation par résolution pour Max-SAT grâce à l'utilisation de la règle du split (définition 2) mais celle-ci entraîne une augmentation exponentielle de la taille de la formule dans le pire des cas [2]. Malgré cela, ces travaux ont permis d'introduire le premier générateur de preuve indépendant pour Max-SAT [3].

Dans cet article, on propose d'apporter une autre alternative pour l'adaptation des réfutations par résolution pour Max-SAT (on parle de max-réfutation). Pour cela, on utilise un algorithme qui suit la réfutation et remplace chaque résolution par une max-résolution et, lorsqu'il a besoin d'un remplaçant pour une clause qui a été consommée, fait appel à un oracle SAT pour que ce remplaçant soit créé.

Exemple 1. *Soit* $\phi = (\overline{x_1} \vee x_3) \wedge (x_1) \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_2} \vee \overline{x_3})$. *Une réfutation de* ϕ *est représentée dans la figure 1.*

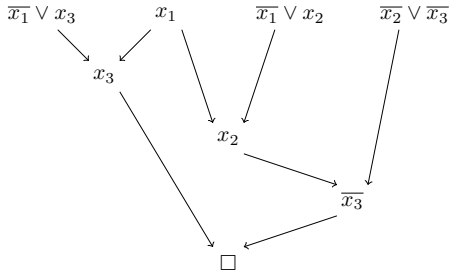


FIGURE 1 – Réfutation par résolution

Définition 1 (Max-Résolution [1])**.** *Étant données deux clauses* $c_1 = x \vee A$ *et* $c_2 = \overline{x} \vee B$ *avec* $A = a_1 \vee \cdots \vee a_s$ *et* $B = b_1 \vee \cdots \vee b_t$*, la max-résolution remplace* c_1 *et* c_2 *par un ensemble de clauses où* c_3 *est la clause résolvante et* $cc_1, \ldots, cc_{t+s}$ *sont des clauses de compensation :*

$$\frac{c_1 = x \vee A \qquad c_2 = \overline{x} \vee B}{\begin{array}{c} c_3 = A \vee B \\ cc_1 = x \vee A \vee \overline{b_1} \\ \cdots \\ cc_t = x \vee A \vee b_1 \vee \cdots \vee b_{t-1} \vee \overline{b_t} \\ cc_{t+1} = \overline{x} \vee B \vee \overline{a_1} \\ \cdots \\ cc_{t+s} = \overline{x} \vee B \vee a_1 \vee \cdots \vee a_{s-1} \vee \overline{a_s} \end{array}}$$

Définition 2 (Scission (ou Split))**.** *L'application de la règle du split sur la clause* $c_1 = (A)$ *où* A *est une disjonction de littéraux et sur une variable* x *remplace* c_1 *par deux clauses* c_2 *et* c_3 *comme suit :*

$$\frac{c_1 = (A)}{c_2 = (x \vee A) \quad c_3 = (\overline{x} \vee A)}$$

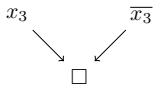
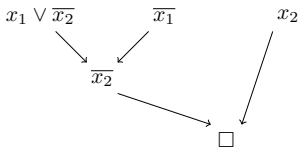
Étape	Résolution initiale	Déroulement de l'algorithme	Formule courante
1	$(\overline{x_1} \vee x_3) \wedge (x_1) \rightarrow (x_3)$	$(\overline{x_1} \vee x_3) \wedge (x_1) \rightarrow (x_3) \wedge (x_1 \vee \overline{x_3})$	$(\overline{x_1} \vee x_2) \wedge (\overline{x_2} \vee \overline{x_3}) \wedge (x_3) \wedge (x_1 \vee \overline{x_3})$
2	$(x_1) \wedge (\overline{x_1} \vee x_2) \rightarrow (x_2)$	Réfutation retournée par l'oracle après propagation de $\overline{x_1}$: 	$(\overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_3) \wedge (x_2) \wedge (x_1 \vee \overline{x_2})$
3	$(x_2) \wedge (\overline{x_2} \vee \overline{x_3}) \rightarrow (\overline{x_3})$	$(x_2) \wedge (\overline{x_2} \vee \overline{x_3}) \rightarrow (\overline{x_3}) \wedge (x_2 \vee x_3)$	$(\overline{x_1} \vee x_3) \wedge (x_1 \vee \overline{x_2}) \wedge (\overline{x_3}) \wedge (x_2 \vee x_3)$
4	$(x_3) \wedge (\overline{x_3}) \rightarrow \square$	Réfutation retournée par l'oracle après propagation de $\overline{x_3}$: 	$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge \square$

TABLE 1 – Exécution de l'algorithme de génération de remplaçants

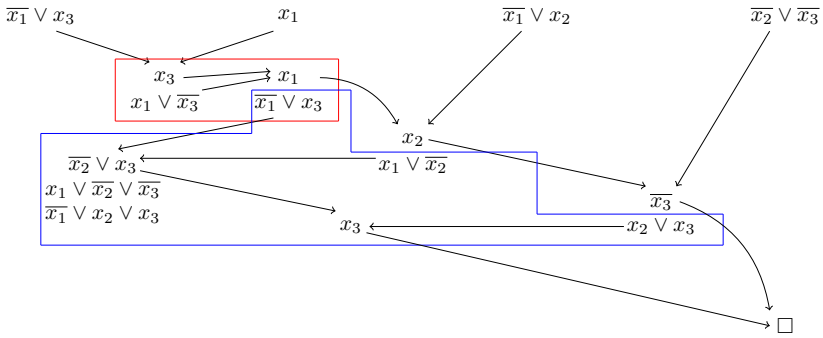


FIGURE 2 – Max-réfutation calculée grâce à l'algorithme de génération de remplaçants

2 L'algorithme de génération de remplaçants

L'idée de l'algorithme de génération de remplaçants est de suivre la liste des étapes de résolution de la réfutation initiale, de remplacer chaque étape de résolution par une max-résolution, et d'appliquer cette étape sur la formule courante. Au cours du processus, si une clause nécessaire à une étape de max-résolution est manquante, on fait appel à un oracle SAT pour regénérer cette clause. Pour cela, on propage la falsification de la clause manquante et toute réfutation par résolution de la formule permet d'inférer la clause manquante. Notons que le procédé est réappliqué récursivement pour générer le remplaçant. La correction de l'algorithme est établie dans le théorème suivant ou $MS(P)$ dénote la projection de P dans Max-SAT [4]. Notons que l'algorithme est capable de transformer linéairement des motifs démontrés exponentiels pour l'adaptation dans [2].

Théorème 1. *Soit ϕ une formule insatisfaisante et P une réfutation par résolution de ϕ. Il existe une max-réfutation de ϕ contenant toutes les étapes de max-résolution de $MS(P)$ dans le même ordre.*

Exemple 2. *On considère la réfutation par résolution de l'exemple 1. Une max-réfutation de la même formule est donnée dans la figure 2. Les étapes de l'algorithme de génération pour construire cette max-réfutation sont détaillées dans le tableau 1. Notons que la preuve entourée en rouge (resp. bleu) dans la figure 2 est obtenue après annulation de la propagation de $\overline{x_1}$ (resp. $\overline{x_3}$) sur la réfutation par résolution obtenue par l'oracle dans l'étape 2 (resp. 4).*

Références

[1] María Luisa Bonet, Jordi Levy, and Felip Manyà. Resolution for Max-SAT. *Artificial Intelligence Volume 171, Issues 8–9, June 2007, Pages 606-618*, 2007.

[2] Matthieu Py, Mohamed Sami Cherif, and Djamal Habet. Towards Bridging the Gap Between SAT and Max-SAT Refutations. In *32nd IEEE International Conference on Tools with Artificial Intelligence, (ICTAI)*, pages 137–144. IEEE, 2020.

[3] Matthieu Py, Mohamed Sami Cherif, and Djamal Habet. A Proof Builder for Max-SAT. In *Proceedings of the 24th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 2021.

[4] Matthieu Py, Mohamed Sami Cherif, and Djamal Habet. Computing Max-SAT Refutations using SAT Oracles. In *33rd IEEE International Conference on Tools with Artificial Intelligence, (ICTAI)*. IEEE, 2021.

[5] J. A. Robinson. A machine-oriented logic based on the resolution principle. In *Journal of the Association for Computing Machinery, vol. 12, pages 23–41*, 1965.

Explicabilité dans Max-SAT

Matthieu Py, Mohamed Sami Cherif, Djamal Habet

Aix-Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France

{matthieu.py, mohamed-sami.cherif, djamal.habet}@univ-amu.fr

Résumé

Dans ce papier, on s'intéresse à la construction de transformations préservant l'équivalence Max-SAT afin d'inférer de l'information (clause ou formule) à partir d'une formule donnée. Dans ce but, on définit et caractérise la notion de clauses et formules explicables, on propose un système de preuve complet pour l'inférence dans Max-SAT et un algorithme associé pour établir ou réfuter l'explicabilité de n'importe quelle clause ou formule. On donne enfin des bornes théoriques sur la taille des transformations calculées. Cet article résume les travaux publiés à la conférence ICTAI 2021 [4].

Mots-clés

Max-SAT, Systèmes de preuve, Inférence.

1 Introduction

Étant donné une formule sous Forme Normale Conjonctive, le problème Max-SAT consiste à déterminer le nombre maximum de clauses qu'il est possible de satisfaire par une affectation des variables. Un système de preuve bien connu pour Max-SAT est basé sur la règle de la max-résolution [2] qui est l'adaptation de la règle de résolution définie dans le contexte du problème SAT [5]. La max-résolution, en tant que système de preuve, est complète pour la résolution du problème Max-SAT, c'est-à-dire qu'il est possible, à partir de n'importe quelle formule sous Forme Normale Conjonctive (FNC) ϕ, de déduire une formule équivalente composée d'une sous-formule satisfiable et d'un multi-ensemble de clauses vides, dont la taille est l'optimum de ϕ.

Cet article s'intéresse aussi aux systèmes de preuve pour Max-SAT et en particulier à la capacité qu'ont ces systèmes pour la déduction d'informations (sous forme de clause ou formule) à partir d'une formule donnée. Comme la max-résolution est inférentiellement incomplète [3], on propose alors un nouveau système de preuve (ExC) inférentiellement complet dont on étudie les relations avec d'autres systèmes de preuve connus. On propose aussi un algorithme (l'algorithme d'explication) permettant d'expliquer ou de démontrer l'inexplicabilité de n'importe quelle clause ou formule dans Max-SAT. On démontre enfin les bornes théoriques sur la taille, en nombre d'étapes d'inférence, des explications calculées.

2 Explication de clauses

Pour commencer, on introduit la notion d'explicabilité, qui peut s'étendre naturellement aux formules (on détaille ici uniquement l'explication de clauses).

Définition 1 (Clause explicable et explication de clause)**.** *Soit ϕ une formule FNC et c une clause non tautologique, on dit que c est explicable dans ϕ s'il existe une formule FNC ϕ' telle que $\phi \equiv c \wedge \phi'$. La transformation de ϕ vers $c \wedge \phi'$ est appelée explication de c dans ϕ.*

Pour faire l'explication de n'importe quelle clause dans Max-SAT, on va simplement se ramener à deux cas extrêmes de clauses explicables ou inexplicables.

Proposition 1. *Soit ϕ une formule FNC et c une clause non tautologique. Si $\exists c' \in \phi$ telle que c' sous-somme c (les littéraux de c' sont dans c) alors c est explicable dans ϕ.*

Exemple 1. *La clause $(x_1 \vee x_2)$ est explicable dans $\phi = (x_1) \wedge (x_2)$ car elle est sous-sommée par (x_1).*

Proposition 2. *Soit ϕ une formule FNC et c une clause non tautologique. Si $\forall c' \in \phi$, c' s'oppose à c (deux littéraux de c' et c s'opposent), alors c est inexplicable dans ϕ.*

Exemple 2. *La clause $c = (\overline{x_1} \vee \overline{x_2})$ est inexplicable dans $\phi = (x_1) \wedge (x_2)$ car elle s'oppose à toutes ses clauses.*

En testant récursivement si on est dans ces deux cas extrêmes, on peut démontrer l'explicabilité d'une clause en utilisant la caractérisation établie dans le théorème suivant.

Théorème 1. *Une clause c est explicable dans une formule ϕ si et seulement si $\exists c' \in \phi$ telle que c' ne s'oppose pas à c et $\forall x \notin var(c)$, $c \vee x$ et $c \vee \overline{x}$ sont explicables dans ϕ.*

3 Le système de preuve ExC

On propose ici un nouveau système inférentiellement complet, appelé ExC et composé de deux règles d'inférence définies ci-dessous. Ce système est comparé à d'autres systèmes de preuves de la littérature dans figure 1).

Définition 2 (Coupe symétrique)**.** *Soient deux clauses $c_1 = x \vee A$ et $c_2 = \overline{x} \vee A$ où A est une disjonction de littéraux, la coupe symétrique remplace c_1 et c_2 par $c_3 = (A)$:*

$$\frac{c_1 = (x \vee A) \quad c_2 = (\overline{x} \vee A)}{c_3 = (A)}$$

Définition 3 (Expansion)**.** *Soit une clause $c = (A)$ où A est une disjonction de littéraux et $B = b_1 \vee ... \vee b_k$, la règle d'expansion remplace c par les clauses suivantes :*

$$\frac{c = (A)}{\begin{array}{c} cc_1 = (A \vee \overline{b_1}) \\ ... \\ cc_k = (A \vee b_1 \vee ... \vee b_{k-1} \vee \overline{b_k}) \\ c' = (A \vee B) \end{array}}$$

Proposition 3. *La règle d'expansion est valide pour Max-SAT.*

Définition 4. *Le système de preuve ExC est composé de deux règles : la coupe symétrique et l'expansion.*

Proposition 4. *ExC est inférentiellement complet.*

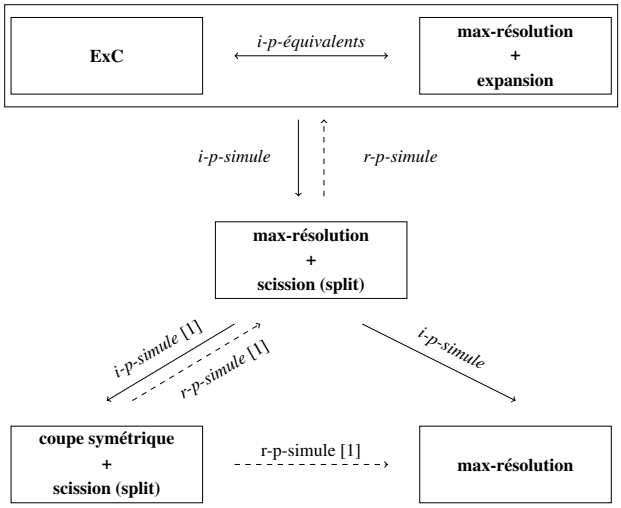


FIGURE 1 – Relation entre **ExC** et d'autres systèmes de preuve de la littérature

4 L'algorithme d'explication

L'algorithme d'explication permet d'expliquer n'importe quelle clause ou de réfuter son explicabilité. Pour cela, on teste si on est dans un des deux cas extrêmes exprimés dans les propositions 1 et 2. Si c'est le cas, on conclut (clause explicable ou non explicable). Sinon, on remplace la clause c par deux clauses $c \vee x$ et $c \vee \overline{x}$ et il suffit d'expliquer ces deux clauses pour expliquer c. Si c est une clause explicable dans la formule ϕ alors l'algorithme décrit ci-dessus est capable de générer une explication dans ExC contenant $O(2^n)$ étapes d'inférence où n est le nombre de variables de ϕ. Ce résultat est établit dans le théorème 2. L'explication de clauses est aussi illustrée dans l'exemple 3.

Théorème 2. *Soit ϕ une formule FNC avec n variables et c une clause explicable dans ϕ. Il existe une explication de c dans ϕ utilisant les règles de ExC et contenant $O(2^n)$ étapes d'inférence.*

Exemple 3. *On considère la formule $\phi = (x_1 \vee x_2) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3) \wedge (\overline{x_1})$ et on veut expliquer la clause $c = (x_1)$. Une explication de c à partir de ϕ est donnée dans la figure 2.*

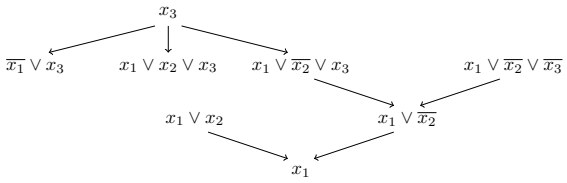


FIGURE 2 – Explication de la clause (x_1) dans ϕ

Notons que l'algorithme d'explication peut être utilisé pour construire naturellement des preuves d'optimalité pour le problème Max-SAT. Intuitivement, l'idée est d'essayer d'expliquer itérativement la clause vide jusqu'à ce qu'elle ne soit plus explicable dans la formule courante. Ainsi, nous déduisons itérativement autant de clauses vides que l'optimum de la formule initiale. Les preuves générées dans ExC contiennent dans ce cas $O(m \times 2^n)$ étapes d'inférence comme établi dans le théorème suivant.

Théorème 3. *Soit ϕ une formule FNC avec n variables et m clauses. On peut déduire $\phi \vdash \underbrace{\square \wedge \cdots \wedge \square}_{opt(\phi)} \wedge \phi'$ où ϕ' est satisfiable dans ExC en $O(m \times 2^n)$ étapes d'inférence.*

Enfin, notons que nos résultats s'étendent également au problème Max-SAT pondéré, où un poids positif est attribué à chaque clause. Pour cela il suffit d'étendre ExC avec les deux règles définies ci-dessous pour gérer les poids.

Définition 5 (Fold & Unfold)**.** *Étant donné une clause c et deux poids positifs w_1 et w_2, les règles fold et unfold sont respectivement définies comme suit :*

$$\frac{(c, w_1) \quad (c, w_2)}{(c, w_1 + w_2)} \qquad \frac{(c, w_1 + w_2)}{(c, w_1) \quad (c, w_2)}$$

Références

[1] Maria Luisa Bonet and Jordi Levy. Equivalence Between Systems Stronger Than Resolution. In *SAT*, pages 166–181, 2020.

[2] María Luisa Bonet, Jordi Levy, and Felip Manyà. Resolution for Max-SAT. In *Artificial Intelligence*, volume 171, pages 606–618, 2007.

[3] Javier Larrosa and Emma Rollon. Towards a Better Understanding of (Partial Weighted) MaxSAT Proof Systems. In *SAT*, pages 218–232, 2020.

[4] Matthieu Py, Mohamed Sami Cherif, and Djamal Habet. Inferring Clauses and Formulas in Max-SAT. In *33rd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2021*. IEEE, 2021.

[5] John Alan Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *Journal of the Association for Computing Machinery*, 12 :23–41, 1965.

De la résolution à la max-résolution

Mohamed Sami Cherif, Djamal Habet, Matthieu Py

Aix-Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France

{mohamed-sami.cherif, djamal.habet, matthieu.py}@univ-amu.fr

Résumé

Adapter une preuve par résolution en une preuve par max-résolution sans augmenter considérablement sa taille est une question ouverte. En effet, la seule classe dont l'adaptation est connue et triviale est celle où les clauses sont utilisées au plus une fois dans les preuves. Dans ce papier, on propose une nouvelle classe de résolution, appelée résolution sans croisement, dans laquelle les clauses réutilisées plusieurs fois sont exploitées indépendemment pour générer de nouvelles informations. On démontre que les preuves de cette classe peuvent être adaptées en preuves par max-résolution sans augmentation considérable de leur taille en utilisant les clauses de compensation générées par la max-résolution.

Mots-clés

Preuve, Résolution sans croisement, Max-résolution.

Abstract

Adapting a SAT resolution proof into a Max-SAT resolution proof without considerably increasing the size of the proof is an open problem. Read-once resolution, where each clause is used at most once in the proof, represents the only fragment of resolution for which an adaptation using exclusively Max-SAT resolution is known and trivial. Proofs containing non read-once clauses are difficult to adapt because the Max-SAT resolution rule replaces the premises by the conclusions. This paper contributes to this open problem by defining, for the first time since the introduction of Max-SAT resolution, a new fragment of resolution whose proofs can be adapted to Max-SAT resolution proofs without substantially increasing their size. In this fragment, called crossing-free resolution, non read-once clauses are used independently to infer new information thus enabling to bring along each non read-once clause while unfolding the proof until a substitute is required.

Keywords

Proof, Max-SAT, Resolution.

1 Introduction

Le problème Max-SAT est une extension d'optimisation du problème de satisfiabilité (SAT) et consiste, étant donné une formule en Forme Normale Conjonctive (FNC), à déterminer le nombre maximum de clauses qu'il est possible de satisfaire par une interprétation des variables. Ce formalisme bien connu est utilisé pour représenter et résoudre de nombreux problèmes industriels et académiques [3, 4]. SAT et Max-SAT sont fortement liés et partagent de nombreux aspects. En effet, les techniques de résolution pour SAT sont souvent utilisées dans le cadre de Max-SAT, en particulier dans les algorithmes faisant appel à des oracles SAT ou encore dans les algorithmes par séparation et évaluation pour Max-SAT [1, 2, 20]. Cependant, en théorie, combler le fossé entre l'inférence SAT et Max-SAT reste l'un des principaux défis de ces dernières années.

L'un des premiers systèmes de preuve pour Max-SAT est basé sur une règle d'inférence, appelée max-résolution [6, 7, 16], qui est une extension de la règle de résolution [27], introduite dans le contexte de SAT. La max-résolution est la règle d'inférence la plus étudiée dans le cadre Max-SAT, à la fois en théorie et en pratique [1, 5, 17, 18, 22, 23, 26]. Cependant, adapter une preuve par résolution pour obtenir une preuve par max-résolution valide de taille raisonnable reste un problème ouvert. Bonnet et al. affirment que "*il semble difficile d'adapter une preuve par résolution classique pour obtenir une preuve par max-résolution, et démontrer que cela est possible sans augmenter considérablement la taille de la preuve est une question ouverte*"[1]. En effet, contrairement à la résolution, la max-résolution remplace les prémisses par les conclusions. Par conséquent, les preuves par résolution *read-once*, où chaque clause est utilisée une fois dans la preuve, représentent le seul fragment de résolution pour lequel une adaptation immédiate et triviale est possible [7, 12].

Des travaux récents [11, 23] tentent de contourner ce problème en permettant l'utilisation de la règle *split*, qui permet intuitivement de dupliquer une clause en ajoutant un littéral, pour adapter linéairement les réfutations par résolution *tree-like*. Plus précisément, l'adaptation tire parti de la structure de telles preuves et applique le *split* pour dupliquer les clauses utilisées plusieurs fois dans la preuve. Cependant, les preuves résultantes sont dans un nouveau système de preuve, appelé ResS [17], dans lequel la max-résolution est augmentée avec le *split*. Pour combler réellement le fossé entre la résolution SAT et Max-SAT, les clauses réutilisées doivent être inférées au travers des

[1] traduction du passage original en anglais dans [7].

clauses de compensation générées par la max-résolution.
Dans cet article, nous contribuons à ce problème ouvert en identifiant un nouveau fragment de résolution, que nous appelons résolution sans croisement, pour lequel une adaptation utilisant uniquement la max-résolution est possible sans augmenter considérablement la taille de la preuve. Les preuves sans croisement sont définies à l'aide des sous-dérivations des clauses utilisées plusieurs fois. Informellement, ces clauses sont utilisées indépendamment pour déduire de nouvelles informations dans de telles preuves. Nous montrons que l'adaptation de preuves par résolution sans croisement à des preuves par max-résolution sans augmentation de leur taille est possible modulo quelques subtilités syntaxiques mineures. De plus, nous montrons que les motifs diamants, qui sont exponentiels pour l'adaptation introduite dans [23], appartiennent au fragment de résolution sans croisement et peuvent être adaptés sans augmentation de leur taille.
Ce papier est organisé comme suit. Dans, la section 2, on présente quelques définitions et notations, ainsi que le contexte nécessaire et les travaux connexes sur la résolution SAT et Max-SAT. Le fragment de résolution sans croisement est introduit dans la section 3 et son adaptation est présentée dans la section 4. On étudie les motifs diamants et on montre qu'ils peuvent être adaptés sans augmenter leur taille dans la section 5. Enfin, on conclut et discute les perspectives dans la section 6.

2 Préliminaires

2.1 Définitions and Notations

Soit X l'ensemble des variables propositionnelles. Un littéral l est une variable $x \in X$ ou sa négation $\overline{x}$. Une clause C est une disjonction (ou un ensemble) de littéraux. Si $|C| = 1$, C est une clause unitaire. Une formule en Forme Normale Conjonctive (FNC) ϕ est une conjonction (ou un multi-ensemble) de clauses. Une interprétation I : $X \rightarrow \{vrai, faux\}$ affecte une valeur booléenne à chaque variable et peut être représentée comme un ensemble de littéraux. Un littéral l est satisfait (resp. falsifié) par une affectation I si $l \in I$ (resp. $\overline{l} \in I$). Une clause C est satisfaite par une affectation I si au moins un de ses littéraux est satisfait par I, sinon elle est falsifiée par I. La clause vide $\square$ ne contient aucun littéral et est toujours falsifiée. Une clause C est une tautologie si elle contient à la fois un littéral et sa négation, i.e. $\exists l \in C$ tel que $\overline{l} \in C$, et dans ce cas, elle est toujours satisfaite. Une clause C s'oppose à une clause C' si C contient un littéral dont la négation est dans C', i.e. $\exists l \in C$ tel que $\overline{l} \in C'$. On note $var(l)$ et $vars(C)$ les variables apparaissant respectivement dans le littéral l et dans la clause C. La taille d'une clause est le nombre de ces littéraux. Une formule FNC ϕ est satisfaite par une affectation I, que l'on appelle modèle de ϕ, si chaque clause $C \in \phi$ est satisfaite par I, sinon elle est falsifiée par I.
Le problème de Satisfiabilité (SAT) consiste à déterminer s'il existe une affectation I qui satisfait une formule FNC donnée ϕ. Dans le cas où une telle affectation existe, on dit que ϕ est satisfiable, sinon on dit que ϕ est insatisfiable. Le coût d'une interprétation I, noté $cost_I(\phi)$, est le nombre de clauses falsifiées par I. Le problème Max-SAT est une extension d'optimisation de SAT qui, pour une formule CNF donnée ϕ, consiste à déterminer le nombre maximum de clauses pouvant être satisfaites par une affectation des variables. De manière équivalente, il consiste à déterminer le nombre minimum de clauses que chaque affectation doit falsifier, i.e. $\min_I \ cost_I(\phi)$.

2.2 Résolution SAT

Un système de preuve et de réfutation bien connu pour SAT est basé sur la règle de résolution [27]. Étant donné deux clauses opposées, cette règle, définie ci-dessous, déduit une clause résolvante qui peut être ajoutée à la formule. Une preuve (ou dérivation) par résolution d'une clause C à partir d'une formule ϕ est une suite finie de résolutions partant des clauses de ϕ et déduisant C, habituellement représentée comme une suite finie de clauses. Si C est la clause vide $\square$, la preuve est appelée réfutation de ϕ. Une preuve par résolution peut également être représentée sous la forme d'un graphe acyclique orienté dont les nœuds sont des clauses de la preuve ayant deux ou zéro arcs entrants (resp. s'il s'agit de résolvantes ou de clauses de la formule initiale). La taille d'une dérivation par résolution π, notée $s(\pi)$, est le nombre de résolvantes qu'elle contient tandis que sa largeur, notée $w(\pi)$, est la taille maximale de toutes ses clauses.

Définition 1 (Résolution [27])**.** *Étant donné deux clauses opposées C_1 et C_2, la règle de résolution est définie comme suit :*

$$\frac{C_1 = x \vee A \qquad C_2 = \overline{x} \vee B}{C_3 = A \vee B}$$

De nombreuses classes restreintes de résolution ont été étudiées dans la littérature, par exemple la résolution *read-once* [13], la résolution arborescente (*tree-like*) [15] et la résolution linéaire [21] entre autres. En particulier, une preuve par résolution est *read-once* si chaque clause est *read-once*, c'est à dire utilisée au plus une fois dans la preuve. De même, une dérivation par résolution est arborescente si chaque clause intermédiaire, c'est-à-dire résolvante, est utilisée au plus une fois dans la dérivation. La résolution linéaire, définie ci-dessous, se situe entre la résolution arborescente et la résolution générale en termes de complexité de preuves [8, 9]. Notons que, lorsque la première condition de (b) est vérifiée dans la définition, la clause D_i est appelée la clause *input parent* de C_{i+1}.

Définition 2 (Résolution linéaire [21])**.** *Soit ϕ une formule FNC et C une clause. Une dérivation par résolution linéaire de C à partir de ϕ est une suite de clauses $C_1, ..., C_m$ tel que :*

(a) $C_1 \in \phi$ et $C_m = C$

(b) Pour tout $i < m$, C_{i+1} est la résolvante de C_i soit avec une clause D_i de ϕ soit avec une clause C_k pour un certain $k < i$.

2.3 Résolution Max-SAT

Le calculus de max-résolution (MaxRes) est l'un des systèmes de preuve pour Max-SAT les plus étudiés dans la littérature qui repose sur une règle d'inférence étendant la résolution pour Max-SAT [6, 7, 16]. Outre la clause résolvante, cette règle, appelée max-résolution et définie ci-après, introduit de nouvelles clauses dites de compensation et qui sont indispensables pour préserver l'équivalence Max-SAT. En tant que règle valide et complète pour Max-SAT [7], la max-résolution joue un rôle important dans le cadre de Max-SAT [5, 17, 23, 26]. En particulier, pour une formule CNF donnée, il est possible de générer une preuve par max-résolution de son optimum en appliquant l'algorithme de saturation [7]. De plus, cette règle est largement utilisée et étudiée dans le cadre des algorithmes par séparation et évaluation pour Max-SAT [1, 10, 14, 18] et plus marginalement dans le cadre des algorithmes faisant appel à des oracles SAT [12, 22].

Définition 3 (Max-résolution [6, 7, 16])**.** *Étant donné deux clauses opposées C_1 et C_2, la règle de max-résolution est définie comme suit :*

$$\frac{C_1 = x \vee A \qquad C_2 = \overline{x} \vee B}{\begin{array}{c} C_r = A \vee B \\ CC_1 = x \vee A \vee \overline{B} \\ CC_2 = \overline{x} \vee \overline{A} \vee B \end{array}}$$

où C_r est la clause résolvante et CC_1, CC_2 sont les clauses de compensation.

Notons que la réécriture suivante est utilisée pour représenter les clauses de compensation sous forme compacte : $C \vee \overline{a_1 \vee a_2 \vee ... \vee a_n} = (C \vee \overline{a_1}) \wedge (C \vee a_1 \vee \overline{a_2}) \wedge ... \wedge (C \vee a_1 \vee a_2 \vee ... \vee \vee \overline{a_n})$. Cette réécriture a été introduite dans [16] comme une règle récursive pour transformer les clauses de compensation en forme CNF. Cela implique également que la max-résolution dépend de l'ordre des littéraux dans les clause prémisses, comme indiqué dans [7, 16]. Par souci de simplification et en abusant certaines notations, nous autoriserons l'utilisation de cette réécriture comme deux règles à part entière pour manipuler les clauses sous forme compacte. Nous appellerons la réécriture gauche-droite *expansion* et droite-gauche *compactage*. Nous discutons cette subtilité à la suite du théorème 1 dans la section 4.

Une preuve (ou dérivation) par max-résolution d'une formule ϕ' (ou simplement une clause $C \in \phi'$) à partir de ϕ est une séquence finie de max-résolutions à partir des clauses de ϕ et déduisant ϕ', généralement représentée par une suite finie de formules. Nous autoriserons l'ajout de clauses tautologiques à n'importe quelle formule dans la preuve. Nous discutons cette subtilité syntaxique à la fin de la section 4. Une preuve par max-résolution peut également être représentée comme un graphe biparti acyclique orienté dont les nœuds sont soit des clauses, soit des étapes d'inférence (auquel cas ils seront omis pour plus de simplicité). Une séquence d'étapes de max-résolution déduisant une clause vide est appelée réfutation par max-résolution. Notons que d'autres règles d'inférence et systèmes de preuve ont également été étudiés dans le cadre de Max-SAT [5, 17, 19, 26].

Contrairement à la résolution, la max-résolution remplace les prémisses par les conclusions. Larrosa et al. décrivent la max-résolution comme un"*mouvement de connaissances*" [16]. Du fait de cette spécificité, il n'est pas facile d'adapter une preuve par résolution en une preuve par max-résolution. En effet, dans les preuves par résolution, plusieurs étapes de résolution peuvent partager la même prémisse, car les prémisses ne sont pas consommées après l'application d'une étape de résolution. En revanche, les prémisses d'une étape de max-résolution sont consommés après son application. Par conséquent, l'adaptation immédiate d'une preuve par résolution pour Max-SAT n'est possible que si elle est *read-once* [7, 6, 12]. Adapter n'importe quelle preuve par résolution en une preuve pour Max-SAT sans augmenter considérablement sa taille reste un problème ouvert.

Des travaux récents [11, 23] augmentent la règle de max-résolution par la règle *split*, formant un nouveau système plus fort que MaxRes et appelé ResS [17], pour adapter linéairement les réfutations de résolution arborescente en réfutations ResS. Plus précisément, l'adaptation tire parti de la structure de telles preuves et applique le *split*, qui permet intuitivement de dupliquer une clause en ajoutant un littéral, pour dupliquer les clauses utilisées plusieurs fois dans la preuve. De plus, l'algorithme de substitution introduit dans [25] permet également de générer des remplaçants pour les clauses non *read-once* en faisant appel à des oracles SAT, mais aucune garantie n'est fournie sur la taille des réfutations ResS calculées. La résolution *read-once* reste donc le seul fragment de résolution pour lequel une adaptation utilisant exclusivement la max-résolution est possible sans augmenter considérablement la taille de la preuve. Dans la section suivante, nous définissons un nouveau fragment de résolution pour lequel une telle adaptation est possible.

3 Résolution sans croisement

La difficulté majeure dans l'adaptation des preuves par résolution à celles par résolution Max-SAT réside dans l'inférence de remplaçants aux clauses non *read-once*. En effet, de tels remplaçants doivent être naturellement déduits à l'aide de la max-résolution lors du déroulement de la preuve initiale, contrairement aux travaux précédents [11, 23] où les clauses non *read-once* sont artificiellement dupliquées à l'aide de la règle *split* avant de dérouler effectivement la preuve. Dans cette section, nous définissons un nouveau fragment de résolution, appelé résolution sans croisement. L'idée derrière ce raffinement est d'assurer une maniabilité suffisante des preuves en termes de structure afin d'inférer des remplaçants aux clauses non *read-once* lorsque cela est nécessaire. À cette fin, nous définissons ci-dessous la notion de sous-dérivation d'une clause non *read-once*.

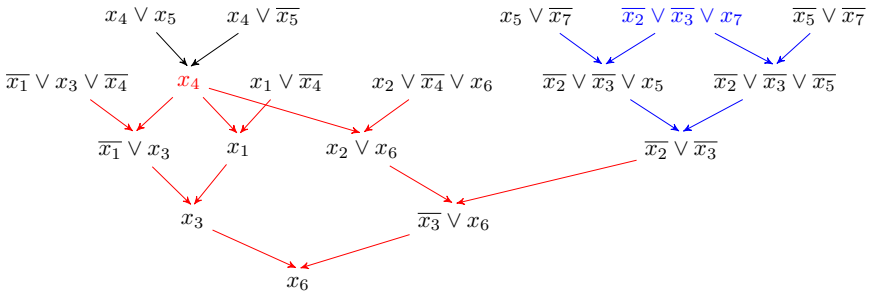


Figure 1: Sous-dérivations de clauses non *read-once* dans une preuve par résolution sans croisement.

Définition 4 (sous-dérivation de clause non *read-once*)**.** *Soit ϕ une formule FNC et π une dérivation par résolution de la clause C à partir de ϕ. La sous-dérivation d'une clause non read-once C' dans π, notée $ED(C')$, est la sous-dérivation de π formée par toutes les étapes de résolution dans les chemins commençant par C' dans π jusqu'à leur premier nœud de jonction. Nous appelons la clause dérivée dans le nœud de jonction, la clause issue de C', notée $EC(C')$.*

Exemple 1. *On considère la dérivation par résolution π représentée dans la figure 1 de la clause $C = x_6$ à partir de la formule $\phi = \{\overline{x_1} \vee x_3 \vee \overline{x_4},\ x_4 \vee x_5,\ x_4 \vee \overline{x_5},\ x_1 \vee \overline{x_4},\ x_2 \vee \overline{x_4} \vee x_6,\ x_5 \vee \overline{x_7},\ \overline{x_2} \vee \overline{x_3} \vee x_7,\ \overline{x_5} \vee \overline{x_7}\}$. Les clauses non read-once x_4 et $\overline{x_2} \vee \overline{x_3} \vee x_7$ et leurs sous-dérivations sont respectivement représentées en rouge et en bleu. De plus, on a $EC(x_4) = x_6$ et $EC(\overline{x_2} \vee \overline{x_3} \vee x_7) = \overline{x_2} \vee \overline{x_3}$.*

Rappelons que les clauses prémisses sont consommées après l'application de la max-résolution. Par conséquent, il semble difficile d'adapter les dérivations par résolution dans lesquelles sous-dérivations de clauses non *read-once* se croisent. En effet, dans de tels cas, la formule peut évoluer significativement car les clauses de compensation peuvent être utilisées tandis que d'autres peuvent être générées. C'est pour cela que la résolution sans croisement garantit que toutes les sous-dérivations de clauses non *read-once* soient disjointes, c'est-à-dire qu'elles ne se croisent pas, comme défini ci-dessous.

Définition 5 (Résolution sans croisement)**.** *Soit ϕ une formule FNC et π une preuve par résolution de la clause C à partir de ϕ. π est dite sans croisement ssi pour chaque paire de clauses non read-once (C_1, C_2) de π, $ED(C_1)$ et $ED(C_2)$ sont disjoints, c'est-à-dire qu'ils ne partagent aucun arc.*

Exemple 2. *On considère la même formule ϕ que l'exemple 1. La preuve par résolution π de la clause $C = x_6$ à partir de ϕ représentée dans la figure 1 est sans croisement puisque les sous-dérivations des clauses non read-once x_4 et $\overline{x_2} \vee \overline{x_3} \vee x_7$ sont disjointes.*

Notons que le fragment de résolution sans croisement implique une propriété intéressante établie dans la proposition suivante. Intuitivement, cette propriété garantit que les clauses non *read-once* soient utilisées indépendamment pour déduire de nouvelles informations dans les preuves par résolution sans croisement. Cela implique que chaque sous-dérivation de clause non *read-once* dans une preuve par résolution sans croisement peut être adaptée indépendamment comme décrit dans la section suivante.

Proposition 1. *Soit ϕ une formule FNC, π une preuve par résolution sans croisement de la clause C à partir de ϕ et C' une clause non* read-once *dans π. toute clause Cl dans $ED(C')$ telle que $Cl \notin \{C', EC(C')\}$ est read-once.*

Preuve. *Soit Cl une clause dans $ED(C')$ telle que $Cl \notin \{C', EC(C')\}$. Clairement, si Cl n'est pas* read-once *alors $ED(Cl)$ partage au moins un arc avec $ED(C')$ ce qui est absurde puisque π est sans croisement.*

4 De la résolution sans croisement à la max-résolution

Dans cette section, nous montrons que les preuves par résolution sans croisement peuvent être adaptées en dérivations par max-résolution modulo quelques subtilités syntaxiques mineures sans augmenter considérablement leur taille. Dans la proposition suivante, nous fournissons d'abord quelques motifs qui seront utilisés dans l'adaptation.

Proposition 2. *Soient A, B, C et $\{l\}$ quatre ensembles de littéraux tels que $|C| > 0$. Les déductions suivantes peuvent être effectuées en $O(|C|)$ étapes d'inférence :*

(a) $(A \vee C) \wedge (B \vee \overline{C}) \vdash_{MaxRes} A \vee B$

(b) $(l \vee A \vee \overline{C}) \wedge (\overline{l} \vee B) \vdash_{MaxRes} A \vee B \vee \overline{C}$

(c) $(l \vee A \vee \overline{C}) \wedge (\overline{l} \vee B \vee \overline{C}) \vdash_{MaxRes} A \vee B \vee \overline{C}$

Preuve. *On donne la preuve du motif (a) par récurrence sur $|C| = n$:*

- *Si $n = 1$, alors $C = \{l'\}$. Clairement, $(A \vee l') \wedge (B \vee \overline{l'}) \vdash_{MaxRes} A \vee B$ par application d'une seule étape de max-résolution sur la variable $var(l')$.*
- *Supposons que $n > 1$ et soit $l' \in C$. Par récurrence, on a $(A \vee C) \wedge (B \vee \overline{C \setminus \{l'\}}) \vdash_{MaxRes} A \vee B \vee l'$*

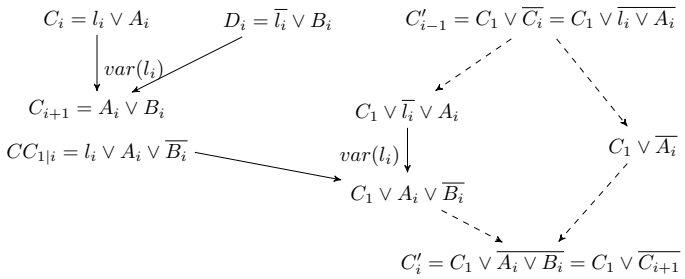


Figure 2: Étape d'induction pour déduire C'_i à la $i^{ème}$ étape. Les arcs continus représentent l'application de la max-résolution tandis que les arcs discontinus représentent le compactage ou l'expansion. Les clauses de compensation non utilisées sont omises.

en $n-1$ étapes d'inférence. De plus, on a $B \vee \overline{C} = (B \vee \overline{C \setminus \{l'\}}) \wedge (B \vee \overline{l'})$ par expansion et $(A \vee B \vee l') \wedge (B \vee \overline{l'}) \vdash_{MaxRes} A \vee B$ par application d'une étape de max-résolution sur $var(l')$. Par conséquent, on conclut que l'inférence $(A \vee C) \wedge (B \vee \overline{C})\} \vdash_{MaxRes} A \vee B$ est possible en $O(n)$ étapes.

Les preuves pour les motifs (b) et (c) sont similaires par récurrence sur $|C|$

Dans la suite, on commence à traiter l'adaptation des preuve par résolution sans croisement et en particulier les sous-dérivations des clauses non *read-once*. Pour générer un remplaçant à une clause non *read-once*, notons qu'on peut utiliser les littéraux dans les nœuds de jonction (c.f. lemme 1 dans [23]) de la sous-dérivation, c'est-à-dire les nœuds où les chemins partant de la clause non *read-once* s'intersectent. Pour générer de tels substituts en utilisant la max-résolution, nous commençons par traiter les parties linéaires *read-once* dans la preuve. De manière informelle, on veut emmener chaque clause non *read-once* quand on déplie la preuve jusqu'à ce qu'elles soient réutilisées. Ceci est formellement établi pour les parties linéaires *read-once* de la preuve dans le lemme suivant. Notons que les implications de l'égalité $\stackrel{*}{=}$ dans la preuve seront discutées plus en détail à la fin de la section.

Lemme 1. *Soit ϕ une formule FNC, $\pi = C_1, ..., C_{s(\pi)}$ une preuve par résolution linéaire* read-once *de la clause $C \neq \square$ à partir de ϕ. On peut déduire $\phi \vdash_{MaxRes} C \wedge (C_1 \vee \overline{C})$ en $O(s(\pi) \times w(\pi))$ étapes d'inférence.*

Preuve. *Soit $m = s(\pi)$. Étant donné que π est* read-once*, elle peut être trivialement adaptée en une dérivation par max-résolution de C à partir de ϕ qui est de même taille, en remplaçant chaque étape de résolution par une max-résolution [7, 12]. Dans la suite, on prouve par induction sur $i \in \{1, .., m-1\}$ qu'on peut inférer $C'_i = C_1 \vee \overline{C_{i+1}}$ à la $i^{ème}$ étape de max-résolution :*

- *Pour $i = 1$, la première max-résolution sur les clauses $C_1 = l_1 \vee A_1$ et $D_1 = \overline{l_1} \vee B_1$ et la variable $var(l_1)$ génère la clause de compensation suivante :*

$$CC_{1|1} = l_1 \vee A_1 \vee \overline{B_1} \stackrel{*}{=} l_1 \vee A_1 \vee \overline{A_1 \vee B_1} = C_1 \vee \overline{C_2} = C'_1$$

Notons que pour établir l'égalité $\stackrel{*}{=}$, on peut ajouter les clauses tautologiques $l_1 \vee A_1 \vee B_1 \vee \overline{A_1}$ (ou alternativement $l_1 \vee A_1 \vee \overline{A_1}$) à la formule), auquel cas $l_1 \vee A_1 \vee \overline{A_1 \vee B_1}$ peut être trivialement déduite par compactage. De plus, si D_1 est une clause unitaire, $CC_{1|1}$ n'est pas générée. Cependant, on peut simplement ajouter les clauses tautologiques $l_1 \vee A_1 \vee \overline{A_1}$ qui correspondent à $C_1 \vee \overline{C_2}$ puisque $D_1 = \overline{l_1}$ (B_1 est vide).

- *Supposons qu'on peut générer $C'_{i-1} = C_1 \vee \overline{C_i}$ à la $i^{ème} - 1$ étape de max-résolution. La $i^{ème}$ étape sur $C_i = l_i \vee A_i$ et $D_i = \overline{l_i} \vee B_i$ et la variable $var(l_i)$ génère la résolvante $C_{i+1} = A_i \vee B_i$ et les clauses de compensation $CC_{1|i} = l_i \vee A_i \vee \overline{B_i}$ et $CC_{2|i} = \overline{l_i} \vee \overline{A_i}$ pour B. L'induction pour inférer C'_i est représentée dans la figure 2. Notons que, comme dans le cas de base, si $D_i = \overline{l_i} (i > 1)$ est une clause unitaire, c'est-à-dire que la $i^{ème}$ étape correspond à une suppression du littéral l_i de $C_i = l_i \vee A_i$ en déduisant la résolvante $C_{i+1} = A_i$, les clauses tautologiques $l_i \vee A_i \vee \overline{A_i}$ peuvent être ajoutées à la formule remplaçant ainsi $CC_{1|i}$ dans la figure 2. Cependant, comme le montre la même figure, l'ajout de telles clauses dans le cas où D_1 est unitaire peut être évité puisque l'étape d'expansion initiale sur C'_{i-1} suffit pour générer $C_1 \vee \overline{A_i} = C_1 \vee \overline{C_{i+1}} = C'_i$.*

Enfin, d'après la proposition 2 (motif b), l'inférence de $C_1 \vee A_i \vee \overline{B_i}$ dans la figure 2 requiert $O(|B_i|)$ étapes de max-résolution et, par conséquent, chaque étape de π est clairement adaptée en $O(w(\pi))$ étapes d'inférence pour générer C et $C_1 \vee \overline{C_m}$. On conclut qu'on peut déduire $\phi \vdash_{MaxRes} C \wedge (C_1 \vee \overline{C_m})$ en $O(s(\pi).w(\pi))$ étapes d'inférence.

Exemple 3. *On considère la preuve linéaire read-once de la clause $\overline{x_3} \vee x_6$ de $\phi = \{x_4,\ x_2 \vee \overline{x_4} \vee x_6,\ \overline{x_2} \vee \overline{x_3}\}$ représentée à gauche de la figure 3. La preuve par max-résolution déduisant $\overline{x_3} \vee x_6$ et $x_4 \vee \overline{x_3} \vee x_6$ est représentée à droite de la figure 3.*

Dans la suite, nous établissons notre résultat sur l'adaptation des preuve par résolution sans croisement. La preuve dans le théorème suivant traite en particulier les

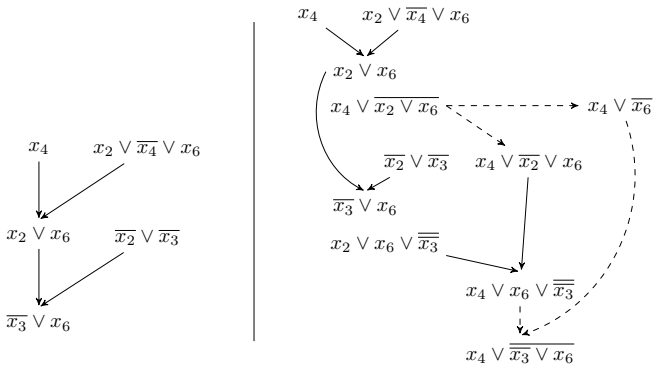


Figure 3: Maintien de la clause non *read-once* tout en dépliant une section linéaire *read-once* de la preuve. Les arcs continus représentent l'application de la max-résolution tandis que les arcs discontinus représentent le compactage ou l'expansion. Les clauses de compensation non utilisées sont omises.

nœuds de jonction dans les sous-dérivations des clauses non *read-once*, c'est-à-dire les nœuds où les chemins partant des clauses non *read-once* se croisent. Plus précisément, l'idée est de ramener la clause non *read-once* à travers ces nœuds particuliers. Une illustration d'une adaptation complète est donnée dans l'exemple 4.

Théorème 1. *Soit ϕ une formule FNC et π une preuve par résolution sans croisement de la clause C à partir de ϕ. On peut déduire $\phi \vdash_{MaxRes} C$ en $O(s(\pi) \times (s(\pi) + w(\pi))^2)$ étapes d'inférence.*

Preuve. *La propriété 1 garantit que chaque sous-dérivation de clause non read-once puisse adaptée indépendamment. Soit Cl une clause non read-once dans π et, sans perte de généralité, on considère la sous-dérivation $ED(Cl)$ d'une clause non read-once Cl. Dans la suite, on prouve qu'à chaque étape de $ED(Cl)$ C', on peut inférer C' et $SC \vee \overline{C'}$ où SC est soit Cl soit son remplaçant dans le chemin menant à C'. La preuve se fait par induction sur la taille de la dérivation. Le cas de base où la dérivation est vide est trivial. Ensuite, en utilisant le lemme 1, on peut supposer sans perte de généralité que C' est dérivée dans un nœud de jonction (de chemins partant de Cl). Soient $l \vee A$ et $\overline{l} \vee B$ les prémisses de l'étape de résolution dérivant $C' = A \vee B$. L'hypothèse d'induction garantit qu'il existe une preuve par max-résolution de $l \vee A$ et $C \vee \overline{l \vee A}$. Comme le montre la figure 4, $Cl \vee \overline{l}$ peut être utilisée pour remplacer les occurrences de Cl dans la dérivation de $\overline{l} \vee B$. Notons que pour éviter d'utiliser des remplaçants tautologiques, on peut supposer sans perte de généralité que $l \notin Cl$ en interchangeant les preuves de $l \vee A$ et $l \vee B$ lorsque cela est nécessaire imposant ainsi un ordre de déroulement de la preuve et entraînant la génération de la même clause comme substitut dans de tels nœuds. De manière similaire au côté gauche, l'hypothèse d'induction garantit aussi l'existence d'une preuve par max-résolution de $\overline{l} \vee B$ et $Cl \vee \overline{l} \vee \overline{\overline{l} \vee B}$ et, par conséquent, $Cl \vee \overline{l} \vee \overline{B}$ par expansion. Clairement, $C' = A \vee B$ peut être dérivée par max-résolution et on montre dans la figure 4 comment $Cl \vee \overline{C'} = C \vee \overline{A \vee B}$ peut être déduite en utilisant les clauses de compensation ainsi que $Cl \vee \overline{l \vee A}$ et $Cl \vee \overline{l} \vee \overline{B}$. Notons que les cas particuliers suivants peuvent se produire :*

- *A ou B est vide, auquel cas une clause unitaire est utilisée pour dériver $C' = A \vee B$. Nous représentons dans la figure 5 comment dériver $Cl \vee \overline{C'}$ dans le cas où A est vide. La dérivation dans le cas où B est vide*

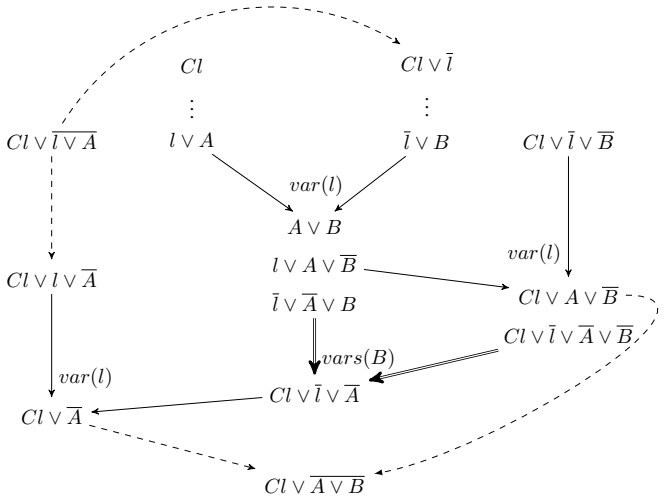


Figure 4: Inférence de $Cl \vee \overline{A \vee B}$ dans un nœud de jonction de $ED(Cl)$. Les arcs continus représentent l'application de la max-résolution, les doubles arcs continus en gras représentent l'application de la max-résolution pour supprimer des ensembles opposés de littéraux tandis que les arcs discontinus représentent le compactage ou l'expansion. Les clauses de compensation non utilisées sont omises.

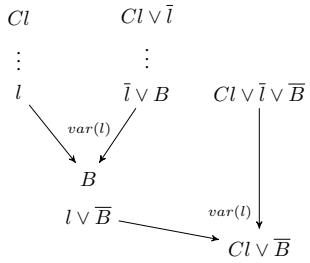


Figure 5: Inférence de $Cl \vee \overline{C'}$ dans le cas où A est vide dans un nœud de jonction de $ED(Cl)$. Les arcs continus représentent l'application de la max-résolution tandis que les arcs discontinus représentent le compactage ou l'expansion. Les clauses de compensation non utilisées sont omises.

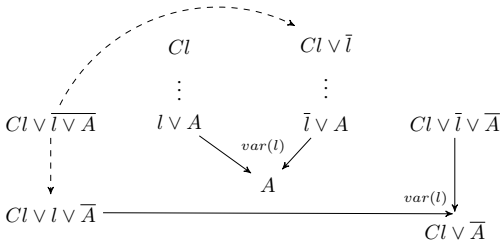


Figure 6: Inférence de $Cl \vee \overline{C'}$ dans le cas où $A = B$ dans un nœud de jonction de $ED(Cl)$. Les arcs continus représentent l'application de la max-résolution tandis que les arcs discontinus représentent le compactage ou l'expansion. Les clauses de compensation non utilisées sont omises.

est symétrique et donc omise. Notons que dans le cas où A et B sont vides, π est une réfutation et il n'est pas nécessaire de dériver $Cl \vee \overline{C'}$ dans la dernière étape de max-résolution. En effet, plus généralement, cela n'est pas non plus nécessaire pour le dernier nœud de jonction dans une sous-dérivation d'une clause non read-once de π.

- *$A = B$, auquel cas les clauses de compensation générées sont tautologiques et ne sont pas nécessaires pour dériver $Cl \vee \overline{C'} = Cl \vee \overline{A}$ comme le montre la figure 6.*

Enfin, $O(|B|)$ étapes d'inférence sont nécessaires pour déduire $Cl \vee A \vee \overline{B}$ dans chaque nœud de jonction en se référant au motif (c) dans la proposition 2. De même, en utilisant l'expansion sur A et le motif (b) dans la proposition 2, il suffit de $O(|A| \times |B|)$ étapes d'inférence pour déduire $Cl \vee \overline{l}$ lor$\overline{A}$. Il est important de noter que la largeur de la preuve peut évoluer tout en générant des remplaçants pour les clauses non read-once car des littéraux peuvent être ajoutés dans les nœuds de jonction. Cependant, la largeur reste bornée par $w(\pi)+s(\pi)$ et donc chaque nœud de jonction peut être adapté en $O((w(\pi) + s(\pi))^2)$ étapes d'inférences. Par conséquent, on conclut qu'on peut déduire $\phi \vdash_{MaxRes} C$ en $O(s(\pi) \times (s(\pi) + w(\pi))^2)$ étapes d'inférence.

Exemple 4. *On considère la formule $\phi = \{\overline{x_1} \vee x_3 \vee \overline{x_4},\ x_4 \vee x_5,\ x_4 \vee \overline{x_5},\ x_1 \vee \overline{x_4},\ x_2 \vee \overline{x_4} \vee x_6,\ \overline{x_2} \vee \overline{x_3}\}$ et la preuve π de la clause x_6 à partir de ϕ représentée dans la figure 1. Pour simplifier, on omet la section (en bleu) déduisant la clause $\overline{x_2} \vee \overline{x_3}$. Notons que cette section de la preuve, c'est à dire la sous-dérivation de la clause non read-once $\overline{x_2} \vee \overline{x_3} \vee \overline{x_7}$, correspond à un motif diamant [23]. Ces motifs seront étudiés dans la section 5. L'adaptation de la preuve π est reportée dans la Figure 7. Nous réutilisons l'adaptation de la section*

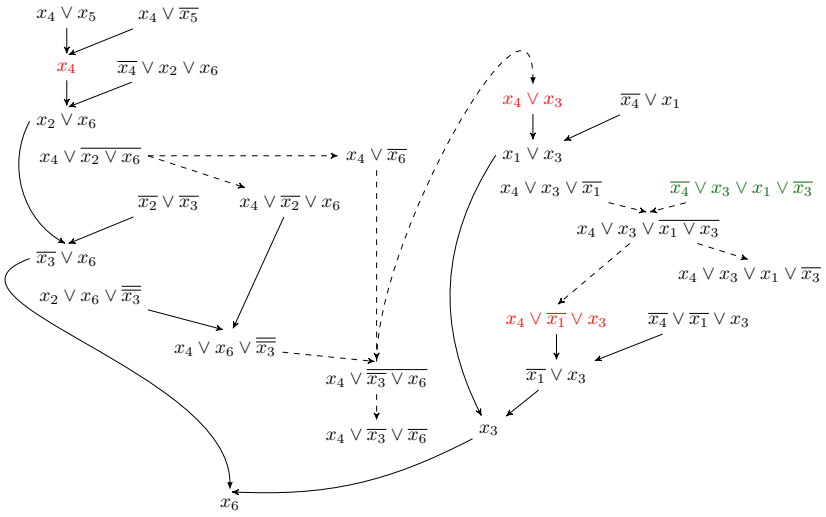


Figure 7: Adaptation d'une preuve par résolution sans croisement. Les arcs continus représentent l'application de la max-résolution tandis que les arcs discontinus représentent le compactage ou l'expansion. Les clauses de compensation non utilisées sont omises.

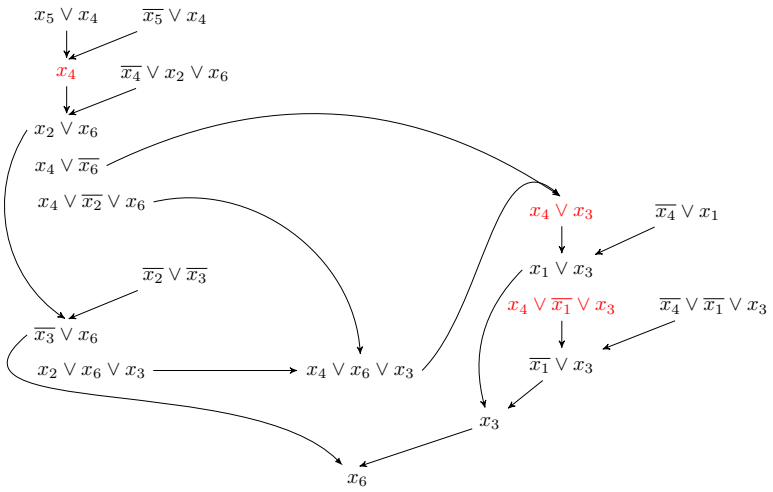


Figure 8: Adaptation d'une preuve par résolution en une preuve par max-résolution (exclusivement). Les clauses de compensation non utilisées sont omises.

linéaire read-once dans l'exemple 3. La clause non read-once et ses remplaçants sont colorés en rouge et les clauses tautologiques ajoutées sont représentées en vert. Notons qu'il s'agit d'une des adaptations possibles selon l'ordre choisi pour adapter les branches de $ED(x_4)$. Enfin, nous soulignons le fait que nous aurions pu générer la clause $C = x_4 \vee \overline{x_6}$ après la dernière étape de max-résolution sur les clauses $\overline{x_3} \vee x_3$ (mais cette inférence n'est pas nécessaire puisque $x_6 = EC(x_4)$ comme expliqué dans la preuve du théorème 1). En effet, C peut être déduite par une étape supplémentaire de max-résolution sur les clauses de compensation obtenues à la dernière étape, c'est-à-dire les clauses $x_3 \vee \overline{x_6}$ et $x_4 \vee \overline{x_3} \vee \overline{x_6}$. Dans la preuve du théorème 1, cela correspond au cas où B est vide dans un nœud de jonction d'une sous-dérivation.

Dans la suite, nous discutons de quelques subtilités syntaxiques mineures qui se produisent dans l'adaptation. Tout d'abord, il est important de noter que l'utilisation des réécritures d'expansion et de compactage en tant que règles à part entière est pertinente pour la simplification mais pas nécessaire. Rappelons que ces deux règles sont principalement utilisées pour alterner entre les différentes formes équivalentes de $\overline{C}$ lorsqu'elle est écrit sous forme FNC. Chaque forme correspond à un ordre différent des littéraux de C. Lors de l'application de la max-résolution, un ordre pertinent peut être choisi si nécessaire. Cependant, une application d'un compactage suivi d'une expansion peut correspondre à un certain réarrangement des variables sous forme FNC. Cela peut se produire lors de l'adaptation de la partie linéaire read-once de la preuve. En effet, comme le montre la figure 2, un compactage peut être suivi d'une expansion pour isoler la clause $C_1 \vee \overline{l_i} \vee A_i$ de la forme compacte $C_1 \vee \overline{A_i}$. De même, comme le montre la figure 4, il peut être nécessaire d'isoler la clause $Cl \vee \bar{l}$ de la forme compacte $Cl \vee l \vee \overline{A}$ dans les nœuds de jonction. Plus précisément, il pourrait être nécessaire de réorganiser un certain littéral au début ou à la fin de l'ordre. Pour cela, nous prouvons dans la proposition 3 qu'il est possible d'alterner le premier et le dernier littéral dans la forme FNC de $\overline{C}$ en $O(|C|)$ étapes d'inférence. cela implique que, dans la preuve du théorème 1, les règles de compactage et d'expansion peuvent être omises et remplacées par $O(s(\pi) \times (s(\pi) + w(\pi)))$ max-résolutions. Cela n'a évidemment pas d'impact sur notre résultat en termes de taille de l'adaptation résultante. Dans l'exemple 5, nous fournissons une adaptation entièrement simplifiée de la preuve de l'exemple 4 sans l'utilisation de règles de réécriture.

Proposition 3. *Soit n un entier naturel et $l_1, ..., l_n$ n littéraux. On peut déduire $(\overline{l_1}) \wedge (l_1 \vee \overline{l_2}) \wedge ... \wedge (l_1 \vee ... \vee l_{n-1} \vee \overline{l_n}) \vdash_{MaxRes} (\overline{l_n}) \wedge (l_n \vee \overline{l_2}) \wedge ... \wedge (l_n \vee l_2 \vee ... \vee l_{n-1} \vee \overline{l_1})$ en $O(n)$ étapes d'inférence.*

Preuve. *By induction on n we have:*

- *Pour $n = 1$, le résultat est trivial.*
- *Pour $n > 1$, l'application de la max-résolution sur les clauses $l_1 \vee .... \vee l_{n-2} \vee \overline{l_{n-1}}$ et $l_1 \vee .... \vee l_{n-1} \vee \overline{l_n}$ w.r.t $var(l_{n-1})$ génère la clause résolvante $C = l_1 \vee .... lorl_{n-2} \vee \overline{l_n}$ et la clause de compensation $CC = l_1 \vee .... \vee l_{n-2} \vee l_n \vee \overline{l_{n-1}}$. De plus, par récurrence, on peut déduire $(\overline{l_1}) \wedge (l_1 \vee \overline{l_2}) \wedge ... \wedge (l_1 \vee ... \vee l_{n-2} \vee \overline{l_n}) \vdash_{MaxRes} (\overline{l_n}) \wedge (l_n \vee \overline{l_2}) \wedge ... \wedge (l_n \vee l_2 \vee ... lorl_{n-2} \vee \overline{l_1})$ en $O(n-1)$ étapes d'inférence. Une seule étape de max-résolution supplémentaire sur les clauses CC et $l_n \vee l_2 \vee ... \vee l_{n-2} \vee \overline{l_1}$ w.r.t $var(l_1)$ est suffisante pour générer la clause résolvante $l_n \vee l_2 \vee ... \vee l_{n-2} \vee \overline{l_{n-1}}$ et la clause de compensation $l_n \vee l_2 \vee ... oul_{n-1} \vee \overline{l_1}$. Par conséquent, nous déduisons le résultat voulu en $O(n)$ étapes d'inférence.*

Exemple 5. *On considère la même formule ϕ dans l'exemple 4. Nous représentons sur la figure 8 une preuve par max-résolution (sans réécriture) de la clause x_6 à partir de ϕ. Notons que le réarrangement suivant $\overline{x_6} \wedge (x_6 \vee$*

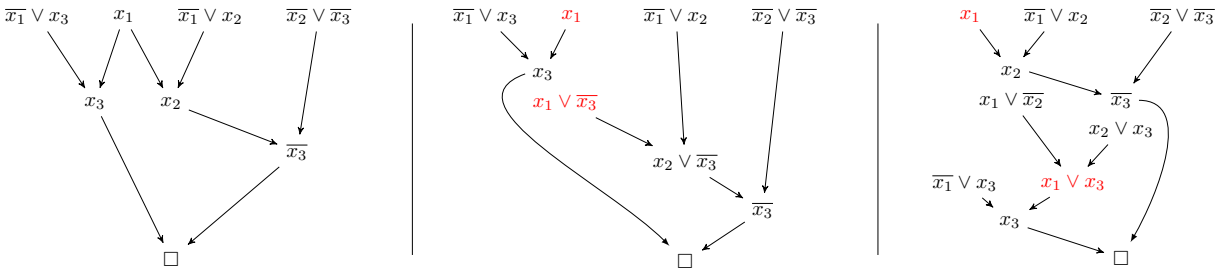


Figure 9: Deux adaptations possibles de la réfutation par résolution sans croisement représentée à gauche en fonction de l'ordre des étapes de résolution impliquant la clause non read-once x_1. Les clauses de compensation non utilisées sont omises.

$x_3) \vdash_{MaxRes} x_3 \wedge (x_3 \vee x_6)$ *a été utilisé pour générer le replaçant* $x_4 \vee x_3$. *De plus, la clause tautologique* $\overline{x_4} \vee x_3 \vee x_1 \vee \overline{x_3}$ *colorée en vert dans la figure 7 et le réarrangement dans lequel elle est impliquée ne sont pas nécessaires puisque le dernier remplaçant requis pour* x_1, *c'est-à-dire* $x_4 \vee \overline{x_1} \vee x_3$, *est naturellement généré par l'étape de max-résolution précédente. Par conséquent, ils peuvent être supprimés comme c'est le cas pour l'adaptation complète sans réécriture dans la figure 8.*

Dans la suite, nous discutons des implications de l'égalité $\overset{*}{=}$ utilisée dans la preuve du lemme 1, c'est-à-dire $l \vee A \vee \overline{B} \overset{*}{=} l \vee A \vee \overline{A \vee B}$. Cette transformation est valide pour Max-SAT (c.f. Remarque 13 dans [16]). Cependant, pour éviter de l'ajouter comme règle à part entière et comme expliqué dans la preuve du lemme 1, on peut envisager l'ajout de clauses tautologiques. Cela peut également être nécessaire en cas de clauses unitaires. Il est important de noter que le nombre de clauses tautologiques ajoutées à la formule dans une adaptation d'une preuve par résolution sans croisement π est en $O(s(\pi) \times (w(\pi) + s(\pi)))$. Un phénomène similaire a également été noté dans [8]. De plus, on remarque que l'adaptation peut également s'appuyer sur des clauses de compensation tautologiques qui sont générées par la max-résolution. De telles clauses sont généralement supprimées ou omises dans la littérature [6, 7, 16] mais elles peuvent contenir des informations importantes qui sont nécessaires pour déduire des substituts aux clauses non read-once.
Enfin, on établit notre résultat sur les réfutations sans croisement dans le corollaire suivant. Nous illustrons également dans l'exemple 6 une adaptation d'une réfutation par résolution sans de croisement à une réfutation par max-résolution.

Corollaire 1. *Soit* ϕ *une formule FNC insatisfiable et* π *une réfutation par résolution sans croisement de* ϕ. *On peut déduire* $\phi \vdash_{MaxRes} \square$ *de* ϕ *en* $O(s(\pi)^3)$ *étapes d'inférence.*

Preuve. *Résultat trivialement déduit du théorème 1 puisque* $w(\pi) = O(s(\pi))$ *pour les réfutations.*

Exemple 6. *On considère la formule FNC insatisfiable* $\phi = \{x_1,\ \overline{x_1} \vee x_3,\ \overline{x_1} \vee x_2,\ \overline{x_2} \vee \overline{x_3}\}$ *et la réfutation* π *de* ϕ *représentée à gauche dans la Figure 9. Clairement,* π *est sans croisement car elle ne contient qu'une seule clause non read-once* x_1. π *correspond également à la sous-dérivation de cette clause, c'est-à-dire* $ED(x_1) = \pi$ *avec* $EC(x_1) = \square$. *Deux adaptations possibles de* π *sont illustrées dans la figure 9. La clause non read-once et ses remplaçants sont colorés en rouge. Les adaptations possibles correspondent à différents ordre possibles de la preuve. Dans la première adaptation, on considère que l'étape de résolution sur les clauses* $\overline{x_1} \vee x_1$ *et* x_1 *précède celle sur les clauses* x_1 *et* $\overline{x_1} \vee x_2$, *et inversement pour l'adaptation à droite. Notons que la première adaptation correspond à l'exemple fourni par Bonet et al. dans [6, 7] (c.f. Exemple 1 dans [6] ou Exemple 3 dans [7]).*

5 Motifs diamants

Dans cette section, nous étudions des réfutations de résolution particulières, appelées motifs diamants k-empilés, qui ont été introduites et montrées exponentielle pour l'adaptation (à des réfutations ResS) dans [23]. Un motif diamant (x, y, A) où $x, y \notin A$ est la séquence de résolutions représentée sur la figure 10. Notons que le motif diamant particulier $(x, y, \square)$ est une réfutation par résolution. Maintenant, imaginons que la clause x de $(x, y, \square)$ est dérivée d'un autre motif diamant. Nous itérons le même raisonnement pour définir les motifs diamants k-empilés comme dans la définition 6.

Définition 6 (Motif diamants k-empilés)**.** *Soit* $k \geq 1$ *un entier naturel et soit* x_i *et* y_i *où* $1 \leq i \leq k$ *des variables distinctes. Un motif diamants k-empilé est formé de* k *diamants* (x_i, y_i, A_i) *où* $1 \leq i \leq k$ *tels que* $A_1 = \square$ *et* $A_i = (x_1 \vee \ldots \vee x_{i-1})$ *pour* $1 < i \leq k$. *Chaque diamant* (x_i, y_i, A_i) *est empilé sur* $(x_{i-1}, y_{i-1}, A_{i-1})$ *de sorte que la dernière conclusion du premier est la prémisse centrale de ce dernier.*

Lorsque $k > 2$, la taille d'un motif diamants k-empilés P

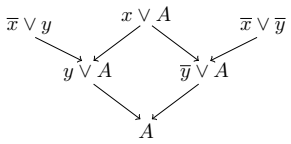


Figure 10: Motif Diamant (x, y, A)

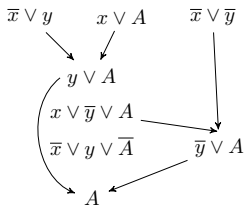


Figure 11: Adaptation d'un diamant (x, y, A)

est $s(P) = 3k$ tandis que la taille de la réfutation calculée dans ResS [17], c'est-à-dire la max-résolution augmentée avec la règle *split*, par l'adaptation dans [23] est au moins 2^{k-1} ce qui est exponentiel en la taille de P. Dans la figure 11, nous montrons comment adapter chaque motif diamant sans augmenter sa taille. Ceci est impliqué par le fait que chaque diamant est clairement une preuve par résolution sans croisement et plus spécifiquement une sous-dérivation d'une clause non *read-once*. Ainsi, un diamant et plus généralement un motif diamants k-empilés P peut être adapté en au plus $s(P)$ étapes de max-résolution.

6 Conclusion

Dans ce papier, nous avons introduit un nouveau fragment de résolution, appelé résolution sans croisement, dans lequel les sous-dérivations des clauses non read-once sont disjointes. Nous avons montré que les preuve par résolution sans croisement et en particulier les réfutations sans croisement peuvent être adaptées en preuves par max-résolution sans augmenter considérablement leur taille. À notre connaissance, il s'agit du premier fragment non trivial, c'est-à-dire différent de la résolution read-once, dont l'adaptation est montrée possible en utilisant uniquement la max-résolution avec une garantie raisonnable sur la taille des preuves adaptées. L'idée derrière l'adaptation est d'inférer naturellement des remplaçants aux clauses non read-once en les ramenant tout en déroulant la preuve par résolution initiale jusqu'à leur réutilisation et en s'appuyant sur les clauses de compensation produites par la max-résolution. De plus, nous montrons que les motifs diamants (k-empilés), qui ont été montrés exponentiels pour l'adaptation dans [23], appartiennent au ce nouveau fragment et peuvent être adaptés en preuves par max-résolution sans augmenter leur taille.

Nos résultats contribuent au problème ouvert d'adapter les preuves par résolution en preuves par max-résolution sans augmenter considérablement leur taille [6, 7] et, par conséquent, aident à combler l'écart entre la résolution pour SAT et Max-SAT. Enfin, contrairement aux solveurs SAT, les solveurs Max-SAT ne sont toujours pas en mesure de produire des certificats pour Max-SAT, principalement en raison de la variété des paradigmes de résolution et du manque de caractérisation d'une intersection plus large entre la résolution SAT et Max-SAT. Notre travail peut être utile dans ce contexte et en particulier pour améliorer l'efficacité des constructeurs de preuves indépendants pour le problème Max-SAT [24].

References

[1] André Abramé and Djamal Habet. Ahmaxsat: Description and Evaluation of a Branch and Bound Max-SAT Solver. *JSAT*, 9(1):89–128, 2014.

[2] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. SAT-based MaxSAT algorithms. *Artif. Intell.*, 196:77–105, 2013.

[3] F. Bacchus, J Berg, M. Järvisalo, and R. Martins, editors. *MaxSAT Evaluation 2021: Solver and Benchmark Descriptions*. University of Helsinki, Department of Computer Science, Finland, 2021.

[4] Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. *Maximum Satisfiability*, pages 929 – 991. FAIA. IOS PRESS, Netherlands, 2 edition, 2021.

[5] Maria Luisa Bonet and Jordi Levy. Equivalence Between Systems Stronger Than Resolution. In *SAT*, 2020.

[6] Maria Luisa Bonet, Jordi Levy, and Felip Manyà. A Complete Calculus for Max-SAT. In Armin Biere and Carla P. Gomes, editors, *SAT*, pages 240–251, 2006.

[7] Maria Luisa Bonet, Jordi Levy, and Felip Manyà. Resolution for Max-SAT. *Artif. Intell.*, 171(8-9):606–618, 2007.

[8] Joshua Buresh-Oppenheim and Toniann Pitassi. The complexity of resolution refinements. *J. Symb. Log.*, 72(4):1336–1352, 2007.

[9] Sam Buss and Jan Johannsen. On Linear Resolution. *JSAT*, 10(1):23–35, 2016.

[10] Mohamed Sami Cherif, Djamal Habet, and André Abramé. Understanding the power of Max-SAT resolution through UP-resilience. *Artif. Intell.*, 289:103397, 2020.

[11] Yuval Filmus, Meena Mahajan, Gaurav Sood, and Marc Vinyals. MaxSAT Resolution and Subcube Sums. In *SAT*, volume 12178, pages 295–311. Springer, 2020.

[12] Federico Heras and João Marques-Silva. Read-Once Resolution for Unsatisfiability-Based Max-SAT Algorithms. In *IJCAI*, pages 572–577, 2011.

[13] Kazuo Iwama and Eiji Miyano. Intractability of Read-Once Resolution. In *Proceedings of the Tenth Annual Structure in Complexity Theory Conference, Minneapolis, Minnesota, USA, June 19-22, 1995*, pages 29–36, 1995.

[14] Adrian Kügel. Improved Exact Solver for the Weighted MAX-SAT Problem. In *POS-10. Pragmatics of SAT*, volume 8 of *EPiC Series in Computing*, pages 15–27, 2010.

[15] Sukhamay Kundu. Tree resolution and generalized semantic tree. In *Proceedings of the ACM SIGART international symposium on Methodologies for intelligent systems*, pages 270–278, 1986.

[16] Javier Larrosa, Federico Heras, and Simon de Givry. A logical approach to efficient Max-SAT solving. *Artif. Intell.*, 172(2-3):204–233, 2008.

[17] Javier Larrosa and Emma Rollon. Towards a Better Understanding of (Partial Weighted) MaxSAT Proof Systems. In *SAT*, volume 12178, pages 218–232, 2020.

[18] Chu Min Li, Felip Manyà, and Jordi Planes. New Inference Rules for Max-SAT. *JAIR*, 30:321–359, 2007.

[19] Chu Min Li, Felip Manyà, and Joan Ramon Soler. A Clause Tableau Calculus for MaxSAT. In *IJCAI*, pages 766–772, 2016.

[20] Chu-Min Li, Zhenxing Xu, Jordi Coll, Felip Manyà, Djamal Habet, and Kun He. Combining Clause Learning and Branch and Bound for MaxSAT. In *CP*, pages 38:1–38:18, 2021.

[21] Donald W Loveland. A linear format for resolution. *Symposium on Automatic Demonstration*, pages 147–162, 01 1970.

[22] Nina Narodytska and Fahiem Bacchus. Maximum Satisfiability Using Core-Guided MaxSAT Resolution. In *AAAI*, pages 2717–2723, 2014.

[23] Matthieu Py, Mohamed Sami Cherif, and Djamal Habet. Towards Bridging the Gap Between SAT and Max-SAT Refutations. In *ICTAI*, pages 137–144. IEEE, 2020.

[24] Matthieu Py, Mohamed Sami Cherif, and Djamal Habet. A Proof Builder for Max-SAT. In Chu-Min Li and Felip Manyà, editors, *SAT*, volume 12831 of *Lecture Notes in Computer Science*, pages 488–498. Springer, 2021.

[25] Matthieu Py, Mohamed Sami Cherif, and Djamal Habet. Computing Max-SAT Refutations using SAT Oracles. In *ICTAI*, pages 404–411. IEEE, 2021.

[26] Matthieu Py, Mohamed Sami Cherif, and Djamal Habet. Inferring Clauses and Formulas in Max-SAT. In *ICTAI*, pages 632–639. IEEE, 2021.

[27] John Alan Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *J. ACM*, 12(1):23–41, 1965.

Session 5 : Applications

Génération de texte sous contraintes pour mesurer des performances de lecture : Une nouvelle approche basée sur les diagrammes de décisions multivalués

A. Bonlarron[1,3], A. Calabrèse[2], P. Kornprobst[1], J.-C. Régin[3]

[1] Université Côte d'Azur, Inria, France
[2] Aix Marseille Univ, CNRS, LPC, Marseille, France
[3] Université Côte d'Azur, I3S, France

Alexandre.Bonlarron@inria.fr

Résumé

Mesurer les performances de lecture est l'une des méthodes les plus utilisées en clinique ophtalmologique pour juger de l'efficacité des traitements, des procédures chirurgicales ou des techniques de rééducation. Cependant, l'utilisation des tests de lecture est limitée par le faible nombre de textes standardisés disponibles. Pour le test MNREAD, qui est l'un des tests de référence pris comme exemple dans ce papier, il ne comporte que deux jeux de 19 phrases en français. Ces phrases sont difficiles à écrire car elles doivent respecter des règles de différentes natures (e.g., liées à la grammaire, la longueur, le lexique et l'affichage). Ils sont aussi difficile à trouver : Sur un échantillon de plus de trois millions de phrases issues d'ouvrages de la littérature jeunesse, seulement quatre satisfont les critères du test de lecture MNREAD. Pour obtenir davantage de phrases, nous proposons une approche originale de génération de texte qui prenne en compte l'ensemble des règles dès la génération. Notre approche est basée sur les Multi-valued Decision Diagrams (MDD). Nous représentons le corpus par des n-grammes et les différentes règles par des MDD, puis nous les combinons à l'aide d'opérateurs, notamment des intersections. Les résultats obtenus montrent que cette approche est prometteuse, même si certains problèmes demeurent comme la consommation mémoire ou la validation a posteriori du sens des phrases. En 5-gramme, nous engendrons plus de 4000 phrases qui respectent les critères MNREAD et proposons ainsi facilement une extension d'un jeu de 19 phrases au test MNREAD.

Mots-clés

Génération de texte, texte standardisé, tests de lecture, test MNREAD, basse vision, diagrammes de décisions multivalués, contraintes.

Abstract

Measuring reading performance is one of the most widely used methods in ophthalmology clinics to judge the effectiveness of treatments, surgical procedures, or rehabilitation techniques. However, reading tests are limited by the small number of standardized texts available. For the MNREAD test, which is one of the reference tests used as an example in this paper, there are only two sets of 19 sentences in French. These sentences are challenging to write because they have to respect rules of different kinds (e.g., related to grammar, length, lexicon, and display). They are also tricky to find : out of a sample of more than three million sentences from children's literature, only four satisfy the criteria of the MNREAD reading test. To obtain more sentences, we propose an original approach to text generation that considers all the rules at the generation stage. Our approach is based on Multi-valued Decision Diagrams (MDD). First, we represent the corpus by n-grams and the different rules by MDDs, and then we combine them using operators, notably intersections. The results obtained show that this approach is promising, even if some problems remain, such as memory consumption or a posteriori validation of the meaning of sentences. In 5-gram, we generate more than 4000 sentences that meet the MNREAD criteria and thus easily provide an extension of a 19-sentence set to the MNREAD test.

Keywords

Text generation, standardized text, reading tests, MNREAD test, low vision, multivalued decision diagrams, constraints.

1 Introduction

Les performances de lecture sont devenues l'une des mesures cliniques les plus utilisées pour juger de l'efficacité des traitements, des procédures chirurgicales ou des techniques de rééducation [22, 9]. Ces performances de lecture sont mesurées à partir du temps nécessaire pour lire des textes standardisés, i.e., conçus pour être équivalents en termes de longueur, d'affichage et de linguistique. Ce besoin d'avoir des textes équivalents est la clé pour éviter tout biais lié aux textes eux-mêmes. D'autre part chaque texte ne doit être présenté qu'une seule fois aux sujets ou patients, ceci pour éviter d'introduire un biais de mémorisation et assurer une mesure précise.

Parmi les différents tests existants [19], le test MNREAD [11, 12] est probablement l'un des tests de lecture standardisés les plus utilisés au monde pour mesurer les performances de lecture dans des contextes cliniques mais aussi de recherche, chez les personnes ayant une vision normale ou faible (basse vision [3]). Un test MNREAD est composé de phrases standardisées imprimées en 19 tailles par pas de 0,1 logMAR (une unité angulaire qui permet de quantifier l'acuité visuelle). L'objectif est d'évaluer comment la performance de lecture dépend de la taille de la police [4]. Il est disponible en 19 langues, avec un nombre de tests variable en fonction de la langue (e.g., cinq anglais, deux en français). Étant donné que des mesures répétées sont nécessaires dans de nombreuses applications de MNREAD, les communautés scientifiques et médicales ont besoin d'un grand nombre de jeux de tests, notamment pour éviter les biais de mémorisation. Ceci est impossible aujourd'hui car le nombre de phrases MNREAD est largement insuffisant.

Cependant, accroître le nombre de phrases MNREAD est un problème difficile à cause des règles multiples que les phrases doivent respecter. Ecrire ces phrases ou les trouver dans des corpus n'est clairement pas la solution car elles sont très spécifiques. Pour résoudre ce problème, qui est un problème dominé par des règles, une première idée serait d'utiliser des méthodes d'apprentissage profond (e.g., GPT, Bert) qui sont en plein essor [6]. Cependant, elles pourraient s'avérer difficile à mettre en oeuvre dans notre cas pour les deux raisons suivantes : (i) elles nécessitent de larges bases d'apprentissage, mais nous ne disposons que d'un faible nombre de phrases MNREAD d'origine, et (ii) à notre connaissance, l'introduction de règles (contraintes) dans les réseaux profonds pose des limitations, mais c'est un sujet qui évolue vite actuellement [14, 20].

Une autre idée est de concevoir des approches plus ad-hoc dédiées aux tests de lectures [5, 15, 13]. En particulier, dans [13], les auteurs, qui sont aussi les créateurs du test MNREAD, proposent une approche basée sur des modèles de phrases (*templates*) et capable de générer des millions de phrases en procédant à une marche aléatoire. Cependant, cette méthode semi-automatique présente plusieurs inconvénients majeurs : (i) elle repose sur des modèles de phrases qui doivent être créés manuellement (i.e., des séquences d'espaces réservés, chacun contenant une liste de mots possibles qui s'intègrent dans la phrase à ce moment-là) ; (ii) elle fonctionne en deux étapes (i.e., la création de la phrase suivie de la sélection de la phrase) impliquant des calculs supplémentaires et un temps d'exécution plus long ; (iii) elle ne peut pas être étendue facilement à d'autres langues, comme par exemple pour le français où l'on doit tenir compte des accords et conjugaisons.

Pour remédier à ces limitations, nous proposons dans cet article de modéliser le problème de génération de textes standardisés comme un problème d'optimisation sous contraintes. L'objectif est d'avoir une approche plus automatique (sans besoin de définir de modèle de phrase *a priori*), qui produisent uniquement des phrases respectant les règles (sans avoir à les vérifier *a posteriori*), et enfin qui puissent être généralisable (i.e., avec d'autres règles ou dans d'autres langues). Pour cela, nous proposons une approche originale basée sur les diagrammes de décision multivalués (en anglais *multi-valued decision diagrams*, MDD). Les MDD ont déjà été utilisés avec succès dans de nombreux autres problèmes comme la génération de musique [21] ou de poèmes [17]. D'autres utilisations des MDD pour des problèmes d'optimisations peuvent être consultés dans l'ouvrage de référence de Bergman et al. [2], ou dans la thèse plus récente de Perez [16]. Les MDD sont une généralisation des diagrammes de décision binaire (BDD) [1]. Ce sont des structures de données permettant de calculer et stocker des solutions (des n-uplets) d'un problème à l'aide d'un graphe orienté acyclique (DAG). Utiliser les MDD dans notre contexte offre trois avantages principaux : (i) Les MDD permettent de représenter un grand nombre de solutions dans une structure compressante, ce qui va être crucial étant donné la taille des données que nous allons devoir manipuler, (ii) Les MDD permettent de décomposer le problème en définissant autant de MDD que de règles que le texte doit respecter. (iii) Il est possible d'effectuer efficacement des opérations entre MDD permettant ainsi de combiner différentes règles [18].

Pour sa construction, un MDD dans sa forme ordonné est structuré en couches où chaque couche représente une variable ordonnée X_i. Il y a deux noeuds particulier dans le MDD : la racine (`root`) et le noeud true terminal (`tt`). Chaque chemin entre le noeud `root` et `tt` forme un n-uplets de label qui correspond à une affectation valide des variables associées à chaque couche. Le label l_{a_k} de l'arc a_k appartenant à un chemin quelconque entre la couche $i-1$ et i est l'affectation de la variable X_i tel que X_i = l_{a_k}. Un chemin p = (a_1, a_2, a_3,..., a_r) existe dans le MDD si et seulement si l'assignation des variables X_i=l_{a_i} est valide pour tout i. Ainsi calculer un MDD contenant r couches revient à calculer les r-uplets valides de la fonction $f : \{0...d\}^r \mapsto \{true, false\}$. Un exemple de MDD de type somme est donné dans la Fig. 1.

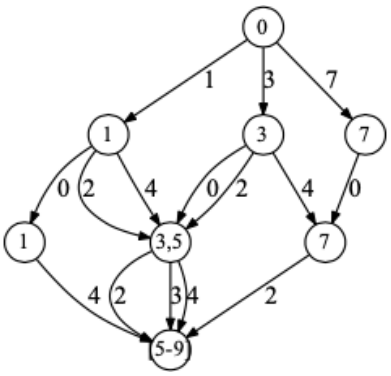


FIGURE 1 – Exemple de MDD représentant l'ensemble de $x_1 + x_2 + x_3 \in [5, 9]$. Pour chaque variable x_i, le domaine $D(.)$ est $D(x_1) = \{1, 3, 7\}$, $D(x_2) = \{0, 2, 4\}$, $D(x_3) = \{2, 3, 4\}$. Par exemple, $(7, 0, 2)$ appartient à l'ensemble des solutions appartenant au MDD.

Le plan de cet article est le suivant. Dans la section 2 nous montrons comment modéliser le problème de génération de texte sous contraintes avec des MDD. Pour l'illustrer, on se focalise dans ce papier sur la génération de phrases de type MNREAD dont on commencera par rappeler les règles MNREAD, puis on expliquera comment construire les MDD associés, et enfin comment on peu les intersecter. Les résultats sont présentés dans la section 3 dans laquelle on montre le potentiel de notre méthode. Nous terminons en section 4 par une discussion.

2 Méthode

2.1 Caractérisation des phrases MNREAD : définition des règles

Afin d'être standardisées, toutes les phrases MNREAD doivent obéir à cinq règles $(\mathcal{R}_i)_{i=0..4}$ explicitées par les créateurs du test (voir [13] pour plus de détails) :

- règles grammaticales : des phrases en forme déclarative, sans ponctuation, pas de noms propres ($\mathcal{R}_0$)
- règles lexicales : des phrases construites uniquement à partir des 3000 lemmes (forme canonique) des mots les plus fréquents dans les manuel de niveau CE2, comme défini dans le lexique Manulex [10] ($\mathcal{R}_1$)
- règles de longueur : des phrases 9 à 15 mots ($\mathcal{R}_3$) et 59 caractères avec les espaces et sans compter le point final ($\mathcal{R}_2$)
- règles d'affichage : des phrases qui peuvent s'afficher sur trois lignes, avec une justification gauche-droite et des espace inter-mots compris entre des valeurs seuils strictes, proportionnelles à la taille de l'espace standard pour la police donnée ($\mathcal{R}_4$).

Un exemple de phrase MNREAD respectant ces règles est donné dans la Fig. 2.

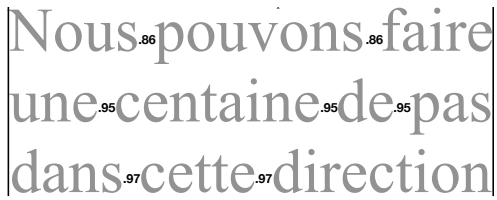


FIGURE 2 – Affichage d'une phrase MNREAD type respectant les contraintes $\mathcal{R}_3 \ldots \mathcal{R}_4$. Chaque phrase du test est affichée avec une justification à gauche et à droite sur trois lignes de texte qui tiennent dans une boîte dont la largeur est égale à 17,3 fois la taille d'une lettre standard en police Times-Roman. La largeur des espaces entre les mots (indiquée comme multiple de la largeur normale des espaces) doit être comprise entre 0,80 et 1,25 (adapté de [13]).

Les règles $(\mathcal{R}_i)_{i=0..4}$ sont nécessaires mais non suffisantes pour appartenir au test. En effet, remarquons qu'il y a également des règles implicites, plus qualitatives et plus difficile à formaliser mais qui semblent "évidentes", comme des règles sémantiques (la phrase doit avoir un sens et respecter la congruence) ou syntaxiques (la phrase doit avoir une structure "simple"). Une phrase considérée de type MNREAD sera donc une phrase qui respecte toutes les règles précédemment énoncées, qui a du sens et enfin qui ressemble aux phrases officielles du test.

2.2 Définition d'un ensemble de n-grammes

Dans l'esprit, les MDD vont décrire tous les chemins possibles entre mots pour construire des phrases. Cependant, afin d'avoir plus de chance d'obtenir des phrases dont la syntaxe est correcte, l'idée est de ne considérer que des recombinaisons de bouts de phrases existantes. Plus précisement, étant donné un corpus constitué d'un ensemble de phrases écrites par des auteurs (dont la syntaxe et le sens sont donc corrects), l'idée est de les décomposer en n-grammes (sous-séquence de n mots construite à partir d'une phrase donnée) que l'on va chercher à recombiner tout en respectant les règles MNREAD.

En pratique, comme il est fait classiquement, nous allons créer un dictionnaire de n-grammes à partir d'un corpus constitué d'un ensemble de livres. Ce qui fait la spécificité de notre problème, c'est que nous allons ignorer un certain nombre de phrases (et donc de n-grammes) afin de prendre en compte dès le début les règles grammaticales $\mathcal{R}_0$ (e.g., les phrases interrogatives vont être ignorées, ceci afin de ne pas rajouter des n-grammes correspondant à des tournures interrogatives qui ne sont pas souhaitées dans notre application). Il y a ensuite une série de transformations à l'aide d'expressions régulières qui peuvent être appliquées aux phrases du corpus pour obtenir davantage de n-grammes utilisables (e.g., les points de suspensions, deux points et point-virgules sont considérées comme des points finaux ; les guillemets sont ignorés). Enfin, une série de réécriture est appliquée aux abréviations de statuts usuels de manière à retrouver leurs formes non abrégées (e.g., M. est réécrit Monsieur).

2.3 Construction des MDD : des règles aux contraintes

Pour résoudre notre problème, il s'agit de formaliser les règles MNREAD dans le paradigme MDD. Chaque MDD va résoudre un sous-problème correspondant à une règle. Ensuite, il s'agira d'effectuer des opérations (intersections) entre ces MDD pour obtenir les solutions finales. Pour cela, tous les MDD doivent avoir la même taille, i.e., le même nombre de couches, ce qui correspond aux nombres de mots que notre solution doit comporter. Les labels des arcs seront des informations en rapport avec les mots. Cette section présente comment chaque MDD est construit. La Fig. 3 montre une représentation de ces MDD dans un cadre simplifié.

MDD_U (associé à tous les mots du corpus). Ce MDD est le MDD universel associé au corpus. Ce MDD contient toutes les séquences de mots d'une taille 9 à 15 qui peuvent être écrites à partir des mots appartenant au corpus. L'ensemble des solutions qu'il contient est égal au produit car-

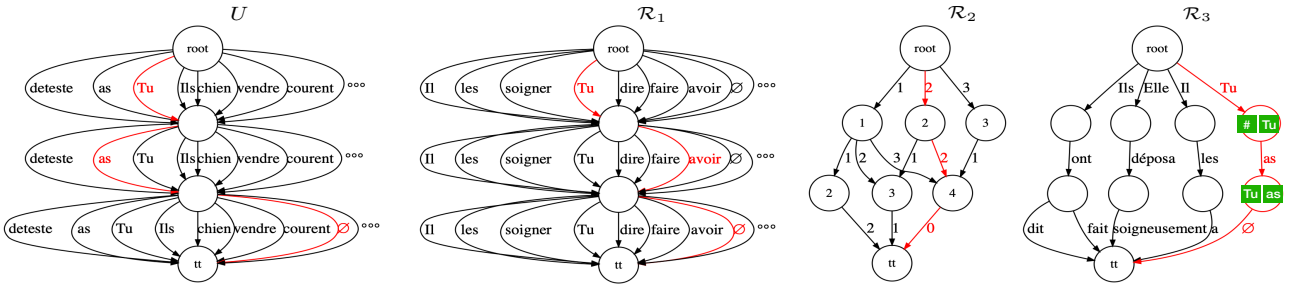


FIGURE 3 – Illustrations des différents MDD dans un cadre simplifié avec quatre couches de noeuds (ceci à des fins d'illustration). Le MDD$_{\mathcal{R}_4}$ n'est pas représenté car il y aurait besoin de plus de couches pour en montrer le principe (en particulier la réinitialisation de la somme), mais conceptuellement il suit le même principe que le MDD$_{\mathcal{R}_2}$. Dans chaque MDD, un chemin est indiqué en rouge. Ce chemin correspond à une même phrase. Cette phrase sera donc une solution pour notre problème, comme résultats de l'intersection des MDD. Remarquons que pour des raisons de lisibilité, les états de MDD$_{\mathcal{R}_3}$ n'ont été représentés que pour la solutions mise en évidence, en supposant que l'on utilise des 2-grammes (en vert).

tésien des domaines de toutes les couches. On complétera ces séquences avec le mot vide, le mot qui ne peut être suivi que par le mot vide. Ainsi, toutes les séquences ont une taille 15. Dans ce MDD, il n'y a pas d'**état** et chaque **arc** est un mot.

MDD$_{\mathcal{R}_1}$ (associé à $\mathcal{R}_1$, des phrases utilisant un sous-ensemble de lemmes). Ce MDD va nous servir à modéliser le vocabulaire du lexique autorisé.

Dans ce MDD, il n'y a pas de dépendance entre les variables pour l'affectation des valeurs. Il n'y a donc pas de nécessité de définir une notion d'état qui permettrait de statuer sur la validité de l'existence des arcs. Ce MDD a donc $r + 1$ noeuds pour r variables.

Un **arc** correspond à un lemme appartenant au vocabulaire autorisé (Manulex). Chaque variable est libre de prendre n'importe quelle valeur du domaine. Dans la couche i, entre le noeud de la couche i et $i + 1$, il y a 3000 arcs sortant, i.e., un arc pour chaque mot du lexique.

Le MDD associé à $\mathcal{R}_1$ est très facile à construire. Il ne peut pas être réduit. Le MDD$_{\mathcal{R}_1}$ est un MDD universel. Une solution de ce MDD est une suite de lemmes d'une taille égale au nombre de couches du MDD : entre 9 et 15.

Nous faisons appel à la bibliothèque treetagger pour passer d'un mot à son lemme [23]. Remarquons que ce travail consistant à restreindre le vocabulaire des phrases avec un lexique aurait pu être réalisé au niveau des pré-traitements sur le corpus. L'intérêt de vérifier $\mathcal{R}_1$ avec un MDD se justifie dans notre méthodologie car nous voulons profiter de la modularité et de la robustesse de l'approche MDD pour ainsi être en mesure de changer à volonté le lexique autorisé des phrases. Remarquons que, MDD$_{\mathcal{R}_1}$ n'est pas nécessairement un sous-ensemble de MDD$_U$ car ils sont construits à partir de données différentes (au niveau corpus) et la nature des données qu'ils contiennent est aussi différentes. Le MDD$_{\mathcal{R}_1}$ contient seulement des lemmes (e.g., masculin singulier pour un nom, infinitif pour un verbe) construits à partir du lexique Manulex [10]. A l'inverse, le corpus peut lui être construit à partir de livre très spécifiques. En effet, un livre (même jeunesse) sur les volcans risque de ne pas utiliser le mot "tomate" qui appartient bien pourtant au lexique Manulex, et *a fortiori* au MDD$_{\mathcal{R}_1}$.

MDD$_{\mathcal{R}_2}$ (associé à $\mathcal{R}_2$, des phrases de 59 caractères). Ce MDD est le MDD responsable de modéliser les phrases dont la somme des caractères des mots est égale à 59 caractères. Cela revient à imposer une contrainte de somme [24] sur les phrases. Nous voulons definir un MDD dont la somme des labels des arcs qui composent toutes solutions est égale à 59. C'est un MDD de type somme (voir l'exemple de la Fig. 1).

Un **état** est représenté par un entier s égal à la valeur de la somme partielle de tous les arcs appartenant aux chemins qui mènent au noeud associé à l'état. Un état est valide si et seulement si de la couche 0 à $r - 1$ si $0 \leqslant s \leqslant 59$, et si à la couche r, $s = 59$; Un **arc** est le nombre de caractères d'un mot. Une solution du MDD est une suite de 15 entiers ayant pour somme 59 caractères.

MDD$_{\mathcal{R}_3}$ (associé à $\mathcal{R}_3$, des phrases de 9 à 15 mots). Ce MDD est le MDD de base qui fait le lien avec le dictionnaire de n-grammes. Il va contenir toutes les combinaisons de mots et indirectement de n-grammes possibles conduisant à produire des phrases.

Un **état** est un n-gramme. Dans notre construction, les n-grammes des premières et dernières couches doivent correspondre à des n-grammes de début et fin de phrase. Ceci permet d'éliminer une partie des phrases n'aboutissant pas et une partie de celles qui laissent ouvertement sur leur fin.

Un **arc** est un mot, un n-gramme dans un état est donc un historique des n derniers arcs qui appartenant au chemin entre `root` et le noeud ayant pour état un n-gramme donné. Le mot vide est un mot possible si il ne peut être suivi que par le noeud `tt` ou un mot vide. Cette astuce permet soit de construire un MDD contenant des phrases de taille différente mais aussi à ajouter des couches de manières à ce que tout les MDD fassent la même taille pour une composition.

$\text{MDD}_{\mathcal{R}_4}$ (associé à $\mathcal{R}_4$, affichage des phrases). Ce MDD est aussi un MDD de type somme comme le $\text{MDD}_{\mathcal{R}_2}$. L'idée est ici aussi de modéliser une somme sur les éléments des phrases, mais cette fois-ci sur la taille (la largeur) des caractères au sens d'une police de caractère définie.
Initialement, un **état** est représenté par un entier s la valeur de la somme partielle de tous les arcs appartenant aux chemins qui mènent au noeud associé à l'état. On peut aussi définir un intervalle de somme valide dans lequel un état doit appartenir pour modéliser la taille des espace appartenant à un intervalle défini. Il y a un noeud entre chaque paire d'arcs, un arc correspond à un mot dans notre modèle. C'est au niveau du noeud, et donc dans l'état que les espaces doivent être gérés.
Un **arc** est la taille d'un mot dans sa police d'écriture.
Dans $\mathcal{R}_4$, il y a trois lignes, ce qui signifie qu'il faut en réalité vérifier trois sommes différentes, une pour chaque ligne, pour ce faire on ajoute à la définition d'un état, trois variables booléenne, pour connaître quelle est la ligne qui est entrain d'être calculée. Un état est valide de 0 à r-1 si la somme s est inférieur à taille maximale d'une ligne avec des petits espacements. De plus, si la taille des espaces appartient à l'encadrement des espaces de $\mathcal{R}_4$ l'état est valide, la variable booléenne associée à la ligne courante devient vrai et enfin la somme est remise à zéro. Au niveau de la couche r, il faut aussi vérifier que les variables booléennes associées à la ligne 1 et 2 soient vraies.

2.4 Opérations entre MDD : calcul des solutions

Pour obtenir les solutions qui respectent toutes les contraintes, on calcule le MDD final comme l'intersection à la volée des MDD définis précédemment. Cette opération d'intersection à la volée et toutes les autres opérations sur les MDD repose sur un algorithme associé à l'opérateur générique APPLY $\bigoplus$ défini dans [18, 7]. Tout l'intérêt de cette intersection à la volée est de permettre de réaliser cette opération de conjonction de contraintes efficacement sur les MDD.
Le MDD final MDD_F se calcule de la façon suivante :

$$\begin{aligned}
\text{MDD}_{I_1} &= \text{MDD}_U \bigcap \text{MDD}_{\mathcal{R}_1},\\
\text{MDD}_{I_2} &= \text{MDD}_{I_1} \bigcap \text{MDD}_{\mathcal{R}_2},\\
\text{MDD}_{I_3} &= \text{MDD}_{I_2} \bigcap \text{MDD}_{\mathcal{R}_3},\\
\text{MDD}_F &= \text{MDD}_{I_3} \bigcap \text{MDD}_{\mathcal{R}_4}.
\end{aligned}$$

3 Résultats

Le modèle décrit dans la Section 2 est implémenté en Java 17 dans un solveur de MDD (MDDLib) développé au sein de l'équipe de recherche. Les expériences ont été réalisées sur une machine Ubuntu 18.04 utilisant un CPU Intel(R) Xeon(R) Gold 5222 @ 3.80GHz et 188 GB de RAM.

3.1 Constitution de corpus de n-grammes

Pour constituer notre corpus de n-grammes, nous sommes partis d'un ensemble de 658 livres appartenant à la catégorie jeunesse. Ce choix est motivé par le fait vouloir générer in fine des phrases de structure simple et utilisant un lexique simple (voir $\mathcal{R}_1$). Pour se convaincre de la pertinence de notre approche qui consiste à générer des phrases plutôt que d'en chercher des existantes, nous avons d'abord regardé combien de phrases étaient de type MNREAD. Sur les 3165037 phrases contenues dans le corpus brut, seulement quatre phrases étaient de bonnes candidates pour être des phrases MNREAD (voir Tableau 1).

Je couvris la tête de mes mains pour me protéger du serpent
Je lançai de grands coups de pied pour tenter de me libérer
Mais il n'y aura pas assez de place pour emmener le sorcier
Elle poussa un petit cri lorsque la chose bougea de nouveau

TABLE 1 – Les quatre phrases de type MNREAD trouvées dans le corpus d'origine de 658 livres.

Partant de ces phrases, nous avons suivi trois étapes pour définir notre corpus de n-grammes : (i) Les quatre phrases identifiées dans le corpus brut (Tab. 1) ont été retirées. L'idée est de partir d'un corpus ne contenant aucune phrase de type MNREAD pour éviter tout biais (garder ces phrases serait donner des bouts de solution valides) et pour montrer comment on peut générer des phrases "ex-nihilo" ; (ii) Seules les phrases respectant la règle $\mathcal{R}_0$ ont été retenues ; (iii) Les phrases retenues sont décomposées en n-grammes (pour un n donné). L'ensemble des n-grammes obtenus pour toutes les phrases retenues définit notre corpus de référence.
En pratique, nous allons aussi considérer des sous-ensembles de n-grammes obtenus à partir de pourcentages de phrases retenues. La relation entre le nombre de n-grammes (uniques) et le pourcentage de phrases utilisées pour les obtenir, est présentée dans la Fig. 4 et la Table 2. Par exemple, si l'on choisit 50% des phrases du corpus, cela nous donne 2.8 millions de 5-grammes. On peut y voir la relation non-linéaire entre le nombre de n-grammes et le pourcentage de phrases considérées. Plus le nombre de grammes augmente plus le nombre de n-grammes uniques est élevé. Enfin, ces résultats suggèrent que le nombre de n-grammes pourrait être facilement augmenté par l'accroissement de la taille du corpus (il n'y a pas "d'effet plateau" à ce stade).

3.2 Analyse des performances

Le Tableau 3 présente une analyse des performances de notre approche pour différentes valeurs de n-grammes, aussi bien du point de vue computationnel que du nombre de solutions obtenues. Les pourcentages de corpus pour chaque valeur de n-gramme ont été choisi de façon à obtenir un nombre de solutions du même ordre de grandeur afin de pouvoir les comparer.
Nous pouvons faire quelques observations générales : Le MDD_{I_1} est un MDD facile à calculer, dans la mesure où il n'y a pas d'état, il s'agit de filtrer les valeurs des domaines de chaque couches.
Pour le MDD_{I_2}, cela revient à calculer un MDD somme, ce qui est habituellement assez rapide, mais il faut tout de

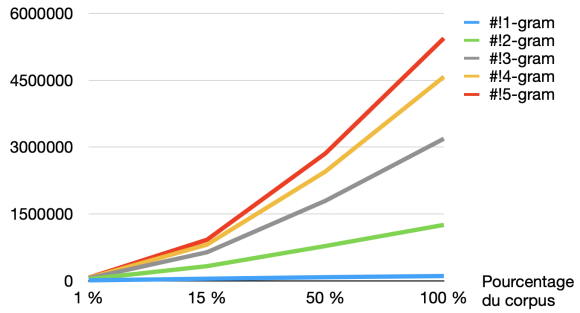


FIGURE 4 – Nombre de n-grammes uniques en fonction du pourcentage du corpus considéré et pour différents grammages.

	1%	10%	50%	100%
#!phrase	5947	59484	297405	594800
#!1-gram	11279	40004	81792	108231
#!2-gram	39747	244947	781699	1255730
#!3-gram	56254	451096	1796797	3189270
#!4-gram	63885	559297	2453949	4578657
#!5-gram	70069	630035	2860526	5444025

TABLE 2 – Caractéristiques du corpus : nombre de phrase, de mots (1-gram), de k-grammes ($k = 2..5$) uniques pour un pourcentage de corpus donné.

3-gramme avec 1% corpus					
MDD	nodes	arcs	sols	t	mem
I_1	16	91620	6.10^{56}	0m1s	2Go
I_2	712	3532517	$1,9.10^{42}$	0m23s	11Go
I_3	187079	104636	1601119	5m9s	36Go
F	11518	15928	9899	0m4s	2Go

4-gramme avec 15% corpus					
MDD	nodes	arcs	sols	t	mem
I_1	16	285975	2.10^{64}	0m1s	2Go
I_2	760	11403386	6.10^{44}	0h10m	12Go
I_3	356013	581504	2179577	2h47	39Go
F	18660	24387	10478	0m4s	2Go

5-gramme avec 100% corpus					
MDD	nodes	arcs	sols	t	mem
I_1	16	534225	1.10^{68}	0m1s	2Go
I_2	778	21639890	$1,3.10^{46}$	0h41m	16Go
I_3	529294	816078	1225086	10h6m	64Go
F	16459	13043	4335	0m2s	2Go

TABLE 3 – Analyse de performance. Pour chaque MDD intermédiaire (MDD_{I_i}) jusqu'au MDD final (MDD_F), ce tableau donne le nombre de noeuds, nombre d'arêtes, nombre de solutions, le temps de calcul de l'intersection, et la mémoire du processus. Trois résultats sont présentés avec différentes valeurs de n-grammes.

même dans le calcul de l'intersection à la volé : itérer sur le domaine du MDD précédant, pour une couche donné le cardinal du domaine est égale à (#arcs/#couches) de MDD_{I_1}. En effet, dans MDD_{I_1} tout les mots sont autorisés et ce pour toutes les couches à supposer qu'ils appartiennent au lexique.

Le MDD_{I_3} est long et coûteux en mémoire à calculer, c'est à ce moment là du calcul que nous pouvons dire que nous manipulons des phrases. Jusque là, c'était simplement des séquences de mots quelconques d'un nombre de caractères fixés et d'un vocabulaire restreint. En revanche, le MDD_F est très rapide à calculer, (quelques secondes à chaque fois) et peu coûteux en mémoire. Cependant, ceci est contextuel à sa position dans l'intersection. Le plus gros du calcul a déjà était effectué dans les intersections précédentes, un grand nombre d'arcs formant des solutions non valides ont déjà été filtrées dans le calcul de MDD_{I_3}.

Plus précisément, pour le MDD_{I_3}, cette croissance de la mémoire et du temps de calcul s'explique par deux raisons : (i) Nous calculons un grand nombre de phrases inutiles, car nous devons attendre d'atteindre les dernières couches pour vérifier qu'elles disposent d'un k-gramme de fin phrase. C'est à ce moment que nous savons si le chemin construit (i.e., la phrase) sera relié à `tt`. Remarquons que le nombre de n-grammes de début et de n-gramme de fin de phrase est fonction du corpus d'entrée, alors que le nombre "milieu de phrase" (séquences de plusieurs n-grammes formant un morceau de phrase) est fonction de la combinatoire possible des n-grammes entre eux pour n fixé. Plus généralement, il faudrait être en mesure d'anticiper la validité des états appartenant aux futures couches pour une couche donnée dans le MDD ; (ii) Il s'avère que ce $MDD_{\mathcal{R}_3}$ a une compression médiocre. En effet, le nombre de solutions est du même ordre de grandeur que le nombre d'arcs du MDD. Cette décompression à la construction est la conséquence l'utilisation de l'état n-gramme qui est discriminant en fonction des n derniers arcs visités.

Concernant la mémoire utilisée dans l'implémentation actuelle, il nous faut tout de même plus de 64 Go en 5-gramme pour effectuer nos calculs sur un corpus relativement petit. Notre corpus fait quelques centaines de Mo, sachant qu'il n'est plus inhabituel de voir des corpus manipulés d'une taille de l'ordre du gigaoctet, voire de la centaine de gigaoctets. Ceci est à nuancer par le fait que la constitution de notre corpus est un facteur important de la qualité des phrases engendrées. Il n'y a pas d'intérêt à le faire grossir sans avoir de justification. D'où notre choix de restreindre à des livres de littérature jeunesse dont on fait l'hypothèse d'y trouver des phrases simples.

Enfin, comme on peut le voir dans le Tableau 3, si nous souhaitons obtenir plus de phrases, soit on augmente la taille du corpus, soit on diminue la taille des n-grammes. En effet, on produit un nombre équivalent de phrase en utilisant 1% du corpus en 3-gramme qu'en utilisant 100% en 5-gramme.

3.3 Analyse des phrases

Parmi les phrases que nous sommes capable d'engendrer, il y a des phrases dites admissibles, i.e., dont la syntaxe et

Bien que je ne veux pas que les yeux de ce que ça me plaira
A l'autre bout de la place au fond de la voiture de ma mère
Allez le dire à mon père que je n'ai pas envie de me marier

TABLE 4 – Trois exemples de phrases en 3, 4 et 5 grammes respectivement.

Phrases admissibles
L'homme tourna la tête en direction de la voiture de police
Le dragon de métal tourna lentement la tête vers le plafond

Problème de sens
Prends place à côté de son mari pour sa douceur et sa grâce
J'avoue que je n'ai pas le temps de me réchauffer plus tard
La pluie formait un ruisseau de plus en plus insupportables

Problème de syntaxe
Vous lui attrapez la main et vous aide à monter sur son dos

TABLE 5 – Exemples de phrases en 5-grammes de différents types : (i) deux phrases admissibles, (ii) trois phrases ayant un problème de sens, (iii) une phrase ayant un problème de syntaxe.

Le policier passa la main dans sa poche et se mit à compter
Le policier passa la main dans sa poche et se mit à marcher
Le policier passa la main dans la poche arrière de mon jean
Le policier passa la main dans la poche de son jean déchiré
Le policier passa la main dans la poche de sa nouvelle robe

TABLE 6 – Exemples de phrases engendrées en 5-grammes ayant le même début de phrase.

J'essaie de changer de sujet avant de la perdre pour de bon
Tuez tous ceux qui se trouvaient sur le dessus de son siège
Dis à mon père que je n'ai pas envie de me les mettre à dos
L'homme avait tiré de sa poche un gros rat gris qui dormait
Posez une main sur le bras de son mari et le roi de ce pays
J'imagine qu'on ne peut pas savoir tant qu'on ne l'a pas vu
Pour ma part je me tourne et me dirige vers le bout du quai
Prends place à côté de son mari pour sa douceur et sa grâce
Très peu de choses arrivent sans que je le voie de mes yeux
Or sa mère lui avait demandé de reculer du bord de la route
J'espère que je n'ai pas vu mes parents depuis plus d'un an
Allez le dire à mon père que je n'ai pas envie de me marier
J'avoue que je n'ai pas le temps de me réchauffer plus tard
Vous seriez surpris de ce que venait de dire le vieil homme
Cette fois encore le jeune homme ne se décidait pas à poser
J'ai du mal à croire que tu me dises que tu ne m'aimes plus
Rien qui ressemble à une petite maison en bois et en pierre
Ceux qui ont perdu la vie à cause de ce que nous avons fait
Comme il regrettait à présent de ne pas se noyer de chagrin

TABLE 7 – Proposition d'un jeu de 19 phrases engendrées à partir de 5-grammes.

le sens sont corrects, mais aussi des phrases dont le sens est curieux, ou bien des phrases dont la syntaxe n'est pas correcte (voir Tab. 5).

On engendre des phrases qui respectent des règles mais on a tendance à engendrer toutes les variantes d'une même amorce phrase (voir Tab. 6). Ces variantes peuvent être toutes de bonne qualité comme on peut le voir avec l'amorce : "Le policier passa la main dans sa poche". Cependant, dans l'optique de former un jeu de phrases pour test, remarquons que nous devrons garantir une variabilité pour éviter d'avoir dans le même jeu de phrases des bouts de phrases similaires.

On constate alors que plus on diminue la taille des n-grammes plus la qualité grammaticale et sémantique des phrases diminuent. (voir Tab. 4)

Dans le Tableau 7, on présente un jeu de 19 phrases que nous avons sélectionnées parmis les 4000 solutions obtenues. Hormis le fait que ces phrases devront faire l'objet d'une validation expérimentales, une prochaine analyse consistera à estimer le pourcentage de phrases admissibles parmis les phrases générées (en terme sémantique et de syntaxe).

4 Conclusion

Ces résultats préliminaires montrent qu'il est possible de modéliser les règles du test MNREAD avec succès dans le paradigme MDD et d'être en mesure de prendre en comptes ces règles dès la génération. Comme nous l'avons montré, il est maintenant possible d'engendrer un grand nombre de phrases : plusieurs milliers en 5-grammes.

Par rapport à l'approche de Mansfield et al. [13], notre approche présente plusieurs avantages. D'abord, notre approche ne nécessite pas un travail manuel préliminaire long et fastidieux de construction de structures de phrases admissibles. A l'inverse, notre approche ne requiert que la constitution d'un corpus de phrases. Aussi en profitant de la combinatoire inhérente à l'utilisation de n-grammes, notre approche permet beaucoup plus facilement d'engendrer des phrases d'une plus grande diversité. Tout l'intérêt de notre approche est aussi qu'elle est *a priori* facilement généralisable dans différentes langues. Nous avons pu l'appliquer français (où les accords et les conjugaisons sont plus complexes qu'en anglais), là où la méthode de Mansfield [13] profite de cette flexibilité de l'anglais qui permet de construire plus facilement des *templates*. Il est toutefois difficile de nous comparer en terme de temps de calcul d'autant plus que l'entrée de nos méthodes sont très différentes (*templates* de phrases vs n-grammes). La méthode proposée par Mansfield et al. consiste à engendrer puis tester ; à l'inverse, notre méthode à pour esprit d'éviter de construire autant que possible des phrases non valides (même si certaines contraintes nous oblige à construire la phrase jusqu'au dernier mot pour s'assurer de sa validité). La composition des résolutions des sous-problèmes résulte en une convergence vers l'ensemble exhaustif de solutions du problème pour un corpus donné.

Notre méthode produit des phrases majoritairement cor-

rectes syntaxiquement, même si cette syntaxe peut-être lourde ou maladroite. Ceci est intéressant car, alors que rien n'est imposé en terme de syntaxe, une structure syntaxique en général correcte est obtenue grâce à la combinaisons n-grammes. Par contre, notre méthode ne peut aucunement garantir le sens des phrases. Ceci étant dit, à partir de ces milliers de phrases respectant toutes les contraintes MNREAD, il nous a été facile d'identifier un nouveau jeu de 19 phrases candidates ce qui, en soit, est déjà un résultat très important.

Dans la suite de ce travail, nous souhaitons donc considérer la modélisation d'autres contraintes, non pas associées à de nouvelles règles mais associées à la syntaxe et au sens. Tout l'objectif est d'être capable d'obtenir des phrases qui intègrent ces contraintes dès la génération pour éviter la phase de vérification a posteriori. Pour y parvenir, une idée sera de considérer des travaux qui tendent à montrer que cette notion d'acceptabilité d'une phrase est corrélée à la fréquence des mots dans une langue et donc indirectement, entre autres, aux probabilités qui peuvent être calculées à partir de modèles de langue comme celui des n-grammes [8].

Aussi nous avons choisi d'utiliser des MDD déterministes, mais nous envisageons d'utiliser des MDD non-deterministes comme perspectives d'amélioration du modèles notamment pour les $\text{MDD}_{\mathcal{R}_4}$ et $\text{MDD}_{\mathcal{R}_2}$ et ainsi, reconsidérer l'ordre de l'intersection proposée dans l'optique d'améliorer les performances.

Pour finir, nous voulons insister sur l'une des grandes forces de cette approche qui est sa modularité. En fonction des besoins, on va pouvoir modifier comme on le souhaite les contraintes ou en rajouter. Ainsi, les perspectives à plus long terme de ce travail sont de voir comment adapter notre approche dans d'autres problématiques de génération de texte sous contraintes, comme pour la dyslexie ou l'apprentissage de langues.

Références

[1] S.B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, C(27) :509–516, June 1978.

[2] D. Bergman, A. A. Cire, W.-J. Hoeve, and J. Hooker. *Decision Diagrams for Optimization*. Springer Publishing Company, Incorporated, 1st edition, 2016.

[3] R.R.A. Bourne, S.R. Flaxman, T. Braithwaite, M.V. Cicinelli, A. Das, J.B. Jonas, J. Keeffe, J. Kempen, J. Leasher, H. Limburg, K. Naidoo, K. Pesudovs, S. Resnikoff, A. Silvester, G.A. Stevens, N. Tahhan, T.-Y. Wong, H.R. Taylor, R. Bourne, and P. Sieving. Magnitude, temporal trends, and projections of the global prevalence of blindness and distance and near vision impairment : a systematic review and meta-analysis. *The Lancet Global Health*, 5(9) :e888–e897, 2017.

[4] A. Calabrese, C. Owsley, G. McGwin, and G.E. Legge. Development of a reading accessibility index using the MNREAD acuity chart. *JAMA Ophthalmol.*, 134(4) :398–405, April 2016.

[5] M.D Crossland, G.E. Legge, and S. C. Dakin. The development of an automated sentence generator for the assessment of reading speed. *Behavioral and Brain Functions : BBF*, 4 :14–14, 2007.

[6] Z. Hu, Z. Yang, X., R. Salakhutdinov, and E. P. Xing. Toward controlled generation of text. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, page 1587–1596. PMLR, 2017.

[7] V. Jung and J.-C. Régin. Efficient operations between MDDs and constraints. In *Proceedings of the 19th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 2022.

[8] J. H. Lau, A. Clark, and S. Lappin. Grammaticality, acceptability, and probability : A probabilistic view of linguistic knowledge. *Cognitive science*, 41(5) :1202–1241, 2017.

[9] G. E. Legge. *Psychophysics of Reading in Normal and Low Vision*. CRC Press, 2 edition, 2021.

[10] B. Lété, L. Sprenger-Charolles, and P. Colé. Manulex : A grade-level lexical database from french elementary-school readers. *Behavior Research Methods, Instruments & Computers*, 36 :166–176, 2004.

[11] J.S. Mansfield, S.J. Ahn, G.E. Legge, and A. Luebker. A new reading-acuity chart for normal and low vision. *Ophthalmic and Visual Optics/Noninvasive Assessment of the Visual System Technical Digest*, 3 :232–235, 1993.

[12] J.S. Mansfield and G.E. Legge. The mnread acuity chart. In *Psychophysics of reading in normal and low vision*, page 167–191. Taylor and Francis, 2006.

[13] S. Mansfield, N. Atilgan, A. Lewis, and G. Legge. Extending the mnread sentence corpus : Computer-generated sentences for measuring visual performance in reading. *Vision research*, 158 :11–18, 2019.

[14] P. Márquez-Neila, M. Salzmann, and P. Fua. Imposing hard constraints on deep networks : Promises and limitations. *CoRR*, abs/1706.02025, 2017.

[15] J.-L. P., D. Paillé, and T. Baccino. A new sentence generator providing material for maximum reading speed measurement. *Behav Res*, 47 :055–1064, 2015.

[16] G. Perez. *Diagrammes de décision : contraintes et algorithmes*. PhD thesis, Université Côte d'Azur, 2017.

[17] G. Perez and J.-C. Régin. MDDs : Sampling and probability constraints. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, page 226–242, 2017.

[18] G. Perez and J.-C. Régin. Efficient operations on MDDs for building constraint programming models. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2015.

[19] W. Radner. Reading charts in ophthalmology. *Graefe's Archive for Clinical and Experimental Ophthalmology*, 255(8) :1465–1482, August 2017.

[20] S. N. Ravi, T. Dinh, V. S. Lokhande, and V. Singh. Explicitly imposing constraints in deep networks via conditional gradients gives improved generalization and faster convergence. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01) :4772–4779, 2019.

[21] P. Roy, G. Perez, J.-C. Régin, A. Papadopoulos, F. Pachet, and M. Marchini. Enforcing structure on temporal sequences : the Allen constraint. In *International conference on principles and practice of constraint programming*, page 786–801. Springer, 2016.

[22] G. S. Rubin. Measuring reading performance. *Vision Research*, 90(C) :43–51, September 2013.

[23] H. Schmid. Probabilistic part-ofispeech tagging using decision trees. In *New methods in language processing*, page 154, 2013.

[24] M. Trick. A dynamic programming approach for consistency and propagation for knapsack constraints. *Annals of Operations Research*, 118 :73–84, 2003.

Optimisation de parcage des avions à l'aéroport Paris Charles de Gaulle

Thibault Falque[1,2], Christophe Lecoutre[2], Bertrand Mazure[2], Karim Tabia[2]

[1] Exakis Nelite
[2] CRIL, Univ Artois & CNRS

{falque,lecoutre,mazure,tabia}@cril.univ-artois.fr

Résumé

Plus que jamais, les acteurs du transport aérien (c'est-à-dire les compagnies aériennes et les aéroports), dans un climat de forte concurrence, doivent bénéficier d'une gestion optimisée des ressources aéroportuaires afin d'améliorer la qualité de leurs services et de contrôler les coûts induits. Dans cet article, nous étudions le problème d'affectation des parkings d'aéroport (APAP). Nous introduisons plusieurs modèles de programmation par contraintes (CP), que nous comparons, et présentons quelques résultats expérimentaux prometteurs à partir de données provenant d'ADP (Aéroport de Paris).

Mots-clés

programmation par contraintes, modélisation

Abstract

More than ever, air transport players (i.e., airline and airport companies), in a strongly competitive climate, need to benefit from a carefully optimized management of the airport resources in order to improve the quality of service and to control the induced costs. In this paper, we investigate the Airport Parking Assignment Problem (APAP). We introduce a Constraint Programming (CP) model, and present some promising preliminary experimental results from data coming from ADP (Aéroport de Paris).

Keywords

constraint programming, modelization

1 Introduction

Avant la crise sanitaire du COVID-19, l'Association Internationale du Transport Aérien (IATA) prévoyait que le nombre de passagers aériens allait presque doubler d'ici 2036, pour atteindre 7,8 milliards de personnes. Dans un tel contexte, l'optimisation de la gestion des ressources aéroportuaires reste essentielle pour maîtriser les coûts induits tout en conservant une bonne qualité de services.

Pour de nombreux problèmes de planification et d'ordonnancement du transport aérien, les techniques et outils développés à partir de la programmation mathématique et de la programmation par contraintes restent essentiels. En particulier, lorsque les compagnies aériennes ont accès aux ressources fournies par l'aéroport, la consommation de ces ressources (par exemple, les banques d'enregistrement, les parkings d'avions) doit être soigneusement planifiée tout en optimisant une fonction objectif déterminée par certaines règles commerciales ; voir, par exemple, [24, 22, 10].

Un problème classique du transport aérien est le problème d'affectation des portes d'aéroport (AGAP), qui consiste à affecter chaque vol à une porte d'embarquement disponible tout en maximisant à la fois le confort des passagers et l'efficacité opérationnelle de l'aéroport ; voir les études dans [1, 6] et les modèles dans [20, 21, 23]. Concenant le problème d'allocation de parking, [14] présente une modélisation MILP du problème en considérant des contraintes de capacité, d'ombrage et de satisfaction.

Étant donné que des améliorations significatives ont été apportées ces dernières années à la programmation par contraintes (CP), comme, par exemple, des algorithmes de filtrage (et de compression) efficaces pour les contraintes de table [4, 7], ou la génération de clauses paresseuses [5], s'attaquer aux tâches d'optimisation des aéroports avec la CP reste une question très intéressante. Dans cet article, nous nous intéressons au problème d'affectation des parkings d'aéroport (APAP) tel que défini à l'aéroport international CDG. Nous proposons des approches de Programmation par Contraintes (CP), et montrons son intérêt potentiel en présentant quelques résultats expérimentaux prometteurs.

Dans la section 2 nous introduisons le problème des parkings de l'aéoroport international de Charles de Gaulle. Dans la section suivante, nous présentons le problème sous forme de modèle COP. Dans la section 4, nous comparons les différentes variantes du modèle proposé sur des données de planification réelles (post-COVID) avec la solution actuellement utilisée par ADP. Nous faisons aussi une comparaison des modèles entre eux. Enfin nous terminons par une conclusion et des perspectives.

2 Problème de parcage des avions à Paris CDG

L'aéroport CDG est le neuvième plus grand aéroport du monde en termes de trafic de passagers. On compte 498175 mouvements (décollages ou atterrissages) par an, ce qui

correspond à environ 1400 mouvements par jour.
À l'aéroport, l'un des problèmes combinatoires à résoudre consiste à affecter chaque vol à un parking disponible (Pkg) ; ce problème est appelé APAP (Airport Parking Assignment Problem). Dans cette section, nous décrivons cette problématique. Tout d'abord, il existe deux types de parkings à l'aéroport : les parkings « **contacts** », désignés par $\mathcal{P}_{ct}$, qui sont des parkings « reliés » à un terminal par une porte et une passerelle, et les parkings « **au large** », désignés par $\mathcal{P}_{rt}$, qui sont des parkings accessibles en bus.

Définition 1 (Ensemble des parkings)**.** L'ensemble de tous les parkings, $\mathcal{P}_{ct} \cup \mathcal{P}_{rt}$, est noté par $\mathcal{P}$.

Définition 2 (Rotation)**.** Une **rotation** est l'ensemble des actions (atterrissage, stationnement, décollage) nécessaires à un avion, lors de son passage à CDG.

Bien qu'une rotation implique généralement deux vols (pour l'arrivée et le départ de l'avion), elle peut parfois n'impliquer qu'un seul vol (arrivée ou départ seul). Dans le cas d'un vol arrivée seul, l'avion occupe un parking avant d'être déplacé vers un hangar. Dans le cas d'un vol départ, l'avion sort d'un hangar avant d'occuper un parking. Pour chaque rotation, nous disposons des informations (données) suivantes :

- **start**(ψ) est l'heure de début de la rotation (c'est-à-dire l'heure à laquelle l'avion doit être géré parce qu'il vient d'atterrir ou qu'il sort d'un hangar) ;
- **end**(ψ) est l'heure de fin de la rotation (c'est-à-dire l'heure à laquelle l'avion ne doit plus être géré car il décolle ou va dans un hangar) ;
- **ntasks**(ψ) est le nombre de tâches (comme expliqué ci-dessous) formant la rotation ;
- **kind**(ψ) est le type avion de la rotation ;
- **airline**(ψ) est la compagnie de la rotation.

On peut considérer l'affectation d'un vol à un parking pendant une rotation comme une **tâche**. En effet, en fonction de la durée de la rotation, l'avion peut être déplacé plusieurs fois. Un vol peut alors être placé sur un seul parking pendant toute la durée de sa rotation (1 tâche), sur deux parkings successifs (2 tâches), voire trois parkings (3 tâches). Dans ce dernier cas le parking du milieu est nécessairement au large. Les vols à une seule arrivée ou un seul départ n'occupent nécessairement que 1 parking (tâche). La construction de l'ordonnancement du problème de stationnement consiste à trouver un stationnement pour chaque tâche de chaque rotation.
Il convient de noter que les conditions permettant de déterminer si un avion est mobile ou non (c'est-à-dire s'il utilise plus d'un stationnement) dépendent de certains temps de traitement tels que, par exemple, le temps de transit minimum, le temps de traitement minimum pour le départ, etc.
L'ensemble complet de rotations Ψ est divisé en trois sous-ensembles Ψ_1, Ψ_2, et Ψ_3, en fonction du fait que les rotations sont composées d'une, deux ou trois tâches.
Concernant la j^{me} tâche d'une rotation ψ, on suppose que l'on connaît le poids de la tâche $w_{\psi,j}$ (utilisé dans l'expression de la fonction objectif), l'heure de début de la tâche $s_{\psi,j}$, et l'heure de fin de la tâche $e_{\psi,j}$. Notez que $s_{\psi,1} = \texttt{start}(\psi)$ et $e_{\psi,k} = \texttt{end}(\psi)$ si k est l'indice de la dernière tâche de la rotation ψ.
Bien entendu, l'affectation d'un parking à une tâche est soumise aux préférences de placement des compagnies aériennes. Pour chaque tâche de chaque rotation, une récompense est donnée : la récompense de l'affectation du parking p à la tâche j de la rotation ψ est désignée par $r^p_{\psi,j}$. En supposant que nous disposons d'une série [1] de variables binaires $x^p_{\psi,j}$ associées à chaque tâche de rotation (indiquant quel parking sera utilisé), et en considérant le poids $w_{\psi,j}$ de chaque tâche de rotation, on peut alors définir la fonction objectif global comme suit :

$$\text{maximize} \sum_{\substack{\psi \in \Psi \\ j \in 1..\texttt{ntasks}(\psi)}} w_{\psi,j} \times x^p_{\psi,j} \times r^p_{\psi,j} \qquad (1)$$

En outre, il existe certaines contraintes physiques, imposées par l'infrastructure.

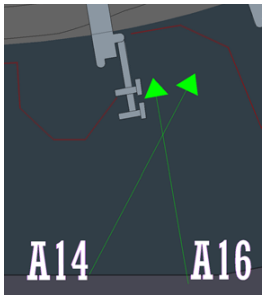


FIGURE 1 – Exemple pour la contrainte ombrage

Définition 3 (Capacité)**.** Les contraintes de `capacité` empêchent certains types d'avions d'être placés sur certains parkings.

Définition 4 (Ombrage)**.** Les contraintes `ombrages` bloquent le positionnement d'un avion sur certains parkings voisins (quel que soit le type d'avion).

Définition 5 (Ensemble des parkings ombrés)**.** Pour tout parking $p \in \mathcal{P}$, on note $\mathcal{S}_p$ l'ensemble des parkings ombrés par p.

Exemple 1. *Par exemple, pour la contrainte* `ombrage`*, la figure 1 montre que, si un avion est placé sur le parking* A14*, alors le parking* A16 *est « ombré ».*

Définition 6 (Réduction)**.** Les contraintes `réductions` sont similaires aux contraintes `ombrages` sauf qu'elles prennent en compte les types avions. La constrainte `reduction` est définie par des 4-tuples (k, p, k', ϕ) avec ϕ étant un ensemble de stationnements. Pour chaque stationnement p' dans ϕ, un avion de type k' est autorisé sur p' si un avion de type k est placé sur p.

Définition 7 (Ensemble des réductions)**.** Nous notons $\mathcal{D}$ l'ensemble de toutes ces réductions (tous les 4-tuples).

1. Notez que nous n'utilisons pas de variables 0/1 dans le modèle proposé à la section 3.

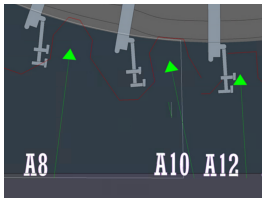


FIGURE 2 – Exemple pour la contrainte ombrages

Exemple 2. *À titre d'illustration, considérons la figure 2, l'*`ombrage` *ne sera effectif que si un certain type d'avion a été placé sur* A10 *et la réduction permettra toujours à un sous-ensemble de types avions d'être placés sur* A8 *et* A12*.*

Définition 8 (NoOverlap)**.** Une contrainte `NoOverlap` reflète l'impossibilité physique d'affecter deux tâches (deux vols) au même parking.

Définition 9 (Chevauchement de tâches)**.** Une paire (ψ, i) correspondant à la tâche i d'une rotation ψ se chevauche avec une paire (ψ', j) correspondant à la tâche j d'une rotation ψ' si $s_{\psi,i} < e_{\psi',j}$ et $s_{\psi',j} < e_{\psi,i}$.

Définition 10 (Ensemble de tâches en chevauchement)**.** Les tâches se chevauchant avec (ψ, i) sont désignées par l'ensemble $\mathcal{O}_{\psi,i}$, qui contient toutes les paires (ψ', j) se chevauchant avec (ψ, i).

3 Modèle COP

Nous allons maintenant décrire le problème plus formellement au moyen d'un réseau de contraintes. Un *Réseau de contraintes* (CN) est constitué d'un ensemble fini de variables soumises à un ensemble fini de contraintes. Chaque variable x peut prendre une valeur dans un ensemble fini appelé le *domaine* de x. Chaque contrainte c est spécifiée par une relation définie sur (le produit cartésien des domaines de x). Une *solution* d'un CN est l'affectation d'une valeur à chaque variable de sorte que toutes les contraintes soient satisfaites. Un *Réseau de contraintes sous optimisation* (CNO) est un réseau de contraintes qui comprend en plus une fonction objectif `obj` qui fait correspondre toute solution à une valeur dans $\mathbb{R}$.

Pour la modélisation des CNO, également appelés problèmes d'optimisation de contraintes (COP), plusieurs langages de modélisation et bibliothèques existent tels que, par exemple, OPL [30], MiniZinc [25, 29], Essence [13] et PyCSP^3 [18]. Notre choix s'est porté sur la bibliothèque Python récemment développée PyCSP^3 qui permet de générer des instances spécifiques (après avoir fourni des données ad hoc) au format XCSP3 [2, 3], qui est reconnu par certains solveurs de CP bien connus comme ACE (AbsCon Essence) [15], OscaR [26], Choco [27], et PicatSAT [35].

Nous allons définir dans cette section plusieurs variantes du problème d'affectation des parkings. Nous commencerons par les élements communs des différents variantes puis définirons les spécificités de chaque variante.

Rot.	`Compagnie`	`#Tches`	`Type`	`Dbut`	`Fin`
ψ_1	$\mathtt{a}_1$	1	k_1	$8h00$	$10h00$
ψ_2	$\mathtt{a}_2$	2	k_2	$8h00$	$12h00$
ψ_3	$\mathtt{a}_2$	2	k_2	$12h00$	$16h00$
ψ_4	$\mathtt{a}_3$	3	k_3	$9h00$	$15h00$
ψ_5	$\mathtt{a}_3$	3	k_4	$10h00$	$16h00$

TABLE 1 – Données sur les rotations

Pkg	Capacité
$\mathtt{p}_1$	$\{k_1, k_2\}$
$\mathtt{p}_2$	$\{k_3, k_2\}$
$\mathtt{p}_3$	$\{k_1, k_2, k_3\}$
$\mathtt{p}_4$	$\{k_1, k_2, k_3, k_4\}$

TABLE 2 – Capacité

3.1 Définitions et contraintes communes

Tout d'abord, nous présentons les variables de notre modèle. En plus de la variable utilisée pour compter le nombre de rotations qui ne sont pas cassées, nous avons besoin de deux matrices de variables pour représenter le stationnement assigné, et les récompenses associées :

- p est une matrice de $|\Psi| \times 3$ variables ayant pour domaine $\{0, ..., |\mathcal{P}| - 1\}$; $p[\psi][j]$ représente l'index du parking (code) assigné à la *jth* tâche de ψ ;
- r est une matrice de $|\Psi| \times 3$ variables ayant pour domaine $\{0, ..., 100\}$; $r[\psi][j]$ représente la satisfaction de la compagnie pour la *jth* tâche de la rotation ψ.

Notez que, par souci de simplicité, nous introduisons toujours trois variables dans p et r pour chaque rotation, même si une seule tâche (ou deux tâches) est impliquée. Pour traiter ces cas, nous introduisons des contraintes d'égalité obligeant les variables à prendre la même valeur. En pratique, on pourrait éviter d'introduire de telles variables redondantes.

Deuxièmement, nous devons introduire les contraintes dans notre modèle. En raison de la nature du problème (et des données), il est naturel d'utiliser ce qu'on appelle des contraintes de table, qui énumèrent explicitement soit les tuples autorisés (tableau positif) ou les tuples interdits (tableau négatif) pour une séquence de variables (représentant la portée de la contrainte). Des algorithmes efficaces pour ces tableaux de contraintes ont été développés au cours de la dernière décennie. [8, 16, 19, 31].

Pour la suite, nous introduisons un exemple afin d'illustrer les contraintes ajoutées. Considérons un exemple avec un ensemble de 5 rotations $\{\psi_1, ..., \psi_5\}$, et un ensemble de 4 stationnements $\{\mathtt{p}_1, ..., \mathtt{p}_4\}$. Les informations concernant les rotations sont données dans le tableau 1.

3.1.1 Contraintes sur les tâches

Tout d'abord, il est nécessaire de présenter quelques contraintes de stationnement concernant les tâches des rotations. Lorsque la division d'une rotation en 2 ou 3 tâches n'est pas possible (parce que le temps de rotation n'est pas assez grand), nous devons nous assurer que les 3 variables de la rotation sont affectées au même parking (parce que,

pour simplifier, comme déjà mentionné, nous avons toujours 3 variables de p qui sont introduites pour chaque rotation). Nous ajoutons donc les contraintes suivantes :

$$p[\psi][1] = p[\psi][2] = p[\psi][3], \forall \psi \in \Psi_1 \quad (2)$$

$$p[\psi][2] = p[\psi][3], \forall \psi \in \Psi_2 \quad (3)$$

$$p[\psi][2] \in \mathcal{P}_{rt}, \forall \psi \in \Psi_3 \quad (4)$$

L'équation 2 oblige les 3 variables de p pour la rotation ψ à être assignées au même parking (car la rotation n'implique qu'une seule tâche). L'équation 3 force les deuxième et troisième variables de p de la rotation ψ à être assignées au même parking (parce que la rotation n'implique que deux tâches). Enfin, l'équation 4 oblige le parking du milieu à être placé sur un parking au large.
Pour notre exemple, supposons que $\Psi_1 = \{\psi_1\}$, $\Psi_2 = \{\psi_2, \psi_3\}$, et $\Psi_3 = \{\psi_4, \psi_5\}$, nous devons alors spécifier :

$$p[\psi_1][1] = p[\psi_1][2] = p[\psi_1][3]$$
$$p[\psi_2][2] = p[\psi_2][3] \wedge p[\psi_3][2] = p[\psi_3][3]$$
$$p[\psi_4][2] \in \mathcal{P}_{rt} \wedge p[\psi_5][2] \in \mathcal{P}_{rt}$$

3.1.2 Contraintes capacité

Pour appliquer de telles contraintes, nous avons simplement besoin de contraintes unaires. En supposant que l'ensemble des stationnements respectant la capacité d'une rotation ψ est noté $\mathcal{P}_\psi$, nous ajoutons la contrainte suivante :

$$p[\psi][j] \in \mathcal{P}_\psi, \forall \psi \in \Psi, \forall j \in 1..3 \quad (5)$$

Nous forçons ici toutes les tâches d'une rotation ψ à prendre leurs valeurs dans l'ensemble $\mathcal{P}_\psi$.
La capacité de chaque parking (dans notre exemple) est définie dans la Table 2. Nous devons donc ajouter les contraintes unaires suivantes par rapport à la rotation ψ_1 :

$$p[\psi_1][j] \in \{\mathtt{p}_1, \mathtt{p}_3, \mathtt{p}_4\}, \forall j \in 1..3$$

3.1.3 Calcul de la satisfaction compagnie

Selon les préférences des compagnies, des récompenses différentes sont associées à des stationnements différents. Nous pouvons utiliser des contraintes de table binaires pour « calculer » les récompenses lors du filtrage de ces contraintes.

$$\langle p[\psi][j], r[\psi][j] \rangle \in \{(p, r^p_{\psi,j}) \mid p \in \mathcal{P}_\psi\}, \quad \forall \psi \in \Psi, \forall j \in 1..\mathtt{ntasks}(\psi) \quad (6)$$

Nous ajoutons également une contrainte table en conflit pour interdire les parkings incompatibles avec cette tâche.
Pour notre exemple, supposons que les récompenses sont données par la matrice suivante :

Compagnie / Pkg	$\mathtt{p}_1$	$\mathtt{p}_2$	$\mathtt{p}_3$	$\mathtt{p}_4$	$\mathtt{p}_5$
a_1	75	75	100	50	50
a_2	60	0	100	50	50
a_3	0	100	80	50	50

À partir de ces données, nous devons par exemple ajouter la contrainte suivante concernant la première rotation ψ_1 (notez que $\mathtt{ntasks}(\psi_1) = 1$) :

$$\langle p[\psi_1][1], r[\psi_1][1] \rangle \in \{(\mathtt{p}_1, 75), (\mathtt{p}_2, 75), (\mathtt{p}_3, 100), (\mathtt{p}_4, 50), (\mathtt{p}_5, 50)\}$$

3.1.4 Contraintes Ombrage

Rappelons que lorsqu'un parking p_1 est « ombré » par un parking p_2 alors les deux valeurs associées ne peuvent pas être assignées ensemble à une paire de de tâches qui se chevauchent. Cela conduit à des contraintes de table négatives binaires. Bien que cela ne soit pas explicitement indiqué ci-dessous, l'attribution de la même valeur deux fois pour une paire de tâches qui se chevauchent est également interdite (contraint `NoOverlap`). Les contraintes d'ombrage s'écrivent comme suit :

$$\langle p[\psi][i], p[\psi'][j] \rangle \notin \{(p, p') \mid p \in \mathcal{P}_\psi \wedge p' \in \mathcal{P}_{\psi'} \cap \mathcal{S}_p\}, \quad \forall \psi \in \Psi, \forall i \in 1..3, \forall (\psi', j) \in \mathcal{O}_{\psi,i} \quad (7)$$

Dans le contexte de notre exemple, une contrainte d'`ombrage` existe entre les parkings p_1 et p_3. Nous avons $\mathcal{O}_{\psi_1,1} = \{(\psi_2, 1), (\psi_4, 1)\}$, $\mathcal{P}_{\psi_1} = \mathcal{P}_{\psi_2} = \{p_1, p_3, p_4, p_5\}$, $\mathcal{P}_{\psi_4} = \{p_2, p_3, p_4\}$, et $\mathcal{S}_{p_1} = \{p_3\}$.
Par conséquent, nous pouvons ajouter les deux contraintes suivantes concernant ψ_1 : $\langle p[\psi_1][1], p[\psi_2][1] \rangle \notin \{(p_1, p_3)\}$ et $\langle p[\psi_1][1], p[\psi_4][1] \rangle \notin \{(p_1, p_3)\}$.

3.1.5 Contraintes Réduction

Ce dernier sous-ensemble de contraintes prend en compte les réductions sous la forme de contraintes binaires négatives de table.

$$\langle p[\psi][i], p[\psi'][j] \rangle \notin \{(p, p') \mid p' \in \phi\}, \quad \forall \psi \in \Psi, \forall i \in 1..\mathtt{ntasks}(\psi), \quad \forall \langle k, p, k', \phi \rangle \in \mathcal{D} \mid k = \mathtt{kind}(\psi), \quad \forall (\psi', j) \in \mathcal{O}_{\psi,i} \mid k' \neq \mathtt{kind}(\psi') \quad (8)$$

Pour notre exemple, une contrainte `réduction` existe entre les parkings p_1 et p_2. Supposons que $\mathcal{D} = \{(k_1, p_1, k_3, \{p_2\}), (k_2, p_1, k_4, \{p_2\})\}$. Nous avons également $\mathcal{O}_{\psi_1,1} = \{(\psi_2, 1), (\psi_4, 1)\}$ et $\mathcal{P}_{\psi_1} = \{p_1, p_3, p_4\}$.
La tâche 1 de la rotation ψ_2 chevauche la tâche 1 de la rotation ψ_1. Rappelons que $\mathtt{kind}(\psi_2) = k_2$. Ainsi, pour la réduction de ψ_1, nous ajoutons la contrainte suivante qui interdit la paire de valeurs (p_1, p_2) pour la paire de variables $\langle p[\psi_1][1], p[\psi_2][1] \rangle$. En d'autres termes, il est interdit d'utiliser p_2 avec la rotation ψ_2 car ψ_2 n'a pas le type autorisé par p_2 après avoir placé ψ_1 sur p_1 (on dit que sa capacité est `réduite`).
Nous ajoutons alors la contrainte suivante $\langle p[\psi_1][1], p[\psi_2][1] \rangle \notin \{(p_1, p_2)\}$.
Bien que ψ_1 chevauche également avec ψ_4, il n'y a aucune restriction à considérer avec ψ_1 car ψ_4 a pour capacité le type k_3 (voir Table 1) qui est autorisé par rapport à la réduction.

3.1.6 Fonction objectif

La fonction objectif s'exprime comme suit en considérant les variables de récompenses :

$$\text{maximize} \sum_{\substack{\psi \in \Psi \\ j \in 1..\texttt{ntasks}(\psi)}} w_{\psi,j} \times r[\psi][j] \qquad (9)$$

3.2 Variante classique

Avec les variables et contraintes présentées dans la précédente section, nous pouvons définir une première variante du problème composée des contraintes 2, 3, 4, 5, 6,7 8 et de la fonction objectif 9. Nous noterons cette variante `no-variant`.

3.3 Variante `AllDifferent`

Dans cette variante nous proposons de modifier la contrainte ombrage en postant en plus des contraintes tables déjà proposées des contraintes `alldifferent`. Notons que pour cette contrainte, il est tentant de vouloir utiliser une contrainte `AllDifferent` comme déjà proposé dans l'état de l'art [28, 11]. Dans ces approches, une contrainte `AllDifferent` est ajoutée entre toutes les paires de tâches qui se chevauchent cependant cela ne peut pas fonctionner ici car elle serait moins contraignante que ce qu'exprime la contrainte `ombrage`. En effet une contrainte `AllDifferent` obligerait $p[\psi][i]$ et $p[\psi'][j]$ à être différent. Or en mettant p et p' comme valeur à ces deux variables, la contrainte serait respectée mais violerait la contrainte `ombrage`.

Néanmoins il est possible d'utiliser des contraintes `AllDifferent` avec un graphe d'intervalles. Chaque sommet du graphe représente une tâche (l'intervalle de temps de la tâche) et est connecté par une arête à un autre sommet, si et seulement si, il existe un chevauchement entre les deux intervalles. Le graphe d'intervalles serait donc, pour notre problème, noté $\mathcal{G} = (\mathtt{V}, \mathtt{E})$ avec $\mathtt{V} = \{\mathtt{I}_{\psi,i}, \forall \psi \in \Psi, \forall i \in 1..3\}$ et $\mathtt{E} = \{(\mathtt{I}_{\psi,i}, \mathtt{I}_{\psi',j}), \forall \psi \in \Psi, \forall i \in 1..3, \forall (\psi', j) \in \mathcal{O}_{\psi,i}\}$. $\mathtt{I}_{\psi,i}$ représente l'intervalle de temps de la tâche i de la rotation ψ. C'est-à-dire $e_{\psi,i} - s_{\psi,i}$. On note alors $\mathcal{C}_\mathcal{G}$ l'ensemble des cliques maximums. Pour un tel ensemble, nous pouvons poster les contraintes `AllDifferent` suivantes.

$$\text{allDifferent}(\{p[\psi][i], \forall (\psi, i) \in \mathtt{c}\}), \forall \mathtt{c} \in \mathcal{C}_\mathcal{G} \qquad (10)$$

Pour cette variante nous avons donc les mêmes contraintes que la variante précédente et ajoutons la contrainte 10. Nous notons `alldifferent` cette variante du problème.

3.4 Variante `not-break`

Cette variante du problème tient compte du fait que le déplacement d'un avion d'un parking à un autre a un certain coût. Pour cette raison, le nombre de stationnements utilisés pour une rotation (avec 2 ou 3 tâches) doit être minimisé. Pour faire cela, nous ajoutons une nouvelle variable `nb`, ayant pour domaine $\{0, ..., |\Psi|\}$, comptant le nombre de fois où rotation n'est pas rompue, c'est-à-dire le nombre de fois où un seul stationnement est utilisé pendant toute la durée de la rotation.

Nous devons ajouter la contrainte suivante afin de calculer le nombre de rotations « non-cassées » :

$$\mathtt{nb} = \sum_{\psi \in \Psi_2} s_{\psi,1=\psi,2} + \sum_{\psi \in \Psi_3} s_{\psi,1=\psi,2=\psi,3} \qquad (11)$$

où $s_{\psi,1=\psi,2}$ est une variable booléenne valant vraie si le même parking a été affecté pour la première et troisième tâche de la rotation ψ et où $s_{\psi,1=\psi,2=\psi,3}$ est vraie si toutes les tâches de la rotation ψ ont été affectées au même parking.

La fonction objectif est modifée comme suit :

$$\text{maximize} \sum_{\substack{\psi \in \Psi \\ j \in 1..\texttt{ntasks}(\psi)}} w_{\psi,j} \times r[\psi][j] + \mathtt{nb} \qquad (12)$$

Dans cette variante nous ajoutons la contrainte 11 et nous changeons la fonction objectif en 12. Nous noterons cet ensemble de contraintes `not-break`.

4 Expérimentations

Cette section présente quelques résultats expérimentaux concernant notre modèle et ses variantes sur des instances réelles d'ADP. Pour réaliser les expériences, nous avons utilisé le solveur de contraintes ACE, qui est le nouvel avatar de AbsCon [2]. Les options par défaut du solveur ont été utilisées : $\texttt{wdeg}^{\texttt{ca.cd}}$ [34] comme heuristique de choix de variables, `lexico` comme heuristique de choix de valeurs, `last-conflict` (lc) [17] comme méthode paresseuse pour simuler un retour en arrière intelligent, `solution-saving` [32, 9] pour simuler une forme de recherche de voisinage, et une politique de redémarrage géométrique [33] avec un cutoff de base fixé à 10 mauvaises décisions et une raison fixée à 1.1. Une seule option a été modifiée de manière empirique : `Bivs` [12] comme heuristique d'ordonnancement des valeurs. `Bivs` est une heuristique de choix de valeurs qui sélectionne la valeur dont l'affectation, une fois propagée, offre la plus petite (resp. grande) borne inférieure (resp. supérieure) pour la variable objectif. Nous noterons $\texttt{ACE}^{\texttt{BIVS}}$ lorsque l'éxécution a été réalisée avec cette heuristique.

Enfin le temps limite a été fixé à 180s afin de respecter les temps de résolution imposés par ADP.

4.1 A propos des instances

Le système actuel d'ADP permet de planifier au maximum 2 semaines d'utilisation des parkings. En raison de la quantité de vols, il ne permet pas de faire l'ensemble de l'aéroport en une seule fois, les planifications sont toujours effectuées terminal par terminal ou par groupe de terminaux.

Pour notre étude, nous nous sommes restraints au terminal $2A$ de CDG. L'étude est composée de 5 instances faisant respectivement 1, 2, 3,7 et 14 jours de planification.

L'étude est composée de 3 variantes du modèle comme présenté dans la section précédente. Nous notons $\mathcal{I}_{no-variant}$

2. https ://github.com/xcsp3team/ace

Instances	# Tâches	# Variables	# Contraintes
2A - 1 jour	34	180	483
2A - 2 jours	68	362	932
2A - 3 jours	105	570	1459
2A - 7 jours	267	1398	3749
2A - 14 jours	552	2868	7777

TABLE 3 – Description des instances de $\mathcal{I}_{no-variant}$

les instances de la variante définie dans la section 3.2. L'ensemble des instances de la variante de la section 3.4 est noté $\mathcal{I}_{variant-breaking}$. Enfin l'ensembe des instances de la section 3.3 est noté $\mathcal{I}_{variant-alldifferent}$.

4.2 Solution d'ADP

Le système d'ADP utilise un solveur MILP (*Mixed-Integer Linear Programming*) pour résoudre le problème. La résolution s'arrête dès qu'une première solution est trouvée. En fonction du paramètrage, il est possible de faire une recherche locale pour améliorer la solution. Cette recherche locale vient notamment regrouper les tâches sur un même parking, pour éviter de « casser » des rotations. Nous noterons $\texttt{ADP}_{default}$ le solveur d'ADP sans recherche locale et $\texttt{ADP}_{ls}$ le solveur d'ADP avec recherche locale.

4.3 Comparaison entre la solution d'ADP et les modèles proposés

Nous allons d'abord comparer les modèles COP proposés avec la solution d'ADP en faisant varier également l'heuristique de choix de valeur puis nous ferons une comparaison des modèles proposés entre eux toujours en faisant varier l'heuristique de choix de valeur.

Étude de $\mathcal{I}_{no-variant}$ La figure 3a montre le score maximal obtenu du premier modèle COP proposé ($\mathcal{I}_{no-variant}$) par rapport à la solution trouvée par ADP. Nous pouvons observer que `ACE` dépasse les performances de la méthode actuelle. Notons que des optima sont trouvés pour les instances 1, 2 et 3 jours. La figure 3b présente les résultats en utilisant $\texttt{ACE}^{\texttt{BIVS}}$. Là encore, nous pouvons observer que les résultats de notre modèle sont meilleurs que ceux de la solution actuelle.

Étude de $\mathcal{I}_{variant-alldiff}$ La figure 3c montre le score maximal obtenu du second modèle COP proposé ($\mathcal{I}_{no-alldiff}$) qui ajoute les contraintes `AllDifferent` entre toutes les cliques maximales du graphe d'intervalles. Nous pouvons observer que là encore `ACE` dépasse les performances de la méthode actuelle. Notons que des optima sont trouvés pour les instances 1, 2 et 3 jours. La figure 3d présente les résultats en utilisant $\texttt{ACE}^{\texttt{BIVS}}$. Nous pouvons observer aussi que les résultats de notre modèle sont meilleurs que ceux de la solution actuelle.

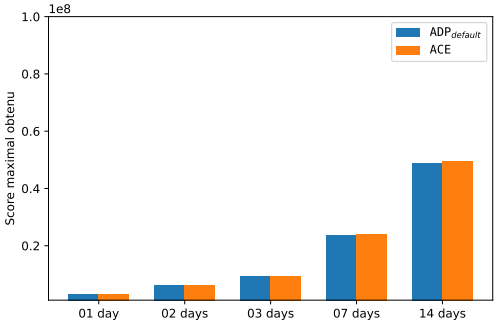
Étude de $\mathcal{I}_{variant-notbreak}$ La figure 3e montre le score maximal obtenu par le second modèle COP proposé prenant en compte le fait de ne pas « casser » les rotations par rapport à la solution d'ADP avec recherche locale dédiée. La figure 3f présente la même approche mais avec $\texttt{ACE}^{\texttt{BIVS}}$. Dans les deux cas, l'approche générique sans recherche locale dépasse la solution utilisée actuellement.
Nous pouvons dire que peu importe la combinaison de l'approche que nous proposons, on constate une amélioration de la borne et nous supprimons la nécessité d'avoir une recheche locale dédiée et donc une phase supplémentaire de calcul comme c'est le cas actuellement.

4.4 Comparaison des modèles proposés

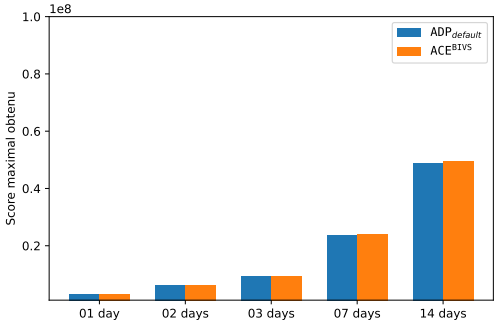
Dans cette dernière partie des experimentations, nous allons comparer les approches proposées entre elles en retirant les solutions d'ADP.
La figure 4 présente l'évolution des bornes trouvées en fonction du temps pour chaque instance du problème. Nous comparons les différentes approches proposées. Nous pouvons remarquer que sur la première instance (figures 4a), pour laquelle un optimum est trouvé, les différentes approches ont un comportement équivalent. Sur les instances de 2 et 3 jours (figures 4b et 4c), nous pouvons observer que les approches avec les contraintes `AllDifferent` permettent de converger plus vite vers l'optimum. Enfin l'utilisation de `Bivs` permet de converger encore plus vite vers l'optimum.
La différence est plus marquée sur les deux instances « difficiles » (figure 4d et 4e). Pour l'instance de 7 jours, les approches `AllDifferent` avec ou sans l'utilisation de `Bivs` stagnent à la première solution trouvée. Dans le cas de l'instance 14 jours, l'approche `AllDifferent` trouve initialement une meilleure solution que les autres approches puis stagnent à la même valeur jusqu'à la fin alors les autres approches progressent.

5 Conclusion

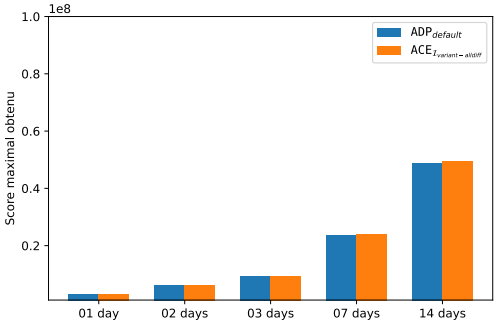
Dans cet article, nous nous intéressons au problème d'affectation des parkings d'aéroport (APAP) tel que défini à l'aéroport de Paris CDG. Nous avons proposé plusieurs variantes d'un modèle d'optimisation par contraintes (COP), en exploitant principalement les contraintes de table, et nous avons présenté une première évaluation empirique de notre approche. Nous avons également montrer que le problème n'était pas équivalent au problème de l'état de l'art et que l'on ne pouvait pas directement utiliser les approches de l'état de l'art basée sur des contraintes `AllDifferent` pour toutes les tâches qui se chevauchent. Cependant une approche avec des contraintes `AllDifferent` et le calcul des cliques maximum du graphe d'intervalles est possible.
Les résultats obtenus insitent, le groupe ADP à remplacer sa solution propriétaire actuelle par la nôtre, basée sur un solveur de contraintes générique open-source et l'utilisation d'un format open-source XCSP facilitant la modélisation des problèmes d'optimisations et leurs modifications. Le problème d'affectation des parkings est une première étape. Dans le futur, l'objectif est de remplacer l'ensemble des optimisations actuelles par une approche COP. Parmi les ressources aéroportuaires à optimiser en plus des parkings avion, nous travaillons sur des solutions pour les banques d'enregistrements, les tapis et les jetées bagages mais également les flux passagers.



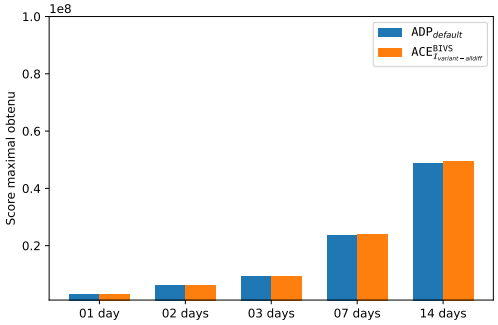
(a) Comparaison des bornes obtenues sur l'ensemble $\mathcal{I}_{no-variant}$ par ACE et ADP$_{default}$



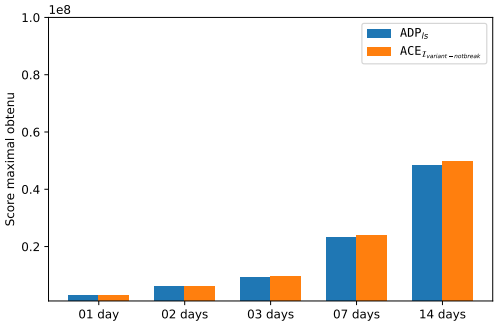
(b) Comparaison des bornes obtenues sur l'ensemble $\mathcal{I}_{no-variant}$ par ACE^{BIVS} et ADP$_{default}$



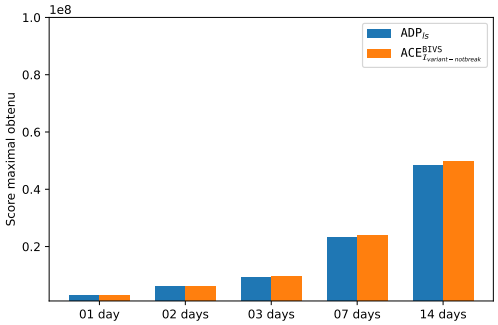
(c) Comparaison des bornes obtenues sur l'ensemble $\mathcal{I}_{variant-alldiff}$ par ACE et ADP



(d) Comparaison des bornes obtenues sur l'ensemble $\mathcal{I}_{variant-alldiff}$ par ACE^{BIVS} et ADP$_{default}$

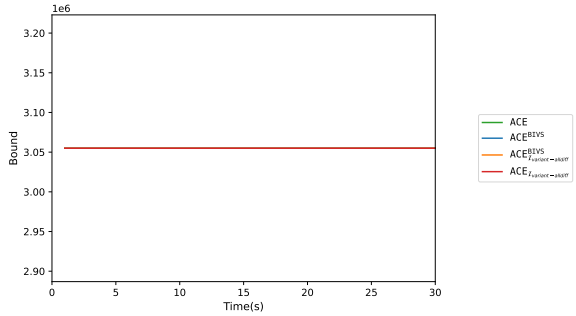


(e) Comparaison des bornes obtenues sur l'ensemble $\mathcal{I}_{variant-notbreak}$ par ACE et ADP$_{ls}$

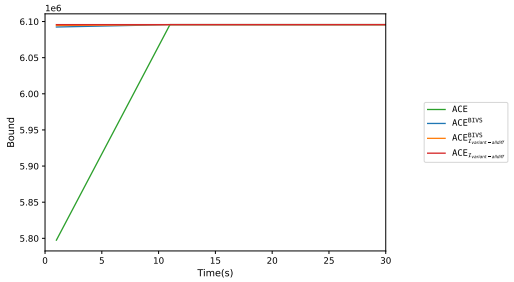


(f) Comparaison des bornes obtenues sur l'ensemble $\mathcal{I}_{variant-notbreak}$ par ACE^{BIVS} et ADP$_{ls}$

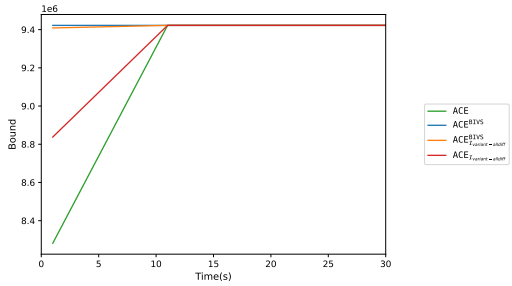
FIGURE 3 – Comparaison des différentes approches avec la solution d'ADP



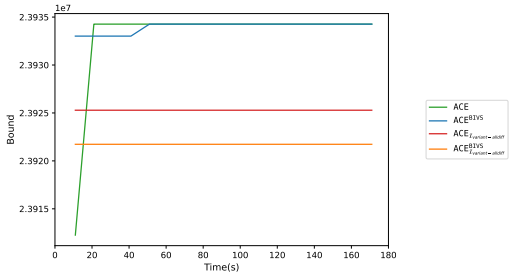
(a) Évolution de la borne en fonction du temps pour l'instance 1 jour



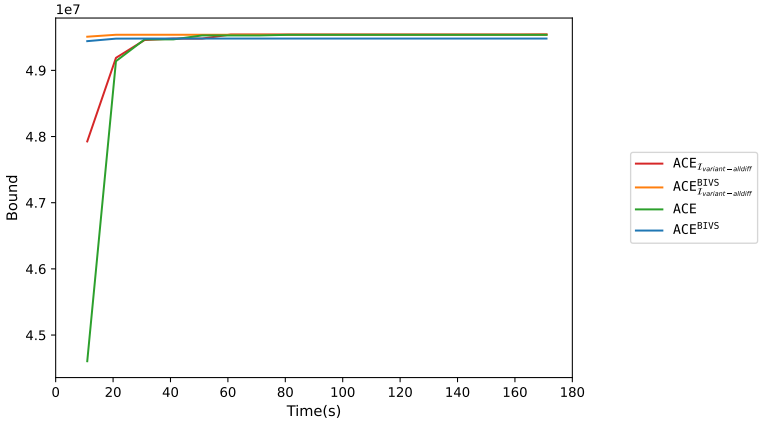
(b) Évolution de la borne en fonction du temps pour l'instance 2 jours



(c) Évolution de la borne en fonction du temps pour l'instance 3 jours



(d) Évolution de la borne en fonction du temps pour l'instance 7 jours



(e) Évolution de la borne en fonction du temps pour l'instance 14 jours

FIGURE 4 – Evolution des bornes en fonction du temps pour l'ensemble des instances

Références

[1] Abdelghani Bouras, Mageed A. Ghaleb, Umar S. Suryahatmaja, and Ahmed M. Salem. The Airport Gate Assignment Problem : A Survey. *The Scientific World Journal*, 2014 :e923859, November 2014.

[2] F. Boussemart, C. Lecoutre, G. Audemard, and C. Piette. XCSP^3 : An Integrated Format for Benchmarking Combinatorial Constrained Problems. Technical Report arXiv :1611.03398, Specifications, CoRR, 2016.

[3] F. Boussemart, C. Lecoutre, G. Audemard, and C. Piette. XCSP^3-core : A Format for Representing Constraint Satisfaction/Optimization Problems. Technical Report arXiv :2009.00514, Specifications 3.0.6, CoRR, 2020.

[4] B. Le Charlier, M. T. Khong, C. Lecoutre, and Y. Deville. Automatic Synthesis of Smart Table Constraints by Abstraction of Table Constraints. In *Proceedings of IJCAI'17*, pages 681–687, 2017.

[5] G. Chu, P. Stuckey, M. Garcia de la Banda, and C. Mears. Symmetries and Lazy Clause Generation. In *Proceedings of IJCAI'11*, pages 516–521, 2011.

[6] Gülesin Sena Daş, Fatma Gzara, and Thomas Stützle. A review on airport gate assignment problems : Single versus multi objective approaches. *Omega*, 92 :102146, April 2020.

[7] J. Demeulenaere, R. Hartert, C. Lecoutre, G. Perez, L. Perron, J.-C. Régin, and P. Schaus. Compact-Table : Efficiently Filtering Table Constraints with Reversible Sparse Bit-Sets. In *Proceedings of CP'16*, pages 207–223, 2016.

[8] J. Demeulenaere, R. Hartert, C. Lecoutre, G. Perez, L. Perron, J.-C. Régin, and P. Schaus. Compact-Table : Efficiently Filtering Table Constraints with Reversible Sparse Bit-Sets. In *Proceedings of CP'16*, pages 207–223, 2016.

[9] 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.

[10] Guido Diepen, J.M. Akker, J.A. Hoogeveen, and J.W. Smeltink. Using column generation for gate planning at Amsterdam Airport Schiphol. January 2007.

[11] Mehmet Dincbas and Helmut Simonis. Apache - a constraint based, automated stand allocation system. pages 267–282, 10 1991.

[12] 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, Boston, MA, November 2017. IEEE.

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

[14] J. Guépet, R. Acuna-Agost, O. Briant, and J.P. Gayon. Exact and heuristic approaches to the airport stand allocation problem. *European Journal of Operational Research*, 246(2) :597–608, 2015.

[15] C. Lecoutre. *ACE : A Generic Constraint Solver*. Technical Report. v1on CoRR, to appear, October 2021.

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

[17] C. Lecoutre, L. Sais, S. Tabary, and V. Vidal. Reasoning from last conflict(s) in constraint programming. *Artificial Intelligence*, 173(18) :1592–1614, 2009.

[18] C. Lecoutre and N. Szczepanski. *PYCSP^3 : Modeling Combinatorial Constrained Problems in Python*. Technical Report. v1 on CoRR, arXiv :2009.00326, September 2020. 100 pages.

[19] Christophe Lecoutre. STR2 : Optimized Simple Tabular Reduction for Table Constraints. *Constraints*, 16(4) :341–371, 2011.

[20] Chendong Li. Airport Gate Assignment : New Model and Implementation. *arXiv :0811.1618 [cs]*, November 2008.

[21] Chendong Li. Airport Gate Assignment A Hybrid Model and Implementation. *arXiv :0903.2528 [cs]*, March 2009.

[22] A. Lim, B. Rodrigues, and Y. Zhu. Airport Gate Scheduling with Time Windows. *Artif Intell Rev*, 24(1) :5–31, September 2005.

[23] J. L'Ortye, M. Mitici, and H.G. Visser. Robust flight-to-gate assignment with landside capacity constraints. *Transportation Planning and Technology*, 44(4) :356–377, May 2021.

[24] R. S. Mangoubi and Dennis F. X. Mathaisel. Optimizing Gate Assignments at Airport Terminals. *Transportation Science*, 19(2) :173–188, 1985.

[25] N. Nethercote, P. Stuckey, R. Becket, S. Brand, G. Duck, and G. Tack. MiniZinc : Towards a Standard CP Modelling Language. In *Proceedings of CP'07*, pages 529–543, 2007.

[26] OscaR Team. OscaR : Scala in OR, 2012.

[27] C. Prud'homme, J.-G. Fages, and X. Lorca. Chocosolver, TASC, INRIA Rennes, LINA, Cosling S.A. 2016.

[28] Helmut Simonis. Models for global constraint applications. *Constraints*, 12(1) :63–92, mar 2007.

[29] P. Stuckey, R. Becket, and J. Fischer. Philosophy of the MiniZinc challenge. *Constraints*, 15(3) :307–316, 2010.

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

[31] H. Verhaeghe, C. Lecoutre, and P. Schaus. Extending Compact-Table to Negative and Short Tables. In *Proceedings of AAAI'17*, pages 3951–3957, 2017.

[32] 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.

[33] T. Walsh. Search in a small world. In *Proceedings of IJCAI'99*, pages 1172–1177, 1999.

[34] H. Wattez, C. Lecoutre, A. Paparrizou, and S. Tabary. Refining constraint weighting. In *Proceedings of ICTAI'19*, pages 71–77, 2019.

[35] N. F. Zhou, H. Kjellerstrand, and J. Fruhman. *Constraint Solving and Planning with Picat*. Springer, 2017.

Approche par contraintes pour une classe d'emplois du temps universitaires

Vincent Barichard, Corentin Behuet, David Genest, Marc Legeay, David Lesaint

[1] Univ Angers, LERIA, F-49000 Angers, France

{prenom.nom}@univ-angers.fr

Résumé

Le calcul d'emplois du temps universitaire est un problème d'optimisation combinatoire complexe tant dans sa modélisation que sa résolution. Nous proposons une approche par contraintes qui englobe la constitution de groupes, la distribution des salles et enseignants, leur allocation et l'ordonnancement des séances. Cette approche se fonde sur un langage dédié à base de règles (`UTP`) permettant de modéliser les différentes entités et contraintes d'une instance. Les instances `UTP` sont encodées en `XML` et un générateur convertit les règles en contraintes dans un format compatible avec les solveurs `MiniZinc` et `CHR++`. Nous présentons dans cet article le langage `UTP` et ces modèles de programmation par contraintes ainsi que des expérimentations préliminaires réalisées sur un cas d'études concret.

Abstract

University course timetabling are complex combinatorial optimization problems to model and solve. We propose a constraint-based approach which encompasses student sectioning, room and teacher distribution planning, session scheduling and resource allocation. Our approach is based on a domain-specific rule-based language `UTP` to model instance entities and constraints. `UTP` instances are encoded in `XML` and a flattener converts rules into constraints using formats supported by `MiniZinc` and `CHR`. This article presents the `UTP` language and the two constraint programming models as well as early expriments carried out on a real case study.

1 Introduction

L'organisation d'emplois du temps universitaires met en jeu des décisions d'ordre stratégique, tactique et opérationnel qui portent sur le maquettage des formations, la constitution des classes et groupes d'étudiants, l'affectation des services d'enseignement, le provisionnement de salles et d'équipements, et, in fine, la programmation des séances et des ressources [6]. Le périmètre de ces problèmes et le processus coordonnant leur résolution varie selon les pays et les institutions ainsi que le niveau d'automatisation et les sytèmes d'aide à la décision mis en oeuvre. Au sein des universités françaises, par exemple, les maquettes de formation sont par convention revues tous les 5 ans et les étudiants s'inscrivent aux formations et personnalisent leurs parcours avant chaque période d'enseignement. Classes et groupes sont alors constitués selon les profils d'étudiants et les seuils d'effectifs par classe. Enseignants et salles sont ensuite positionnés sur les cours à dispenser avant que les séances de classe ne soient programmées et leurs ressources allouées (voir Figure 1). Ce processus reste flexible (changements de personnels, etc.) et doit s'adapter aux impondérables qui rythment l'année universitaire (inscriptions tardives, absences, etc).

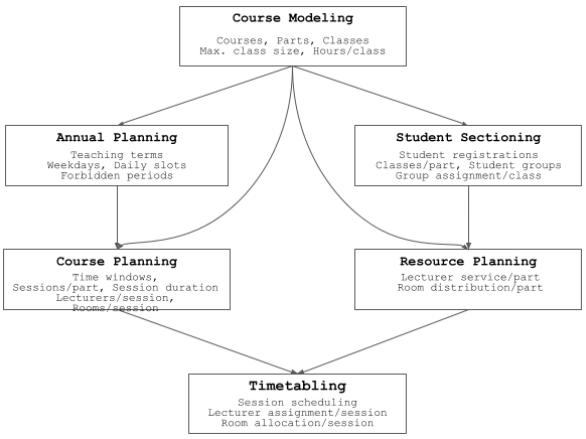


FIGURE 1 – Processus d'organisation d'emplois du temps

Nous proposons dans cet article un langage de modélisation pour une classe étendue de problèmes d'emplois du temps universitaires (`UTP`) se réduisant au problème de satisfaction de contraintes (`CSP`). Ce langage dédié permet de représenter différents aspects du problème relatifs au sectionnement de cours, la programmation de séances, la distribution de ressources et leur allocation afin d'adapter chaque instance aux exigences de son environnement. Le langage intègre un modèle formel et un langage de règles pour représenter entités et contraintes. Le modèle d'entités s'appuie sur un horizon de temps multi-échelles (i.e. semaines, jours et créneaux quotidiens), un ensemble de ressources (i.e. étudiants, salles et enseignants), et une structure hiérarchique de cours (i.e. cours, parties de cours, classes et séances). Chaque séance est à programmer individuellement sur l'horizon de temps et les ressources nécessaires

doivent lui être allouées.
Le modèle d'ordonnancement permet de représenter à la fois des séances mono-ressource et multi-ressources ainsi que des ressources disjonctives, cumulatives et hybrides. D'une part, les séances sont étiquetées à ressource unique (p. ex. cours magistral) ou à ressources multiples (p. ex. cours hybride en distanciel et présentiel) en quantifiant le nombre de salles et d'enseignants requis. Les étudiants se distribuent sur les cours selon leurs inscriptions alors que salles et enseignants se distribuent sur les parties de cours (p. ex. salles de travaux pratiques) ce qui détermine le domaine des ressources allouables à chaque séance. La charge de service est configurable pour les enseignants (i.e. nombre de séances à dispenser par partie de cours) et le volume de séances est figé pour les étudiants selon leur profil (i.e. toute partie de cours est obligatoire) tandis que les salles peuvent être utilisées à volonté.
L'utilisation simultanée d'une ressource n'est contrainte que pour les salles qui ont une capacité d'accueil indépassable, sauf dans le cas particulier des séances multi-salles qui s'appuient sur la capacité cumulée des salles allouées (p. ex. examen sur plusieurs salles). Pour ce qui concerne la programmation des séances, chaque partie de cours possède sa propre grille horaire et chaque classe impose de séquencer ses séances dans un ordre prédéfini. Chaque ressource peut donc être allouée à des séances se chevauchant (p. ex. classe obligatoire et tutorat) à l'exception des salles utilisées conjointement par une séance. Des règles peuvent être surimposées pour rendre des ressources, ou des classes de ressources, disjonctives. Enfin, le modèle impose de partitionner les étudiants en groupes et de ventiler les groupes sur les différentes classes en respectant des seuils d'effectifs et toute contrainte de sous-groupes posée entre classes.
Le langage de règles s'appuie sur un catalogue de prédicats qui permet d'exprimer des contraintes supplémentaires associant séances et entités. Chaque contrainte porte sur une ou plusieurs paires, appelées e-maps, et éventuellement des paramètres selon le prédicat utilisé. Une e-map associe une entité à un sous-ensemble de séances compatibles et s'interprète comme une affectation conditionnelle. Autrement dit, une contrainte ne s'évalue que sur les séances pour lesquelles e-map(s) et solution considérée concordent, c'est-à-dire proposent la même entité. Chaque prédicat peut s'appliquer indistinctement à des ressources ou des éléments de cours. Les e-maps peuvent donc être façonnées pour contraindre les séances allouables à une ressource (p. ex. indisponibilité d'un enseignant), les séances constitutives d'un élément de cours (p. ex. périodicité d'une classe), ou des séances individuelles (p. ex. parallélisation). À noter que les contraintes portant sur des éléments de cours sont de facto inconditionnelles. La Figure 4 présente trois contraintes : C1 et C2 portant chacune sur 2 classes, et C3 portant sur un enseignant. La contrainte C3 porte sur 4 séances mais ne s'appliquera pas qu'à celles qui seront finalement affectées à l'enseignant.
Plutôt que de poser des contraintes individuelles, les règles sont utilisées pour formuler des conjonctions de contraintes ciblant des classes d'entités et de séances (p. ex. règles disjonctives sur enseignants, restrictions temporelles sur un cursus). Chaque règle est liée à un prédicat et se définit par une contrainte quantifiée dont les quantificateurs restreignent le domaine de chaque variable e-map du prédicat. Un langage de sélecteurs est fourni pour construire et filtrer les domaines d'e-maps par rang de séances, identifiant et type d'entités, ou toute classe d'éléments étiquettés par l'utilisateur (p. ex., équipe enseignante, bloc de salles). Une règle dénote donc la conjonction de contraintes résultant de l'instanciation du prédicat sur le produit cartésien des domaines de variables. La Figure 4 décrit les contraintes générées à partir de 2 règles : les contraintes C1 et C2 résultent de la règle R1 et la contrainte C3 de la règle R2.

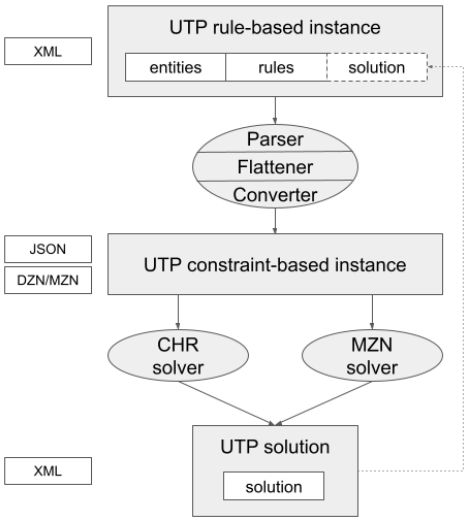


FIGURE 2 – Chaîne de traitements UTP

Le langage UTP est implémenté sous la forme d'un langage XML. L'implémentation comprend un schéma XML validant l'encodage d'instances à base de règles et une chaîne de traitements comportant un parseur XML, un générateur transformant les règles en contraintes, et un encodeur convertissant les instances résultantes dans un format approprié pour les solveurs (voir Figure 2). L'intégration d'un solveur suppose d'implanter le modèle et les prédicats du langage UTP et nous fournissons à cet effet deux implémentations alternatives en MiniZinc [18] et CHR [11].
La suite de l'article s'organise comme suit. Nous présentons brièvement le langage UTP et dressons une comparaison à l'état de l'art en section 2. La section 3 détaille les modèles de programmation par contraintes implémentés en MiniZinc et CHR. La section 4 présente les premiers résultats expérimentaux. La section 5 conclut et présente les perspectives envisagées pour la suite de ce travail.

2 Le langage UTP

Le langage UTP décompose la représentation d'une instance en 3 composantes : le modèle d'entités, l'ensemble de règles et la solution. Nous en donnons ici une description informelle. Une spécification formelle est présentée dans

[7], et [1] détaille la syntaxe XML et le format JSON des instances UTP et donne aussi accès aux codes sources des modèles MiniZinc et CHR++, aux outils, et à un benchmark d'instances.

2.1 Modèle d'entités

Le modèle d'entités d'une instance UTP est schématisé en Figure 3. Il définit l'horizon de temps, la structure des cours, l'ensemble des ressources, ainsi que des propriétés d'entités et des relations associatives. L'horizon de temps se décompose en un nombre de semaines, de journées hebdomadaires et de créneaux quotidiens qui sont propres à chaque instance. La décomposition des semaines en journées et celle des journées en créneaux quotidiens sont uniformes. Les semaines et jours se succédant sur l'horizon de temps ne sont pas supposés consécutifs alors que les créneaux quotidiens le sont. Ces derniers sont de durée égale et divisent la journée de 24h, p. ex., si elle se divise en 1440 créneaux, ils seront d'1 minute chacun. Les créneaux servent d'unité de temps pour dater démarrage et fin de séances et pour mesurer durées de séance, temps de déplacement entre salles et délais entre séances.

Les cours ont une structure arborescente, chaque cours (p. ex. Algo) se décomposant en parties de cours (p. ex. TP d'Algo), chaque partie de cours en classes (p. ex. Classe 2 de TP d'Algo) et chaque classe en séances pré-ordonnées (p. ex. 3ème séance de la Classe 2 de TP d'Algo). Les séances sont les tâches élémentaires à ordonnancer quand on résoud une instance UTP, leur nombre, durée et séquencement intra-classe étant fixés. Précisément, les classes d'une partie de cours comportent un nombre identique de séances de même durée, ces deux constantes étant propres à chaque partie de cours. D'autre part, le langage impose que les séances d'une classe soient séquencées dans toute solution selon le rang qui leur est associé dans la classe. Enfin, les séances sont non-interruptibles et en particulier, ne peuvent pas être à cheval sur deux journées.

Trois types de ressources sont modélisés : les salles, les enseignants et les étudiants (constitués en groupes). Toutes les ressources d'une instance sont déclarées et typées dans le modèle d'entités. Dans la pratique, différentes contraintes émises en amont s'appliquent aux ressources et aux créneaux de cours (p. ex., facultés imposant une grille horaire par type de cursus, départements mettant en œuvre des politiques de mise en commun de salles, étudiants s'inscrivant aux cours). Les contraintes les plus élémentaires sont des contraintes de compatibilité énumèrant les salles appropriées, les enseignants éligibles, les étudiants candidats et les horaires autorisés pour les différents cours. Précisément, à chaque partie de cours est assigné l'ensemble des créneaux de départ, de salles et d'enseignants qui sont autorisés pour toutes les séances de la partie (cf. Figure 3). Concernant les étudiants, l'inscription se fait au niveau des cours, un étudiant devant participer à toutes les parties d'un cours. La constitution des groupes d'étudiants s'effectue à la résolution du problème ou peut être fournie dans la composante solution.

L'utilisation des ressources est également soumise à des contraintes de demande et de capacité. Comme les modalités diffèrent d'un environnement à un autre, le langage prend en charge les ressources disjonctives et cumulatives ainsi que les séances à ressources uniques ou multiples. Étudiants, enseignants et salles sont qualifiés de ressource cumulative s'ils peuvent suivre, enseigner ou héberger des séances en parallèle. Les ressources cumulatives sont indispensables à la modélisation de cours non-obligatoires (p. ex. séances de tutorat facultatives pouvant chevaucher des cours obligatoires) et pour gérer des événements multi-classes (p. ex. salles hébergeant des examens mutualisés). Le langage n'impose aucune limite sur le nombre de séances simultanées auxquelles participent enseignants et étudiants. À l'inverse, les salles ne peuvent héberger que des séances dont l'effectif cumulé est en deça de leur capacité. La capacité des salles et les seuils d'effectif des classes sont encodés dans le modèle d'entités qui autorise par ailleurs des salles à capacité illimitée (p. ex. salles virtuelles). Notons que toute ressource est supposée cumulative par défaut mais des règles disjonctives peuvent être imposées par ressource ou par classe de ressources.

Les séances sont dites multi-ressources si on peut leur allouer plusieurs ressources du même type. Ce type de séances présente un intérêt pratique (p. ex. séances multi-salles pour enseignement hybride, séances de travaux pratiques supervisées par plusieurs enseignants, examens nécessitant plusieurs surveillants) et des contraintes s'appliquent alors aux volumes de ressources requis par séance. Ces dernières s'expriment dans le modèle par des contraintes de cardinalité déclarées sur les parties de cours, chaque partie indiquant le nombre d'enseignants requis par séance (potentiellement aucun) et s'il s'agit de séances à salle unique ou non (nrRoomsPerSession et nrTeachersPerSession en Figure 3). À noter qu'une instance peut mixer des séances mono-ressource et multi-ressources et des ressources disjonctives et cumulatives.

Le modèle d'entités incorpore également des contraintes de flot qui régissent la distribution des étudiants et des enseignants sur les cours. Ces contraintes sont habituellement émises en amont de la génération d'emplois du temps durant les phases d'inscription et de planification de capacité (p. ex. distribution des volumes horaires entre enseignants d'un département). Comme mentionné précédemment, les étudiants s'inscrivent uniquement aux cours. Résoudre une instance UTP implique donc de placer les étudiants dans les classes conformément à la structure des cours et aux inscriptions demandées. La règle adoptée est qu'un étudiant soit assigné à toutes les séances d'une seule classe dans chaque partie de cours. Des contraintes d'imbrication de groupes peuvent être posées entre classes (p. ex. aggréger des groupes de travaux pratiques pour constituer un groupe de cours magistral, préserver les mêmes groupes entre différents cours d'un cursus). Pour l'emploi du temps du personnel, chaque enseignant a un volume fixe de séances dans les parties de cours où il intervient, l'affectation des séances restant à déterminer par le solveur. En complément des types d'entités prédéfinis, le langage offre la possibilité d'étiqueter librement les entités du modèle. Les entités qui

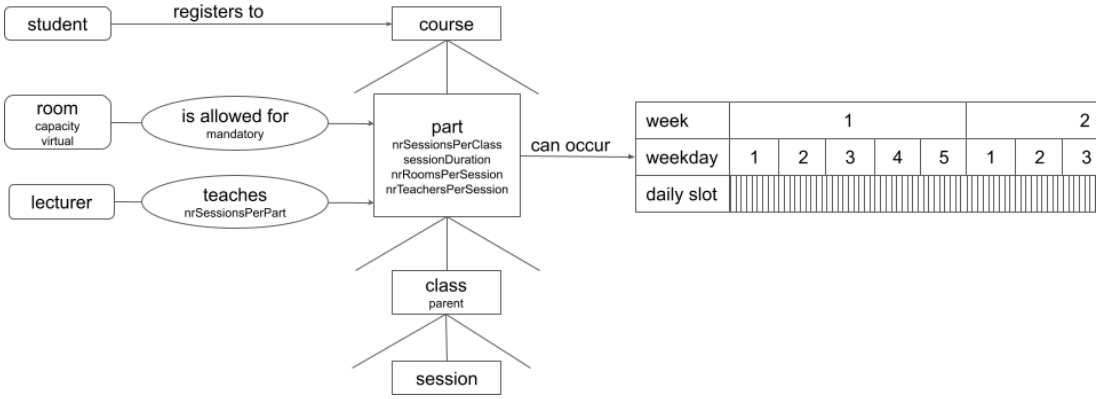


FIGURE 3 – Modèle d'entités

partagent la même étiquette forment un type à part entière. Ces étiquettes peuvent être utilisées à l'instar des types prédéfinis pour sélectionner les entités dans les règles.

2.2 Ensemble de règles

Les règles sont utilisées pour formuler des conjonctions de contraintes. Il s'agit de pouvoir exprimer, de manière concise, une ou plusieurs contraintes liées à un même prédicat. L'expression d'une règle implique d'identifier l'ensemble des séances que l'on souhaite contraindre et de choisir le prédicat à appliquer. La table 1 liste les prédicats `UTP` actuellement implémentés et utilisés dans nos instances.

Une règle porte, selon l'arité de son prédicat, sur un ou plusieurs domaines d'e-maps qui associent chacune une entité à un ensemble de séances compatibles. Les domaines d'e-maps ne sont pas représentés en extension mais à l'aide de sélecteurs. Un sélecteur permet de cibler des entités selon leur type, leur étiquette ou leur identifiant et de filtrer les ensembles de séances associées selon leurs rangs et leur compatibilité avec d'autres entités. Une règle se traduit ensuite par la conjonction de contraintes obtenue en instanciant le prédicat sur le produit cartésien des domaines d'e-maps sélectionnées.

La Figure 4 illustre la sélection de séances sur un exemple jouet et la génération automatique de contraintes à partir de deux règles. Le cours `algorithms` est divisé en une partie de cours magistraux `algoLec` et une partie de travaux pratiques `algoLab`. Le cours magistral est dispensé par `lecturer1` et ne contient qu'une classe de 4 séances. Les travaux pratiques sont encadrés par `lecturer1` et `lecturer2`, et sont constitués de 2 classes de 2 séances.

La première règle (R1) stipule que les travaux pratiques de chaque classe ne peuvent commencer qu'après la troisième séance de cours magistral (entités et séances annotées avec une étoile). Elle est associée au prédicat `sequenced` et utilise deux sélecteurs : le premier sélectionne la troisième séance de la partie `algoLec`, le second sélectionne, pour chaque classe, les premières séances de la partie `algoLab`. La partie `algoLab` ayant deux classes, la règle produit deux contraintes liées à `sequenced` : la première (C1) avec les séances `algoLec1:3` et `algoLab1:1`, la seconde (C2) avec les séances `algoLec1:3` et `algoLab2:1`.

La seconde règle (R2) stipule que `lecturer2` est indisponible sur une période donnée (losanges). Elle est associée au prédicat `forbidden_period` et utilise un sélecteur qui cible les séances de l'enseignant `lecturer2`. La règle produit une seule contrainte (C3) liée à `forbidden_period` (avec les paramètres spécifiant la période d'absence de l'enseignant, ici la période entre les créneaux 9120 et 9240) portant sur l'ensemble des séances de la partie `algoLab` où peut intervenir `lecturer2`. La contrainte ne sera effective que sur deux de ces séances étant donné que `lecturer2` encadre deux séances de travaux pratiques ; ces séances seront identifiées pendant la résolution.

2.3 Solution

L'élément solution comporte des choix de créneaux et de ressources pour les séances, de groupes pour les étudiants, et de classes pour les groupes. La solution ainsi représentée peut être partielle, voire vide, et n'est pas nécessairement consistante avec les contraintes de l'instance. Le support des solutions partielles permet de cibler et résoudre des sous-problèmes. Par exemple, une instance se réduit à un problème d'ordonnancement si elle se base sur une solution complète pour la constitution des groupes et l'affectation des ressources. De même, le support des solutions inconsistantes est un pré-requis pour la réparation de solutions qui seraient devenues inconsistantes suite à des changements non-anticipés (p. ex. absence d'un enseignant, indisponibilité d'une salle suite à des travaux).

Les groupes d'étudiants sont considérés comme le résultat du problème de sectionnement. Pour cette raison, les groupes font partie de l'élément solution, et définissent à la fois l'ensemble des étudiants qui les composent et les classes auxquelles ils appartiennent. Ce processus de sectionnement est soumis à différentes contraintes. D'une part, les groupes ne peuvent être constitués que d'étudiants qui sont inscrits aux mêmes cours. Ensuite, chaque groupe est insécable sauf dans le cas de séances multi-salles. Enfin, l'affectation des groupes aux classes doit satisfaire aux contraintes d'inclusion entre classes définies dans le modèle d'entités.

Nom	Arité	Paramétrique	Sémantique
same_daily_slot	1	non	Les séances démarrent le même créneau quotidien
same_weekday	1	non	Les séances démarrent la même jour de la semaine
same_weekly_slot	1	non	Les séances démarrent les mêmes créneau et journée
same_week	1	non	Les séances démarrent la même semaine
same_day	1	non	Les séances démarrent le même jour
same_slot	1	non	Les séances démarrent en même temps
forbidden_period	1	oui	Les séances ne peuvent pas débuter dans la période donnée
at_most_daily	1	oui	Le nombre de séances dans la période journalière définie est limité
at_most_weekly	1	oui	Le nombre de séances dans la période hebdomadaire définie est limité
sequenced	≥ 2	non	Les séances sont séquencées
weekly	1	non	Les séances démarrent les mêmes créneaux et jours de semaines successives
no_overlap	1	non	Les séances ne peuvent être en parallèle
travel	1	oui	Définition du temps de trajet entre salles
same_rooms	1	non	Les séances ont lieu dans les mêmes salles
same_students	1	non	Les mêmes étudiants suivent les séances
same_teachers	1	non	Les séances sont encadrées par les mêmes enseignants
adjacent_rooms	1	oui	Les séances doivent être dans des salles adjacentes
teacher_distribution	≥ 2	oui	Distribue la charge des enseignants dans les classes

TABLE 1 – Catalogue des prédicats UTP

2.4 Etat de l'art

Nous dressons ici une comparaison du langage UTP et du cadre de représentation ITC implémenté en XML [17, 15].

Les deux approches se distinguent d'abord par la modélisation des programmes (ordonnancements) possibles par classe. Le langage UTP définit chaque classe par une simple séquence de séances de durée égale et le problème consiste à programmer chaque séance. Le schéma ITC procède en extension et associe différents programmes à chaque classe (élément times du schéma). Le problème se ramène alors au choix d'un programme par classe où chaque programme est figé et se définit par la répétition sur plusieurs semaines d'un planning hebdomadaire comportant une ou plusieurs séances de durée égale, placées sur différentes journées et partageant le même créneau quotidien. Les deux représentations ne se réduisent pas l'une à l'autre. Par exemple, UTP ne peut modéliser une classe dont les séances sont de durées variables. A l'inverse, ITC ne peut modéliser une classe programmée sur différents créneaux quotidiens. Toutefois, certains programmes d'intérêt pratique se représentent dans l'une ou l'autre approche en contraignant classes et séances de manière appropriée. Par exemple, une classe hebdomadaire devant se rencontrer sur le même créneau se modélise en combinant des contraintes same_daily_slot, weekly et forbidden_period. L'implémentation d'une méthode de réduction plus complète est en cours d'étude.

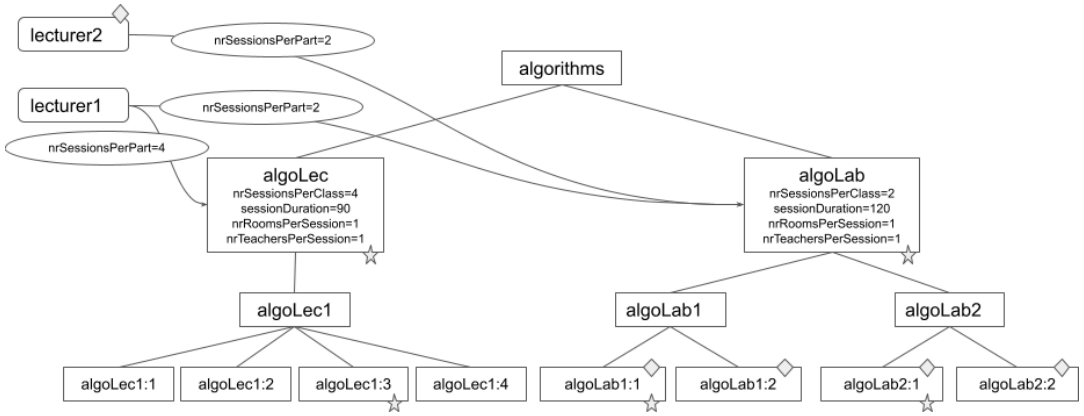
Pour ce qui concerne l'organisation hiérarchique des cours, ITC introduit un niveau intermédiaire modélisant un choix de configuration par cours (élément configuration). Chaque cours possède une ou plusieurs configurations qui sont indépendantes quant à leur décomposition en parties, classes et séances. Le schéma ITC impose simplement qu'un étudiant inscrit à un cours assiste à toutes les parties d'une seule configuration, deux étudiants pouvant être associés à des configurations différentes. Ce concept n'est pas intégré dans la version actuelle du langage UTP. Pour ce qui concerne les ressources, le langage UTP représente explicitement les enseignants à l'instar des salles alors qu'ITC ne modélise que les salles. Il permet aussi d'allouer différentes ressources aux séances d'une même classe alors que le schéma ITC impose qu'une même salle leurs soit allouée. En outre, UTP autorise des séances multi-ressources alors qu'ITC est restreint aux séances mono-salles.

Les deux languages de contraintes se démarquent aussi l'un de l'autre. D'une part, les prédicats ITC s'appliquent aux classes alors que les prédicats UTP s'appliquent à des ensembles de séances quelconques - et en particulier à des séances individuelles - qui peuvent être conditionnés au choix des ressources allouées. D'autre part, le langage de règles et de sélecteurs UTP permet de contraindre n'importe quelle classe de ressources ou d'éléments de cours de manière concise et plus adaptée pour l'expression des besoins.

Enfin, le schéma ITC prescrit une résolution du problème par optimisation combinatoire en intégrant une fonction coût pondérant 4 critères qui pénalisent respectivement les choix de créneaux et de salles pour les classes, les violations de contraintes et le chevauchement de séances par étudiant. Dans sa version actuelle, le langage UTP traite le problème comme un problème de satisfaction de contraintes dures. L'intégration de contraintes souples et la possibilité d'aggréger pénalités ou préférences, que ce soit dans des contextes de construction ou de réparation de solution, est à l'étude.

3 Modèles MiniZinc et CHR

Nous présentons dans cette section deux modèles d'instances UTP développés en MiniZinc et CHR. Ces modèles mettent en jeu des contraintes relatives au partitionnement des étudiants en groupes et à l'attribution des groupes aux classes, à la distribution des ressources sur les séances, à l'ordonnancement des séances, et à l'allocation de leurs




```
sequenced(<(K,_,{3}),(P,algoLec,_)>, <(K,_,{1}),(P,algoLab,_)>)                    (R1)
forbidden_period((<(T,lecturer2,_)>,9120,9240)                                     (R2)
sequenced((algoLec1,{algoLec1:3}), (algoLab1,{algoLab1:1}))                        (C1)
sequenced((algoLec1,{algoLec1:3}), (algoLab2,{algoLab2:1})                         (C2)
forbidden_period((lecturer2,{algoLab1:1,algoLab1:2,algoLab2:1,algoLab2:2}),9120,9240)  (C3)
```

FIGURE 4 – Sélection de séances par des règles

ressources. Nous présentons tout d'abord les données d'instance ainsi que les variables de décision qui sont communes aux deux modèles. La table 2 liste les plages d'entiers identifiant les différents ensembles d'objets manipulés et définit les structures utilisées pour représenter les données d'instance.

3.1 Modèle `MiniZinc`

`MiniZinc` est un langage de modélisation haut-niveau de problèmes d'optimisation sous contraintes [18, 3]. Les modèles `MiniZinc` sont traduits dans le langage cible `Flatzinc` [4] qui permet d'interfacer différents types de solveurs dont les solveurs de programmation par contraintes sur domaines finis tels `Gecode` [2]. `MiniZinc` intègre de nombreuses contraintes globales et le modèle `UTP` présenté en Table 4 et utilisant les variables de décision présentée en Table 3 s'appuie sur quelques contraintes dédiées aux problèmes d'ordonnancement.

Les contraintes de sectionnement répartissent les étudiants dans les groupes et assigne chacun de ces groupes à différentes classes conformément aux règles de sectionnement et aux seuils d'effectifs. La contrainte (1) n'autorise le regroupement d'étudiants que s'ils sont inscrits aux mêmes cours. (2) impose que tout étudiant, assimilé à son groupe, assiste à toute partie de cours dans lequel il est inscrit. (3) assure que les classes d'une partie de cours n'ont aucun groupe en commun. (4) implémente la relation de parenté entre classes. Enfin, (5) vérifie que l'effectif cumulé des groupes attribués à une classe ne dépasse pas le seuil autorisé.

La distribution des ressources s'appuie sur des contraintes de domaine, de cardinalité et de sommes. Les contraintes (6) et (7) définissent les salles et enseignants allouables à chaque séance. (8) contraint le nombre de salles allouées à une séance selon que sa partie de cours est sans salles, mono-salle, ou multi-salles. (9) attribue le nombre attendu d'enseignants à chaque séance et (10) vérifie que chaque enseignant dispense le volume de séances requis par partie de cours où il est pré-positionné.

La programmation des séances et l'allocation des ressources met en jeu des contraintes de positionnement, de séquencement, de non-chevauchement et de capacité. La contrainte (11) définit les créneaux autorisés pour chaque séance. (12) interdit qu'une séance soit à cheval sur 2 journées. (13) séquence les séances d'une classe selon leur rangs. Les contraintes (14) et (15) modélisent les séances multi-salles et l'accès exclusif à leurs ressources. (14) impose qu'une ressource allouée à une séance multi-salles soit disjonctive le temps de son utilisation. (15) assure que le nombre d'étudiants attendus n'excède pas la capacité cumulée des salles allouées. À noter que cette contrainte est purement quantitative et autorise toute répartition d'étudiants dans les salles indépendamment de la structure de groupes. (16) modélise les salles à allouer obligatoirement à toute séance d'une partie de cours. (17) modélise la contrainte de capacité cumulative qui s'applique par défaut à toute salle allouée hors séances multi-salles.

La table 4 présente les variantes de quelques prédicats `UTP` dans le cas où les entités ciblées sont des salles. (18) implémente le prédicat `forbidden_period` qui prend en paramètres les 2 créneaux modélisant la période interdite. (19), (20) et (21) modélisent de manière directe les prédicats `same_weekday`, `same_rooms` et `sequenced`. (22) implémente le prédicat `no_overlap` en s'appuyant sur la contrainte globale `disjunctive`.

3.2 Modèle `CHR`

Constraint Handling Rules (`CHR`) [13, 12] est un langage à base de règles d'inférence à chaînage avant qui remplacent les contraintes du problème par d'autres plus simples jusqu'à la résolution complète. `CHR` est un langage spécialisé

H	ensemble des créneaux définissant l'horizon de temps
C	ensemble des cours
P	ensembles des parties de cours
K	ensembles des classes
S	ensembles des séances
R	ensemble des salles
T	ensemble des enseignants
G	ensemble des groupes d'étudiants
U	ensembles des étudiants

class_{sessions,parents} :
ensemble des séances (resp. classes parentes) d'une classe
part_{classes,lecturers, rooms,sessions} :
ensemble des classes (resp. enseignants, salles, séances) d'une partie
room_sessions :
ensemble des séances possibles pour une salle
session_{part,class} : la partie (resp. classe) d'une séance
student_{courses,parts} :
les cours (resp. parties) que suit un étudiant
mandatory_rooms : les salles obligatoires pour une partie
single_room_sessions : l'ensemble des séances mono-salle
capacity : capacité maximum d'une salle ou d'une classe
is_multi_rooms : indique si une séance est multi-salles
length : durée d'une séance
part_room_use :
modalité d'utilisation de salles pour une partie (none, single, multiple)
rank : rang d'une séance
service :
nombre de séances à dispenser par enseignant par partie
teams :
nombre d'enseignants requis par séance d'une partie
virtual : indique si une salle est de capacité illimitée ou non
dailyslots : créneaux quotidiens autorisés pour une partie
weekdays : journées autorisées pour une partie
weeks : semaines autorisées pour une partie
nr_daily_slots : nombre de créneaux dans une journée
nr_weekly_slots : nombre de créneaux dans une semaine

TABLE 2 – Données d'instances et fonctions utilitaires

permettant de définir des contraintes déclaratives au sens de la programmation logique par contraintes [14, 16]. CHR est une extension de langage qui permet d'introduire des contraintes définies par l'utilisateur, c'est-à-dire des prédicats du premier ordre, dans un langage hôte donné tel que Prolog, Lisp, Java, ou C/C++. Il a ensuite été étendu à $\text{CHR}^{\vee}$ [5] qui introduit le *don't know* [10]. Ce non-déterminisme est offert gratuitement lorsque le langage hôte est Prolog et il permet de spécifier facilement des problèmes de la classe de complexité NP. Pour modéliser et résoudre les instances UTP avec le langage CHR, nous utilisons le solveur CHR++ [8] (pour Constraint Handling Rules in C++), qui est une intégration efficace de CHR dans le langage de programmation C++.

Le modèle CHR est instancié à la lecture de l'instance au format JSON. Le modèle d'entité est d'abord défini, puis les contraintes issues des règles sont déclarées et enfin les domaines des variables sont mis à jour si une partie solution est fourniée, avant de lancer la résolution de l'instance. Le modèle complet pour CHR++ est trop long pour être détaillé ici [1]. Nous donnons dans le tableau 6 la liste des contraintes prises en compte par le solveur. Les variables de décisions devant être instanciées sont données au tableau 5. Elles sont en grande partie similaires à celles du modèle MiniZinc, seules les variables de fin de séances sont ajoutées.

Pour simplifier son implémentation, le modèle CHR est en partie non cumulatif et certaines ressources comme les enseignants ne peuvent se partager. Il considère également que le sectionnement et l'affectation des étudiants aux groupes est fait en amont. Ainsi, calculer une solution revient à trouver une affectation des ressources consistante tout en plaçant les horaires de toutes les séances.

Plusieurs contraintes peuvent être posées dès l'analyse de l'instance. C'est le cas des contraintes (1) à (9) du tableau 6. Les contraintes (2), (3) et (4) filtrent les domaines en retirant les salles, enseignants ou horaires impossibles par construction de l'instance. La contrainte (5) assure qu'une séance commence et se termine le même jour en retirant du domaine les valeurs qui la contredisent.

D'autres contraintes sont posées et gérées par des règles CHR surveillant les modifications des domaines des variables. C'est le cas de la contrainte (1) qui assure l'intégrité des variables de début et de fin de séance. Il en est de même pour (6) qui assure que le nombre d'enseignants demandé pour une session est valide et (7) qui vérifie que le nombre de salles d'une séance correspond bien à ce qui est demandé dans l'instance.

Nous donnons pour exemple la règle CHR++ qui vérifie l'intégrité des variables de début et de fin de séance. Celle-ci est déclenchée dès qu'un domaine d'une variable est mis à jour :

```
session_slot(_, S_Start, S_End, S_Length)
  =>> CP::Int::plus(S_Start, (*S_Length)-1, S_End);;
```

Nous utilisons CHR++ qui permet de manipuler des valeurs associées à des variables logiques et de réveiller les règles correspondantes dès qu'une modification de la valeur survient. Ce mécanisme combiné avec le chaînage avant de CHR nous permet d'implémenter un mécanisme de réveil des règles et de propagation des domaines efficace à la manière d'un solveur CSP.

Les contraintes (8) et (9) ajoutent de nouvelles contraintes CHR au modèle. En effet, les contraintes before et disjunct sont des contraintes CHR assurant la précédence et le non chevauchement de deux séances. Elles sont accompagnées de règles vérifiant la cohérence du graphe disjonctif créé implicitement par l'ajout de toutes ces contraintes. Les prédicats statiques correspondent à ceux lus depuis l'instance. Ils sont traités et des contraintes (contraintes de filtrage, contraintes CHR ou unification de variables) sont ajoutées.

Les contraintes dynamiques de (13) à (18) ne se déclenchent que dans certaines conditions. Des règles CHR

1. Le lecteur intéressé peut télécharger les sources du modèle [1]

array[U] of var G :	x_group	groupe attribué à un étudiant
array[K] of var set of G :	x_groups	ensemble de groupes alloués à une classe
array[S] of var set of R :	x_rooms	ensemble de salles allouées à une séance
array[S] of var set of T :	x_lecturers	ensemble d'enseignants alloués à une séance
array[S] of var H :	x_slot	créneau de départ attribué à une séance

TABLE 3 – Variables de décision (`MiniZinc`)

forall(u, v in U where u<v) (student_courses[u] ! =student_courses[v] -> x_group[u] ! =x_group[v])	(1)
forall(u in U, p in student_parts[u])(exists(k in part_classes[p])(x_group[u] in x_groups[k]))	(2)
forall(p in P, $k1$, $k2$ in part_classes[p] where $k1$<$k2$)(x_groups[$k1$] intersect x_groups[$k2$] = {})	(3)
forall($k1$ in K, $k2$ in class_parents($k1$))(x_groups[$k1$] subset x_groups[$k2$])	(4)
forall(k in K)(maxsize[k]<=sum(g in G)(bool2int(g in x_groups[k]) * sum(u in U)(bool2int(x_group[u] = g)))	(5)
forall(s in S)(x_rooms[s] subset part_rooms[session_part[s]])	(6)
forall(s in S)(x_lecturers[s] subset part_lecturers[session_part[s]])	(7)
forall(s in S, p in P where p = session_part[s])((part_room_use[p] = none -> x_rooms[s] = {}) ∧ (part_room_use[p] = single -> card(x_rooms[s]) = 1) ∧ (part_room_use[p] = multiple -> card(x_rooms[s])>=1))	(8)
forall(s in S)(card(x_lecturers[s]) = team[session_part[s]])	(9)
forall(p in P, l in part_lecturers[p])(sum(s in part_sessions[p]))(bool2int(l in x_lecturers[s]) = service[l, p]))	(10)
forall(p in P, s in part_sessions[p]) (week(x_slot[s]) in weeks[p] ∧ weekday(x_slot[s]) in weekdays[p] ∧ dailyslot(x_slot[s]) in dailyslots[p])	(11)
forall(s in S)((x_slot[s] − 1) div nr_daily_slots = (x_slot[s] + length[s] − 1) div nr_daily_slots)	(12)
forall(k in K, $s1$, $s2$ in class_sessions[k] where rank($s1$)<rank($s2$)) (x_slot[$s1$] + length[s]>=x_slot[$s2$])	(13)
forall(p in P, $s1$ in part_sessions[p], r in part_rooms[p], $s2$ in room_sessions[r] where is_multi_rooms[p] ∧ $s1$! =$s2$) (disjunctive([x_slot[$s1$], x_slot[$s2$]], [bool2int(r in x_rooms[$s1$]) * length[$s1$], bool2int(r in x_rooms[$s2$]) * length[$s2$]]))	(14)
forall(p in P, s in part_sessions[p] where is_multi_rooms[p]) (sum(r in part_rooms[p])(bool2int(r in x_rooms[s]) * capacity[r]) <=sum(g in G)(bool2int(g in x_groups[session_class[s]]) * card(group_students[g])))	(15)
forall(p in P, s in part_sessions[p])(mandatory_rooms[p] subset x_rooms[s])	(16)
forall(r in R where not(virtual[r]))(let {set of S : RS= room_sessions[r] intersect single_room_sessions;} in (cumulative([x_slot[s]|s in RS], [bool2int(r in x_rooms[s]) * length[s]|s in RS], [sum(g in G)(bool2int(g in x_groups[session_class[s]])) * sum(u in U)(bool2int(g = x_group[u]))|s in RS], capacity[r]))	(17)
$forbidden\_period((r, S'), h1, h2)$ = forall(i in S')(r in x_rooms[i] -> (x_slot[i] + length[i]<=h_1 ∨x_slot[i]>h_2))	(18)
$same\_weekday((r, S'))$ = forall(i, j in S' where i<j)((r in x_rooms[i] intersect x_rooms[j]) -> (x_slot[i] div nr_weekly_slots = x_slot[j] div nr_weekly_slots))	(19)
$same\_rooms((r, S'))$ = forall(i, j in S' where i<j)((r in x_rooms[i] intersect x_rooms[j]) -> x_rooms[i] = x_rooms[j])	(20)
$sequenced((r1, S1), (r2, S2))$ = forall(i in $S1$, j in $S2$)(($r1$ in x_rooms[i] ∧ $r2$ in x_rooms[j]) -> x_slot[i]+length[i]<=x_slot[j])	(21)
$no\_overlap((r, S'))$ = disjunctive([x_slot[i]|i in S'], [length[i] * bool2int(r in x_rooms[i])|i in S'])	(22)

TABLE 4 – Contraintes et prédicats du modèle

array[S] of var set of R : x_rooms	ensemble de salles allouées à une séance
array[S] of var set of T : x_lecturers	ensemble d'enseignants alloués à une séance
array[S] of int H : x_slot_start	créneau de départ attribué à une séance
array[S] of int H : x_slot_end	créneau de fin attribué à une séance

TABLE 5 – Variables de décision (`CHR`)

avec garde sont utilisées à cet effet. (13) vérifie qu'un enseignant effectue bien la liste des enseignements auxquels il est inscrit. (14) assure que la capacité des salles est respectée et (15) vérifie que les salles marquées comme obligatoires se retrouvent bien dans la solution. Le prédicat (16) assure que des séances associées au même prédicat $same\_weekday$ sont fixées sur le même jour de la semaine.

Les contraintes (17) et (18) ajoutent des contraintes `CHR disjunct` lorsque certaines conditions sont vérifiées. Ainsi, (17) pose un `disjunct` entre deux séances lorsqu'un même enseignant y participe. (18) ajoute une contrainte `disjunct` entre deux séances si celles-ci ont lieu dans la même salle. Ces contraintes `CHR` viennent enrichir le graphe disjonctif représentant le séquencement de

Contrainte d'intégrité :

$\forall s \in S$: x_slot_end$[s]$ = x_slot_start$[s]$ + length(s) (1)

Contraintes statiques (filtrage des entrées de l'instance) :

$\forall s \in S$: x_rooms$[s] \subseteq$ part_rooms[session_part(s)] (2)

$\forall s \in S$: x_lecturers$[s] \subseteq$ part_lecturers[session_part(s)] (3)

$\forall p \in P, \forall s \in$ part_sessions(p) : $\big($week(x_slot_start$[s]) \in$ weeks$[p]\big)$
$\wedge\big($weekday(x_slot_start$[s]) \in$ days$[p]\big) \wedge \big($dailyslot(x_slot_start$[s]) \in$ dailyslots$[p]\big)$ (4)

$\forall s \in S$: x_slot_start$[s]/nr_daily_slots =$ x_slot_end$[s]/nr_daily_slots$ (5)

$\forall s \in S$: card(x_lecturers$[s]$) = team[part_sessions$[s]$] (6)

$\forall k \in K, \forall s \in$ class_sessions$[k]$:
Si $\big($part_room_use[class_part$(k)]$ = none$\big)$ alors card(x_rooms$[s]$) = 0
Si $\big($part_room_use[class_part$(k)]$ = single$\big)$ alors card(x_rooms$[s]$) = 1
Si $\big($part_room_use[class_part$(k)]$ = multiple$\big)$ alors card(x_rooms$[s]$) ≥ 1 (7)

$\forall k \in K, \forall s, s' \in$ class_sessions$[k], s.t.$ rank$(s) <$ rank(s') : before(s, s') (8)

$\forall k_1, k_2 \in K, s.t. \exists g_1 \in$ class_groups$[k_1], \exists g_2 \in$ class_groups$[k_2],$ avec $g_1 = g_2$:
$\forall s_1 \in$ class_sessions$(k_1), s_2 \in$ class_sessions(k_2) : disjunct(s_1, s_2) (9)

Prédicats statiques :

$forbidden\_period((e, S'), h, h') = \forall i \in S'$: (x_slot_start$[i] < h) \wedge$ (x_slot_start$[i] > h')$ (10)

$sequenced((e_1, S_1), (e_2, S_2)) = \forall i_1 \in S_1, \forall i_2 \in S_2$: before(i_1, i_2) (11)

$same\_rooms((e, S')) = \forall s_1, s_2 \in S', s.t.\ s_1 < s_2$: x_rooms$[s_1] \sim$ x_rooms$[s_2]$ (12)

Contraintes dynamiques :

$\forall p \in P, \forall l \in$ part_lecturers$[p]$: $\big|\big|\{x \mid x \in$ part_sessions$(p), l \in$ x_lecturers$[x]\}\big|\big| =$ service$[p, l]$ (13)

$\forall s \in S, \forall r \in$ session_rooms(s) :
$\sum\{$group_students$[g] \mid g \in$ x_groups[session_class$[s]], r \in$ x_rooms$[s]\} \leq$ room_capacity$[r]$ (14)

$\forall s \in S$: mandatory_rooms[session_part$[s]] \subseteq$ x_rooms$[s]$ (15)

Prédicat dynamique :

$same\_weekday((e, S')) =$
$\forall s_1, s_2 \in S', s.t.\ s_1 < s_2$: x_slot_start$[s_1]/nr_weekly_slots =$ x_slot_start$[s_2]/nr_weekly_slots$ (16)

Contraintes introspectives :

$\forall k_1, k_2 \in K, \forall s_1 \in$ class_sessions$[k_1], \forall s_2 \in$ class_sessions$[k_2],$ s.t. $s_1 \neq s_2$:
x_lecturers$[s_1] \cap$ x_lecturers$[s_2] \neq \emptyset \Rightarrow$ disjunct(s_1, s_2) (17)

$\forall k_1, k_2 \in K, \forall s_1 \in$ class_sessions$[k_1], \forall s_2 \in$ class_sessions$[k_2]$ s.t. $s_1 \neq s_2$:
x_rooms$[s_1] \cap$ x_rooms$[s_2] \neq \emptyset \Rightarrow$ disjunct(s_1, s_2) (18)

TABLE 6 – Contraintes et prédicats du modèle CHR

toutes les séances.

Il est à noter que le modèle CHR effectue du filtrage de domaine mais analyse également le graphe disjonctif afin d'éliminer des non solutions. Les arêtes du graphe disjonctif sont orientées au fur et à mesure de l'avancement de la résolution et de l'instanciation des variables de décision.

4 Expérimentations

Nous avons mené des expérimentations sur une instance réelle modélisant le second semestre de la Licence 3 d'informatique à l'Université d'Angers. L'instance est disponible aux formats XML, JSON et DZN sur le site [1].

L'instance comporte 5 cours communs à tous les étudiants et 2 choix d'options, chacun portant sur 2 cours, soit un total de 7 cours suivis par les étudiants sur les 9. L'instance compte 24 parties de cours et 42 classes. Les séances sont à programmer sur un horizon de 12 semaines de 5 jours chacune où chaque journée se divise en créneaux de 1 minute. Les séances doivent être placées sur une grille horaire qui commence à 08:00, se termine à 19:50 et est composée de plages d'1h20 espacées de 10 minutes. Une séance qui dure 1 plage a donc une durée de 80 créneaux et a 8 créneaux de départ possibles. Certaines séances durent 2 plages, et ont donc une durée de 170 créneaux avec 7 créneaux de départ possibles. Dans le cas où une séance dure 2h (120 créneaux), la séance doit commencer ou se terminer pour s'aligner sur la grille, soit 13 créneaux de départ possibles. L'instance est constituée de 67 étudiants prédivisés en 4 groupes, 12 enseignants et 8 salles. Elle intègre 46 règles dont une majorité de règles coordonnant les séances (parallélisation entre classes de travaux pratiques ou options, séquencement entre cours magistraux, travaux dirigés et travaux pratiques, etc.) et quelques règles restreignant salles et enseignants possibles selon les cours.

Les solveurs MiniZinc et CHR++ présentés en section 3 ont été utilisés pour résoudre cette instance avec une architecture Intel Core i7-10875H 2.30GHz et la résolvent en moins de 5s (hors flattening pour MiniZinc).

La stratégie de résolution employée dans le modèle MiniZinc consiste d'abord à allouer les salles, puis les enseignants avant de placer les séances sur l'horizon de temps. Les variables d'allocation sont ordonnées par l'heuristique first_fail et leurs domaines de valeurs explorés de manière systématique. Les variables de choix de créneaux par séance utilisent aussi l'heuristique first_fail et l'heuristique de choix de valeurs consiste à scinder chaque domaine (indomain_split). Gecode

est le solveur utilisé avec MiniZinc dans nos tests. À noter que les contraintes disjonctives y sont implémentées comme un cas particulier de la contrainte globale cumulative présentée dans [9].

La stratégie de résolution employée avec CHR++ consiste à instancier les variables de décisions en commencant par le tableau de variables x_rooms, puis x_lecturers et enfin x_slot_start (les autres variables sont déduites par propagation). Dans chaque tableau, la prochaine variable à instancier est choisie selon l'ordre de définition dans le tableau et la valeur testée est toujours la plus petite valeur possible du domaine. Entre chaque instanciation, une phase de propagation des contraintes (filtrage des domaines et analyse du graphe disjonctif) est itérée jusqu'à l'obtention d'un point fixe. En cas d'échec, la méthode revient sur son choix précédent pour essayer l'alternative suivante. Il n'y a pour l'instant aucune heuristique spécialisée, mais le choix de la plus petite valeur du domaine semble pertinent. En effet, pour construire un emploi du temps, il est naturel de commencer à fixer les séances en partant du début de l'horizon de temps.

5 Conclusion et perspectives

Dans cet article, nous avons introduit brièvement le langage UTP qui permet de modéliser la problématique de construction d'emplois du temps universitaires. Le langage est générique et permet de s'adapter à diférentes variantes du problème UTP. Par exemple, les ressources sont cumulatives par défaut mais des règles peuvent être surimposées pour rendre certaines ressources disjonctives. De plus, le langage s'appuie sur un catalogue de prédicats qui peut être enrichi afin de s'adapter aux spécificités de différents environnements, et ce sans modifier le langage lui-même.

Dans sa version actuelle, le langage UTP réduit la génération d'emploi du temps à un problème de satisfaction de contraintes dur et ne prend en compte aucun critère d'optimisation. Nous avons pour objectif de développer cet aspect, afin entre autres de pouvoir exprimer des préférences, et définir des méthodes d'évaluation d'une solution pour pouvoir choisir la plus adaptée aux souhaits des décideurs (p. ex. éviter de trop longues interruptions de cours dans une journée pour les étudiants ou regrouper les cours sur des demi-journées pour les enseignants).

Nous avons détaillé également deux modèles de programmation par contraintes implémentés en MiniZinc et CHR++. Ces deux modèles ont été développés comme preuve de concept et seront améliorés notamment en implémentant des stratégies de résolution facilitant le passage à l'échelle sur des instances plus conséquentes.

Remerciements

Ce travail est partiellement financé par le projet Thélème octroyé aux universités d'Angers et du Mans dans le cadre du PIA3.

Références

[1] *University Service Planning (https://ua-usp.github.io/timetabling/)*.

[2] *Generic Constraint Development Environment (https://www.gecode.org/)*, 2022.

[3] *Minizinc (https://www.minizinc.org/)*, 2022.

[4] *Specification of Flatzinc. Version 1.6 (https://www.minizinc.org/downloads/doc-1.6/flatzinc-spec.pdf)*, 2022.

[5] S. Abdennadher and H. Schütz. CHR : A Flexible Query Language. In *FAQS 1998*, pages 1–14, 1998.

[6] A. Nurul Liyana Abdul and A. Nur Aidya Hanum. A brief review on the features of university course timetabling problem. *AIP Conference Proceedings*, 2016(1) :020001, 2018.

[7] V. Barichard, C. Behuet, D. Genest, M. Legeay, and D. Lesaint. A constraint language for university timetabling problems. Submitted, 2022.

[8] V. Barichard and I. Stéphan. Quantified constraint handling rules. In *ICLP 2019*, volume 306, pages 210–223, Las Cruces, 20-25/09/2019 2019.

[9] N. Beldiceanu and M. Carlsson. A new multi-resource cumulatives constraint with negative heights. In *CP 2002*, pages 63–79, 2002.

[10] H. Betz and T.W. Frühwirth. Linear-logic based analysis of constraint handling rules with disjunction. *ACM Transactions on Computational Logic*, 14(1), 2013.

[11] T.W. Frühwirth. Constraint Handling Rules. In *Constraint Programming : Basics and Trends*, pages 90–107, 1994.

[12] T.W. Frühwirth. *Constraint Handling Rules*. Cambridge University Press, 2009.

[13] T.W. Frühwirth. Theory and practice of constraint handling rules. *Journal of Logic Programming*, 37(1-3) :95–138, 1998.

[14] P. Van Hentenryck. Constraint logic programming. *Knowledge Engineering Review*, 6(3) :151–194, 1991.

[15] ITC19. *International Timetabling Competition (https://www.itc2019.org/)*, 2019.

[16] J. Jaffar and M.J. Maher. Constraint logic programming : A survey. *Journal of Logic Programming*, 19/20 :503–581, 1994.

[17] T. Müller, H. Rudová, and Z. Müllerová. University course timetabling and International Timetabling Competition 2019. In *PATAT-2018*, pages 5–31, 2018.

[18] N. Nethercote, P.J. Stuckey, R. Becket, S. Brand, G.J. Duck, and G. Tack. Minizinc : Towards a standard cp modelling language. In *CP 2007*, pages 529–543, 2007.

Session 6 : Résolution de problèmes

Diagrammes de décision binaires optimaux par satisfiabilité Booléenne maximale (Max-SAT) pour la classification

Hao Hu, Marie-José Huguet, Mohamed Siala

LAAS-CNRS, Université de Toulouse, CNRS, INSA, Toulouse, France
{hhu, huguet, siala}@laas.fr

Résumé

L'intérêt croissant pour les systèmes d'intelligence artificielle de confiance motive le besoin de modèles d'apprentissage interprétables tels que des arbres de décision ou des règles de décision. En effet, en raison de leur structure, ces modèles sont intrinsèquement compréhensibles par l'homme, en particulier lorsqu'ils sont de petite taille. Dans cet article, nous nous intéressons aux diagrammes de décision binaire (BDD), qui malgré leur qualité en termes de représentation compacte de fonctions Booléennes, n'ont été que peu étudiés en tant que modèle d'apprentissage interprétable. Tout d'abord, nous proposons des modèles basés sur la Satisfiabilité Booléenne (SAT) pour déterminer s'il existe un BDD avec une hauteur fixée, permettant de classifier correctement tous les exemples d'un jeu de données. Ensuite, nous considérons un modèle MaxSAT pour déterminer des BDD optimaux maximisant le nombre d'exemples correctement classifiés, pour une hauteur fixée. Puis, nous introduisons une méthode permettant de fusionner des sous-arbres compatibles générés dans les BDD pour résoudrons le probleme de fragmentation. Les expérimentations mettent en évidence les avantages de notre modèle MaxSAT comparé avec des approches exactes ou heuristiques de l'état de l'art. Ce travail a été présenté à la conférence AAAI-2022 [1].

Mots-clés

Diagramme de décision binaire, Satisfiabilité Booléenne Maximale, Classification

1 Introduction

Dans le contexte de l'IA de confiance, les méthodes d'optimisation combinatoire, telles que la programmation par contraintes, la programmation linéaire en nombres entiers ou la satisfiabilité Booléenne ont été utilisées avec succès pour apprendre des modèles interprétables tels que des arbres de décision, des règles de décision ou des ensembles de décision. Ces approches déclaratives sont particulièrement intéressantes car elles offrent une certaine flexibilité notamment pour intégrer des contraintes additionnelles lors de l'apprentissage d'un modèle.

Les *Diagrammes de Decision Binaires* (BDD) permettent de représenter de façon compacte des fonctions Booléennes et peuvent être exploités comme modèles d'apprentissage interprétables. En comparaison avec les *Arbres de Decision* (DT), les BDD pouvent réduire les problèmes de *réplication* (reproduction de sous-arbres) et de *fragmentation* (peu d'exemples associés aux feuilles).

Soit g une fonction booleenne définie sur une sequence de variables booleennes $\mathcal{X}$. Le BDD associé à g est un graphe acyclique enraciné. Il contient deux types de noeuds : des noeuds internes associés aux variables booleennes $\mathcal{X}$ et ayant deux fils, et deux noeuds *terminaux*, représentant chacun une valeur binaire ($\{0, 1\}$). La hauteur d'un BDD correspond au nombre de niveaux dans le graphe acyclique. Pour garantir l'unicité du BDD associé à une fonction booleenne donnée, deux propriétés sont considérées : BDD *ordonné* et *reduit*. Dans un BDD ordonné, pour tous les chemins de la racine vers les noeuds terminaux, toutes les variables suivent l'ordre donné par la séquence de variables $\mathcal{X}$. Un BDD est *reduit* s'il n'y a aucune duplication de sous-arbre, et si aucun noeud interne n'a les même fils.

Une fonction Booleenne g peut être représentée par sa *table de vérité*, qui est une chaine de valeurs binaires représentant toutes les affectations possibles des variables booleennes. Nous détaillons dans l'article comment produire un diagramme de décision binaire par l'étude de sa table de vérité.

2 Problèmes de classification étudiés

Nous nous intéressons à deux problèmes de classification. Tout d'abord, nous considérons un problème de décision permettant d'obtenir des BDD de hauteur donnée classifiant correctement tous les exemples. Ce problème est le suivant :

- $\boldsymbol{P_{bdd}(\mathcal{E}, H)}$: Soit un jeu de données $\mathcal{E}$, est-ce qu'il existe un diagramme de decision binaire de hauteur H, pouvant classifier correctement tous les exemples de $\mathcal{E}$?

En utilisant une recherche linéaire, le probleme $P_{bdd}(\mathcal{E}, H)$ peut être utilisé pour déterminer le BDD *de plus petite hauteur* qui classifie correctement tous les exemples de $\mathcal{E}$.

Le second problème étudié est un problème d'optimisation visant à obtenir un BDD ayant la meilleure prédiction. Il est défini comme suit :

- $\boldsymbol{P^*_{bdd}(\mathcal{E}, H)}$: Soit un jeu de données $\mathcal{E}$, trouver un diagramme de décision binaire de hauteur H, maximisant le nombre d'exemples bien classifiés.

Pour le problème de décision $P_{bdd}(\mathcal{E}, H)$, nous proposons deux modèles de satisfiabilité Booléenne (SAT) : un mo-

dèle initial et un modèle amélioré permettant de réduire la taille de l'encodage.Le probleme d'optimation $P^*_{bdd}(\mathcal{E}, H)$ est ensuite modélisé par MaxSAT en adaptant le modèle de plus petite taille.

3 Modèles SAT et MaxSAT

Modèle SAT pour $P_{bdd}(\mathcal{E}, H)$. Nous nous appuyons sur l'idée qu'il est possible de produire un diagramme de décision binaire à partir des caractéristiques de sa table de vérité et que cette table implique une séquence de variables booleennes. Pour réaliser la classification des exemples d'un jeu de données, nous devons trouver une séquence des attributs binaires, qui soit de taille H et corresponde à une bijection vers la séquence des variables booleennes. Nous devons de plus déterminer une table de vérité qui classifie correctement tous les exemples. La séquence des attributs binaires est appelée *séquence des features*. Le modèle SAT contient deux parties :

1. Construction de la séquence des features (de taille H) sélectionnés parmi les attributs du jeu de données pour respecter la structure d'un BDD.
2. Génération de la table de vérité, à partir de la séquence des features, pour classifier correctement tous les exemples.

Les contraintes suivantes modélisent la partie 1 :

- Chaque attribut du jeu de données ne peut être sélectionné au plus qu'une seule fois.
- Chaque indice de la séquence des features contient exactement un attribut.
- La première moitié de la table de vérité doit être différente de la dernière moitié (pour éviter que le premier attribut choisi fasse une division inutile du jeu de données).

Pour les contraintes de la partie 2, le modèle SAT initial est le suivant : pour chaque exemple positif (resp. négatif) de $\mathcal{E}$, nous utilisons 2^H contraintes pour vérifier que cet exemple n'arrive pas à une valeur négative (resp. positive) en suivant son affectation par la table de vérité.

La taille de l'encodage de ce modèle SAT initial est fournie par la *Proposition* 2 de l'article.

Pour diminuer cette taille, nous proposons une version améliorée pour les contraintes de la partie 2 : pour chaque exemple de $\mathcal{E}$, nous utilisons 2^H contraintes pour assurer que cet exemple arrive à une valeur avec la bonne classe en suivant son affectation par la table de vérité.

La taille de l'encodage de ce modèle SAT amélioré est fournie par la *Proposition* 3 de l'article qui montre son avantage par rapport au modèle initial.

Modele MaxSAT pour $P^*_{bdd}(\mathcal{E}, H)$. Pour résoudre le probleme d'optimation, nous transformons *le modele SAT amélioré* vers un modèle MaxSAT. Les contraintes de la partie 1 sur la structure du BDD sont considérées comme des contraintes "dures" (hard clauses). Et les contraintes de la partie 2, sur la classification des exemples, sont considérées comme des contraintes "souples" (soft clauses).

Pour un exemple de $\mathcal{E}$, le nombre de soft clauses satisfaites vaut : soit 2^H (exemple bien classifié), soit $2^H - 1$ (exemple mal classifié). Ainsi l'objectif de maximiser le nombre de soft clauses est équivalent à maximiser le nombre d'exemples bien classifiés.

Fusion des sous-arbres compatibles. Pour éviter le problème de fragmentation, nous proposons un post-traitement, basé sur une méthode de la littérature, permettant de fusionner les sous arbres compatibles. Pour cela, les noeuds feuilles ne contenant aucun exemple sont marqués comme "*unknown*". Deux sous-arbres sont dits compatibles lorsque soit au moins une racine est marquée "*unknown*", soit les deux racines contiennent le même attribut, et leurs fils respectifs sont racines de sous arbres compatibles. Fusionner des sous arbres compatibles conduit à introduire des biais de prédiction en attribuant des classes à des noeuds "*unknown*".

4 Résultats expérimentaux

Pour évaluer la performance du modèle MaxSAT, nous le comparons avec des approches heuristiques et exactes de l'état de l'art.

Par rapport à des approches heuristiques, les résultats expérimentaux montrent que le modèle MaxSAT pour le problème d'optimisation considéré, obtient de meilleures performances en prédiction pour une même hauteur de BDD.

Dans une seconde expérimentation, nous comparons le modèle MaxSAT pour l'obtention de BDD optimaux avec un modèle MaxSAT pour apprendre des arbres de décision optimaux. La première observation est que les deux modèles sont proches en termes de prédiction. Une seconde observation est que les diagrammes de décision obtenus par le modèle MaxSAT proposé sont toujours de plus petites dimensions (en termes de nombre de noeuds) par rapport aux arbres de décision. La troisième observation porte sur le modèle MaxSAT, celui proposé pour les BDD conduit à un encodage de plus petite taille par rapport au modèle MaxSAT pour les arbres de décision.

Finalement, pour améliorer le passage à l'échelle de notre modèle MaxSAT pour les BDD, nous en proposons une version heuristique. L'idée est de sélectionner la séquence des features parmi un ensemble réduit d'attributs. Pour obtenir cet ensemble réduit d'attributs, dans nos expérimentations la méthode CART est utilisée en pré-traitement du modèle MaxSAT. Les résultats obtenus indiquent que la version heuristique proposée est compétitive en prédiction par rapport au modèle MaxSAT précédent. De plus, la taille du modèle MaxSAT obtenu à partir de cette version heuristique est grandement réduite par rapport à la version exacte portant sur l'ensemble des attributs du jeu de données. Cet avantage peut permettre de considérer des jeux de données de plus grandes dimensions.

Références

[1] Hao Hu, Marie-José Huguet et Mohamed Siala : Optimizing Binary Decision Diagrams with MaxSAT for classification. *arXiv preprint arXiv :2203.11386*(2022).

Isomorphismes entre instances et sous-instances STRIPS

Martin C. Cooper, Arnaud Lequen [*], Frédéric Maris

IRIT, Université de Toulouse, France

{Martin.Cooper, Arnaud.Lequen, Frederic.Maris}@irit.fr

Résumé

Dans le domaine de la planification automatique, décider si deux instances encodées en STRIPS sont isomorphes est la manière la plus élémentaire de comparer deux instances. C'est aussi un cas particulier du problème où l'on se donne deux instances P et P', et l'on cherche un isomorphisme entre P et une sous-instance de P'. Dans cet article, nous nous proposons d'étudier la complexité de ces deux problèmes. On montre que le premier est GI-complet, tandis que le second est NP-complet. Malgré cela, nous proposons un algorithme qui permet de construire un tel isomorphisme, lorsqu'il existe. De même, nous montrons expérimentalement que, sur nos jeux de tests, un pré-traitement basé sur des méthodes de propagation de contraintes permet d'améliorer significativement l'efficacité du solver SAT utilisé par notre algorithme.

Mots-clés

Planification automatique, isomorphisme, complexité, propagation de contraintes

Abstract

Determining whether two STRIPS planning instances are isomorphic is the simplest form of comparison between planning instances. It is also a particular case of the problem concerned with finding an isomorphism between a planning instance P and a sub-instance of another instance P'. In this paper, we study the complexity of both problems. We show that the former is GI-complete, and we prove the latter to be NP-complete. Nonethelss, we propose an algorithm to build an isomorphism, when possible. We report experimental trials on benchmark problems which demonstrate that applying constraint propagation in preprocessing can greatly improve the efficiency of a SAT solver.

Keywords

Planning, isomorphism, complexity, constraint propagation

1 Introduction

Les modèles de planification STRIPS [7] encodent implicitement de grands espaces d'états, souvent impossible à représenter explicitement, mais qui ont cependant une structure claire et régulière. Il est donc raisonnable de penser que deux instances de planification différentes puissent avoir une partie de leurs structures respectives en commun. Cependant, cette similarité n'est pas immédiate à identifier. En effet, pour déterminer si une instance P est une sous-instance d'une autre instance P', il faut associer chaque fluent et chaque action de P à son homologue dans P', tout en respectant une propriété de morphisme. Cela requiert l'exploration de l'espace des fonctions de P vers P', de taille exponentielle en les représentations de ces instances. Il reste que trouver un tel morphisme permet de transférer à une instance certaines informations dont l'on dispose sur l'autre. Par exemple, tout plan-solution de P peut être transformé en un plan-solution pour P' efficacement.

En programmation par contraintes, il est monnaie courante de stocker hors-ligne toutes les solutions d'une instance CSP ou SAT en une forme compilée [1]. Une carte de compilation indique quelles opérations peuvent être effectuées en temps polynomial lors de l'exécution en ligne [6]. Or, il est bien connu qu'une instance de planification STRIPS à horizon fixe peut être compilée en une instance SAT, grâce à l'encodage SATPLAN [9]. Ainsi, pour une instance donnée, tous les plans peuvent être stockés en une forme compilée, du moins en théorie – en pratique, la forme compilée est de taille trop importante pour être gardée en mémoire. Les instances de planification les plus favorables à cette approche par compilation sont celles pour lesquelles le nombre de plans-solutions est faible, ou, au contraire, pour lesquelles l'ordre des actions est peu contraint. Dans le cas où l'on a déjà calculé une forme compilée C' représentant tous les plans-solutions pour P', et que l'on a un problème P' similaire à P, on peut se demander si l'on peut synthétiser un plan pour P à partir de C'. Si P est isomorphe à un sous-problème de P', il suffit alors d'appliquer une suite d'opérations de conditionnement à C' pour obtenir une forme compilée C représentant toutes les solutions de P. C'est dans cette optique-là que l'on cherche à étudier des isomorphismes entre sous-problèmes. Un cas simple, mais crucial, survient lorsque P n'admet aucune solution. Alors, un isomorphisme entre P et un sous-problème de P' est une preuve que P' n'admet aucune solution.

Dans cet article, nous commençons par l'étude du problème SI, qui traite de la synthèse d'un isomorphisme entre deux instances STRIPS de tailles identiques. Notre preuve de la GI-complétude du problème nous permet d'affirmer l'existence d'un algorithme quasi-polynomial pour SI [2]. Nous nous penchons ensuite sur le problème SSI, qui cherche à synthétiser un isomorphisme entre une instance STRIPS et

[*] Doctorant

une sous-instance d'une autre instance STRIPS. Nous appelons une telle fonction un *isomorphisme de sous-instance*. Après avoir montré que ce problème est NP-complet, nous proposons un algorithme permettant de déterminer l'existence d'un isomorphisme de sous-instance, ou qui détecte qu'aucun tel morphisme n'existe. Notre algorithme est basé sur une réduction à SAT, renforcée par des techniques de propagation de contraintes, qui nous permettent d'éliminer des associations non-cohérentes entre éléments de P et P'.
Nous avons, jusque-là, considéré que les instances de planification P et P' possédaient des états initiaux et finaux identiques, à isomorphisme près. Même lorsque cette condition est relaxée, un isomorphisme entre P et une sous-instance de P' peut tout de même être utile. Par exemple, si π est un plan-solution pour P, son image dans P' peut alors être convertie en une nouvelle action qui peut être ajoutée à P' afin de faciliter sa résolution. Nous proposons alors cette version plus faible d'isomorphisme de sous-instance, que nous appelons *isomorphisme homogène de sous-instance*. Le problème correspondant est noté SSI-H.
Précédemment, d'autres travaux ont étudié la complexité de nombreux problèmes associés à la synthèse de plans-solutions pour des instances STRIPS [4], ou se sont penchés sur la complexité de domaines particuliers de planification [8]. Plus rarement, on peut aussi faire état de travaux portant sur la modification d'instances de planification, à l'image du problème traitant de l'adaptation d'une instance afin de rendre faisable un plan donné en entrée [10].
Notre article est organisé comme suit : la Partie 2 présente des notations générales, ainsi que des concepts et constructions que nous utilisons au long de l'article. Dans la Partie 3 et dans la Partie 4, nous présentons nos résultats théoriques pour SI et SSI, respectivement. Dans la Partie 5, nous présentons notre algorithme pour SSI. Enfin, la Partie 6 résume nos résultats expérimentaux, suivis d'une discussion.

2 Préliminaires

2.1 Planification automatique

Une instance de planification STRIPS est un 4-uplet $P = \langle F, I, O, G\rangle$ tel que F est un ensemble d'éléments appelés *fluents* (des variables propositionnelles dont la valeur peut changer avec le temps), I et G sont des ensembles de littéraux de F, appelés respectivement l'*état initial* et le *but*, et O est un ensemble d'*actions*. Une action est de la forme $o = \langle \mathsf{pre}(o), \mathsf{eff}(o)\rangle$, où $\mathsf{pre}(o)$ et $\mathsf{eff}(o)$ sont la *préconditions* et l'*effet* de o, des ensembles de littéraux de F.
On note $\mathsf{pre}^+(o) = \{f \in F \mid f \in \mathsf{pre}(o)\}$ les fluents positifs de $\mathsf{pre}(o)$, et $\mathsf{pre}^-(o) = \{f \in F \mid \neg f \in \mathsf{pre}(o)\}$ les fluents négatifs. De même, on note $\mathsf{eff}^+(o) = \{f \in F \mid f \in \mathsf{eff}(o)\}$ et $\mathsf{eff}^-(o) = \{f \in F \mid \neg f \in \mathsf{eff}(o)\}$.
Par abus de notation, on note $\mathsf{pre} : O \to 2^F \cup 2^{\neg F}$ la fonction $o \mapsto \mathsf{pre}(o)$, et on utilise des notations similaires pour $\mathsf{pre}^+, \mathsf{pre}^-, \mathsf{eff}, \mathsf{eff}^+$, and eff^-. Dans cet article, on notera $\mathcal{C} = \{\mathsf{pre}^+, \mathsf{pre}^-, \mathsf{eff}^+, \mathsf{eff}^-\}$, et, pour un ensemble quelconque S de littéraux de F, $\neg S = \{\neg l \mid l \in S\}$.
Un état s est une assignation de chaque fluent de F à une valeur de vérité. Nous associerions ici s avec l'*ensemble* des littéraux de F vrais dans s. Etant donnée une instance $P = \langle F, I, O, G\rangle$, un plan $o_1, \ldots, o_k$ de O est une suite d'états $s_0, \ldots, s_k$ satisfaisant, pour $i \in [\![1; k]\!]$, $s_i = (s_{i-1} \setminus \mathsf{eff}^-(o_i)) \cup \mathsf{eff}^+(o_i)$, et $\mathsf{pre}^+(o_i) \subseteq s_{i-1}$, $\mathsf{pre}^-(o_i) \cap s_{i-1} = \varnothing$. Un plan-solution est alors un plan tel que $s_0 = I$ et $G \subseteq s_k$.

2.2 La classe de complexité GI

Dans cette section, nous présentons la classe de complexité GI, pour laquelle SI est complet, comme nous le montrerons dans une section ultérieure. GI est construite autour du problème d'isomorphisme de graphes, qui consiste à trouver une bijection $u : V \to V'$ entre les sommets de deux graphes $\mathcal{G}(V, E)$ et $\mathcal{G}'(V', E')$, telle que les images des sommets reliés entre eux par une arrête dans $\mathcal{G}$ sont aussi reliés dans $\mathcal{G}'$, et réciproquement. Plus formellement :

$$\{x, y\} \in E \text{ ssi } \{u(x), u(y)\} \in E' \tag{1}$$

Définition 1. *La classe de complexité GI est la classe des problèmes pour lesquels il existe une réduction de Turing en temps polynomial au problème d'isomorphisme de graphes.*

La classe de complexité GI contient un grand nombre de problèmes traitant de l'existence d'un isomorphisme entre deux structures non-triviales encodées de manière explicite. De tels problèmes sont souvent complets pour GI, comme par exemple les problèmes d'isomorphisme entre graphes colorés, entre hypergraphes, entre automates [14], etc. En particulier, nous utiliserons le résultat suivant :

Proposition 1 ([14], Ch. 4, Sec. 15)**.** *Le problème d'isomorphisme entre graphes orientés est GI-complet.*

Comme pour le problème d'isomorphisme de graphes non-orientés, un isomorphisme entre deux graphes orientés $\mathcal{G}(V, E)$ et $\mathcal{G}'(V', E')$ est une bijection $u : V \to V'$ telle que $(x, y) \in E$ ssi $((u(x), u(y)) \in E'$.
Dans cet article, nous considérons aussi une autre catégorie de structures, appelées Modèles Finis. Le problème d'isomorphisme entre modèles finis est aussi connu comme étant GI-complet [14].

Définition 2. *Un Modèle Fini est un $(n+1)$-uplet $M = \langle V, R_1, \ldots, R_n\rangle$ où V est un ensemble fini et non vide, et où chaque R_i est une relation sur les éléments de V avec un nombre d'arguments fini.*

Soient $M = \langle V, R_1, \ldots, R_n\rangle$ et $M' = \langle V', R'_1, \ldots, R'_n\rangle$ des modèles finis. Un isomorphisme entre M et M' est une bijection $u : V \to V'$ telle que, pour tout $i \in \{1, \ldots, n\}$, et pour tout ensemble d'éléments $v_1, \ldots, v_m$ (avec m l'arité de R_i), on a $R_i(v_1, \ldots, v_m)$ ssi $R'_i(u(v_1), \ldots, u(v_m))$.

Proposition 2 ([14], Ch. 4, Sec. 15)**.** *Le problème d'isomorphisme entre modèles finis est GI-complet.*

En l'état des connaissances actuelles, le consensus est que GI est une classe intermédiaire entre P et NP. Plus précisément, le problème d'isomorphisme de graphes peut être résolu en temps quasi-polynomial [2]. Il est donc très peu vraisemblable que le problème soit NP-difficile, malgré le fait qu'aucun algorithme polynomial n'est encore connu.

2.3 Encodages de graphes vers STRIPS

Nous présentons ici deux méthodes pour encoder un graphe $\mathcal{G} = (V, E)$ en une instance de planification $P = \langle F, I, O, G\rangle$. Ces constructions sont nécessaires à divers endroits dans la suite de cet article, et ne diffèrent que par leur prise en compte, ou non, de l'orientation des arêtes de $\mathcal{G}$. Elles modélisent un agent se déplaçant sur ce graphe, allant de sommet en sommet en passant par les arêtes. Ainsi, un agent se trouvant au sommet v serait dans un état représenté par $\{v\}$, où tous les autres fluents autres que v sont faux. Afin de simplifier la lecture, pour toute arête $(v_s, v_t) \in F^2$, on note $\mathsf{move}(v_s, v_t)$ l'action qui représente un déplacement du sommet v_s au sommet v_t. Puisque $F = V$, on a :

$$\mathsf{move}(v_s, v_t) = \langle \{v_s\} \cup \neg(V \setminus \{v_s\}), \{v_t\} \cup \neg(V \setminus \{v_t\})\rangle$$

Dans la construction suivante, les sommets (resp. arêtes) de $\mathcal{G}$ sont en bijection avec les fluents (resp. actions) de P, et on exclut les graphes ayant des multi-arêtes. A noter que d'autres constructions pour move auraient pu être utilisées, pour peu qu'elles encodent chaque arête de manière unique. De plus, ces constructions suffisent pour l'usage théorique que l'on en fait, bien qu'elles encodent des instances de planifications triviales et déjà résolues.

Construction 1. *Soit $\mathcal{G} = (V, E)$ un graphe orienté. On construit l'instance de planification $P_\mathcal{G} = \langle F, I, O, G\rangle$:*

$$F = V$$
$$O = \{\mathit{move}(v_s, v_t) \mid (v_s, v_t) \in E\}$$
$$G = I = \varnothing$$

Pour des graphes non-orientés, la construction est fondamentalement la même, à cela près que tous les déplacements sont possibles dans un sens comme dans l'autre.

Construction 2. *Soit $\mathcal{G} = (V, E)$ un graphe non-orienté. On construit l'instance de planification $P_\mathcal{G} = \langle F, I, O, G\rangle$, comme dans la Construction 1, mais avec :*

$$O = \{\mathit{move}(v_s, v_t), \mathit{move}(v_t, v_s) \mid \{v_s, v_t\} \in E\}$$

3 Problème d'isomorphisme STRIPS

Dans cette partie, nous nous intéressons au problème d'isomorphisme entre deux instances STRIPS. Après avoir défini la forme d'isomorphisme STRIPS que nous proposons, nous présentons nos résultats de complexité.

Définition 3 (Isomorphisme entre instances STRIPS)**.** *Soient $P = \langle F, I, O, G\rangle$ et $P' = \langle F', I', O', G'\rangle$ deux instances STRIPS. Un isomorphisme entre P et P' est une paire (υ, ν) de bijections $\upsilon : F \to F'$ et $\nu : O \to O'$ t.q.*

$$\forall o \in O,\ \nu(o) = \langle \upsilon(\mathit{pre}(o)), \upsilon(\mathit{eff}(o))\rangle \quad (2)$$
$$\upsilon(I) = I' \quad (3)$$
$$\upsilon(G) = G' \quad (4)$$

Où, pour deux ensembles disjoints $F_1, F_2 \subseteq F$ de fluents,

$$\upsilon(F_1 \cup \neg F_2) = \upsilon(F_1) \cup \neg\upsilon(F_2)$$

Une conséquence immédiate de cette définition est qu'un tel isomorphisme conserve l'espace des plans : toute suite d'actions $o_1, \ldots, o_n \in O$ est un plan pour P si, et seulement si, la suite associée $\nu(o_1), \ldots, \nu(o_n)$ est un plan pour P'. Cette propriété de morphisme est assurée par l'équation (2). De la même façon, tous les plans-solutions sont conservés, et ce grâce aux conditions imposées par les équations (3) et (4). On en vient alors au problème SI, que l'on définit formellement afin d'établir sa complexité :

Problème 1. *Problème d'Isomorphisme STRIPS SI*

Entrée	*Deux instances STRIPS P et P'*
Sortie	*Un isomorphisme (υ, ν) entre P et P', s'il en existe un*

Proposition 3. *SI est GI-complet*

Le reste de cette partie est dédiée à la preuve de ce résultat. Nous montrons tout d'abord la GI-difficulté du problème, puis nous montrons ensuite son appartenance à GI.

Lemme 1. *SI est GI-difficile*

Démonstration. Cette preuve consiste en une réduction depuis le problème d'isomorphisme de graphes orientés.
Soit $(\mathcal{G}, \mathcal{G}')$ une instance du problème d'isomorphisme de graphes orientés, où $\mathcal{G} = (V, E)$ et $\mathcal{G}' = (V', E')$.
La preuve est basée sur la Construction 1, qui nous permet d'obtenir en temps polynomial les problèmes de planification STRIPS $P_\mathcal{G}$ et $P_{\mathcal{G}'}$.
Montrons alors qu'il existe un isomorphisme $u : V \to V'$ entre $\mathcal{G}$ et $\mathcal{G}'$ ssi il existe un isomorphisme (υ, ν) entre $P_\mathcal{G}$ et $P_{\mathcal{G}'}$. On procède en deux temps : d'abord, nous identifions les fonctions u et υ, puis nous montrons ensuite que la condition de morphisme entre les arêtes de $\mathcal{G}$ et $\mathcal{G}'$ est nécessairement vérifiée, grâce à la condition de manière sur les actions des instances STRIPS $P_\mathcal{G}$ et $P_{\mathcal{G}'}$, et inversement.
($\Rightarrow$) Supposons tout d'abord qu'il existe un isomorphisme de graphes $u : V \to V'$ entre $\mathcal{G}$ et $\mathcal{G}'$. Montrons qu'il existe alors un isomorphisme entre $P_\mathcal{G}$ et $P_{\mathcal{G}'}$. Définissons la transformation ν sur les éléments de O par $\nu(\langle \mathsf{pre}(o), \mathsf{eff}(o)\rangle) = \langle u(\mathsf{pre}(o)), u(\mathsf{eff}(o))\rangle$. On montre alors que $\nu : O \to O'$ est bien définie, et que le couple (u, ν) définit lui-même un isomorphisme entre $P_\mathcal{G}$ et $P_{\mathcal{G}'}$.
Pour tout $o \in O$, par construction, il existe un unique couple $(v_1, v_2) \in V^2$ tel que $o = \mathsf{move}(v_1, v_2)$. Ainsi :

$$\begin{aligned} o \in O \ &\textit{ssi}\ (v_1, v_2) \in E \\ &\textit{ssi}\ (u(v_1), u(v_2)) \in E' \\ &\textit{ssi}\ \mathsf{move}(u(v_1), u(v_2)) \in O' \\ &\textit{ssi}\ \nu(\mathsf{move}(v_1, v_2)) \in O' \\ &\textit{ssi}\ \nu(o) \in O' \end{aligned}$$

Ainsi, on a donc bien que $P_\mathcal{G}$ et $P_{\mathcal{G}'}$ sont isomorphes.
($\Leftarrow$) Supposons maintenant que $P_\mathcal{G}$ et $P_{\mathcal{G}'}$ sont isomorphes, et soit (υ, ν) un isomorphisme entre ces deux instances STRIPS. Montrons qu'il existe un isomorphisme entre $\mathcal{G}$ et $\mathcal{G}'$. Par hypothèse, on a $\upsilon : F \to F'$ (ou similairement $\upsilon : V \to V'$) et $\nu : O \to O'$ deux bijections.

Par la suite, on note g et g' les bijections $g : E \to O$ et $g' : E' \to O'$, dont l'existence nous est assurée par construction (par exemple, $g((v_1, v_2)) = \mathsf{move}(v_1, v_2)$).
Montrons alors que la fonction υ est un isomorphisme de graphes entre $\mathcal{G}$ et $\mathcal{G}'$. On a que, pour tout $e = (v_1, v_2) \in E$, $g(e) = \mathsf{move}(v_1, v_2) \in O$. Donc $\nu \circ g(e) = \nu(\mathsf{move}(v_1, v_2))$, et on a donc, $\nu \circ g(e) = \mathsf{move}(\upsilon(v_1), \upsilon(v_2)) \in O'$. Alors, $g'^{-1} \circ \nu \circ g(e) = (\upsilon(v_1), \upsilon(v_2))$, mais l'on remarque aussi que $g'^{-1} \circ \nu \circ g(e) \in E'$. Par conséquent, $(\upsilon(v_1), \upsilon(v_2)) \in E'$.
La réciproque se montre par des arguments similaires, en utilisant le fait que g, g' et ν sont des bijections. Par conséquent, ν est un isomorphisme de graphes entre $\mathcal{G}$ et $\mathcal{G}'$. □

Lemme 2. *SI appartient à GI*

Démonstration. La preuve utilise une réduction de SI au problème d'isomorphisme de modèles finis, tel que défini dans la Définition 2. De plus, dans cette preuve, on utilise la construction suivante. Pour toute instance STRIPS $P = \langle F, I, O, G\rangle$, on construit le modèle fini :

$$M_P = \langle V, \mathcal{R}_F, \mathcal{R}_I, \mathcal{R}_O, \mathcal{R}_G, \mathcal{R}_{\mathsf{pre}^+}, \mathcal{R}_{\mathsf{pre}^-}, \mathcal{R}_{\mathsf{eff}^+}, \mathcal{R}_{\mathsf{eff}^-}\rangle$$
$$V = F \sqcup O$$
$$\text{Pour tout } X \in \{F, I, O, G\}, \mathcal{R}_X = X$$
$$\text{Pour } \mathcal{S} \in \mathcal{C}, \mathcal{R}_{\mathcal{S}} = \left\{(o, f) \in V^2 \mid o \in O \textit{ et } f \in \mathcal{S}(o)\right\}$$

où $\mathcal{C} = \{\mathsf{pre}^+, \mathsf{pre}^-, \mathsf{eff}^+, \mathsf{eff}^-\}$. Montrons alors que deux instances de planification STRIPS P et P' sont isomorphes ssi M_P et $M_{P'}$ sont isomorphes.
On notera $M_P = \langle V, \mathcal{R}_F, \ldots, \mathcal{R}_{\mathsf{eff}^-}\rangle$ et $M_{P'} = \langle V', \mathcal{R}'_F, \ldots, \mathcal{R}'_{\mathsf{eff}^-}\rangle$
($\Rightarrow$) Supposons qu'il existe un isomorphisme (υ, ν) entre P et P'. On définit $g : V \to V'$ telle que, pour tout $x \in V$,

$$g(x) = \begin{cases} \upsilon(x) & \text{si } x \in F \\ \nu(x) & \text{si } x \in O \end{cases}$$

g est une bijection, de manière immédiate, par hypothèse sur (υ, ν). De plus, pour $X \in \{F, I, O, G\}$, $\mathcal{R}_X(v)$ ssi $\mathcal{R}'_X(g(v))$, par hypothèse sur (υ, ν).
Soit $o \in O$, $p \in F$. On a que, pour tout $\mathcal{S} \in \mathcal{C} = \{\mathsf{pre}^+, \mathsf{pre}^-, \mathsf{eff}^+, \mathsf{eff}^-\}$,

$$\begin{aligned} &\mathcal{R}_{\mathcal{S}}(o, p) \textit{ ssi } o \in O \textit{ et } p \in \mathcal{S}(o) && (5)\\ &\qquad \textit{ssi } \nu(o) \in O' \textit{ et } \upsilon(p) \in \mathcal{S}(\nu(o)) && (6)\\ &\qquad \textit{ssi } \mathcal{R}'_{\mathcal{S}}(\nu(o), \upsilon(p)) \\ &\qquad \textit{ssi } \mathcal{R}'_{\mathcal{S}}(g(o), g(p)) \end{aligned}$$

L'équivalente entre (5) et (6) repose sur la définition de l'isomorphisme. Les autres équivalences suivent par définition. On a donc bien M_P et $M_{P'}$ sont isomorphes.
($\Leftarrow$) Supposons M_P et $M_{P'}$ isomorphes, et que $g : V \to V'$ est un isomorphisme entre les deux modèles finis. On définit $\upsilon = g_{|F}$ (resp. $\nu = g_{|O}$) comme la restriction de g au sous-domaine F (resp. O). On a alors que $\upsilon : F \to F'$, puisque par l'absurde, on aurait sinon l'existence un élément $v \in V$ tel que $\mathcal{R}_F(v)$ mais ne vérifiant pas $\mathcal{R}'_F(g(v))$, ce qui contredit l'hypothèse comme quoi g est un isomorphisme. De manière similaire, on a que $\nu : O \to O'$.
De même, avec des arguments semblables aux arguments précédents, on a que, pour tout $o \in O$,

$$\begin{aligned} &o = \langle \mathsf{pre}(o), \mathsf{eff}(o)\rangle \\ &\textit{ssi } \forall p \in F, \forall \mathcal{S} \in \mathcal{C}, p \in \mathcal{S}(o) \Leftrightarrow \mathcal{R}_S(o, p) && (7)\\ &\textit{ssi } \forall p \in F, \forall \mathcal{S} \in \mathcal{C}, p \in \mathcal{S}(o) \Leftrightarrow \mathcal{R}'_{\mathcal{S}}(g(o), g(p)) && (8)\\ &\textit{ssi } \forall p \in F, \forall \mathcal{S} \in \mathcal{C}, g(p) \in g(\mathcal{S}(o)) \Leftrightarrow \mathcal{R}'_{\mathcal{S}}(g(o), g(p)) && (9)\\ &\textit{ssi } \forall p' \in F', \forall \mathcal{S} \in \mathcal{C}, p' \in g(\mathcal{S}(o)) \Leftrightarrow \mathcal{R}'_{\mathcal{S}}(g(o), p') && (10)\\ &\textit{ssi } g(o) = \langle g(\mathsf{pre}(o)), g(\mathsf{eff}(o))\rangle && (11)\\ &\textit{ssi } \nu(o) = \langle \upsilon(\mathsf{pre}(o)), \upsilon(\mathsf{eff}(o))\rangle \end{aligned}$$

Les équivalences entre la première ligne et (7), ainsi que entre (10) et (11), tiennent par construction de M_P et $M_{P'}$. L'équivalence entre (7) et (8) est une conséquence du fait que g est un morphisme. Entre (8) et (9), on utilise le fait que g est une bijection. Pour l'équivalence entre (9) et (10), on utilise le fait que g est surjective sur F'.
On a donc bien montré que (υ, ν) est un morphisme, et donc un isomorphisme sur son domaine et codomaine. □

Ces résultats tiennent encore si l'on n'impose pas que les états initiaux et buts respectifs de P et P' soient en bijections, c'est-à-dire si l'on n'impose pas les conditions (3) et (4). En effet, la preuve de difficulté est basée sur une réduction depuis le problème d'isomorphisme de graphes, qui ne possèdent pas de sommet "initial" ou "final" à proprement parler, ce qui rend triviaux l'état initial et le but de la construction. De manière réciproque, la preuve que SI appartient à GI peut inclure, ou non, les informations relatives aux états initiaux et aux buts, et rester correcte pour la version alternative de SI dans laquelle on n'impose aucune condition sur les états initiaux ou le but. Ainsi, cela signifie que la majeure partie de la difficulté de SI réside en la complexité d'associer correctement les structures internes respectives des espaces d'états de chaque instance, et que des propriétés supplémentaires sur certains états (comme être un état final ou initial) n'influencent pas significativement la complexité du problème. Cette observation est cohérente par l'intuition que l'on a de la classe GI : il est connu [14] que trouver un isomorphisme conservant les couleurs entre deux graphes colorés (i.e., un isomorphisme qui conserve une propriété donnée entre noeuds) est aussi un problème complet pour la classe GI.

4 Problème d'isomorphisme de sous-instance STRIPS

Dans cette partie, nous présentons les problèmes SSI-H et SSI, qui traitent de la synthèse (de deux formes différentes) d'isomorphismes entre une instance de planification P et d'une sous-isntance d'une autre instance STRIPS P'. Dans cette section, on identifie la complexité de ces deux problèmes, en prouvant leur NP-complétude. On utilise ce ré-

sultat pour proposer, dans la partie subséquente, un algorithme pour SSI et SSI-H. Cet algorithme est basé sur une réduction vers SAT, enrichi par une étape de pré-traitement s'appuyant sur de la propagation de contraintes.
On commence par introduire la notion d'*isomorphisme homogène de sous-instances*, qui est une forme d'isomorphisme entre P et une sous-instance de P' qui ne conserve cependant pas l'état initial ni le but. Il s'agit donc d'associer l'espace d'états complet du problème P à une partie de l'espace d'états de P', sans pour autant tenir compte des états initiaux ou finaux de ces instances.

Définition 4 (Isomorphisme homogène de sous-instance)**.** *Soient deux instances STRIPS $P = \langle F, I, O, G\rangle$ et $P' = \langle F', I', O', G'\rangle$. Un* isomorphisme homogène de sous-instances *de P à P' est un couple (υ, ν) de fonctions injectives $\upsilon : F \to F'$ et $\nu : O \to O'$ qui respectent la condition (2) de la Définition 3.*

Problème 2. *Isomorphisme homogène de sous-instance STRIPS SSI-H*

Entrée	*Deux instances STRIPS P et P'*
Sortie	*Un isomorphisme homogène de sous-instances (υ, ν) entre P et P', s'il en existe un*

Un isomorphisme homogène de sous-instances entre P et P' peut s'avérer utile, par exemple, dans le cas où l'on a réussi à compiler l'ensemble des plans pour P' et que l'on souhaite extraire un plan pour P. La définition suivante d'isomorphisme de sous-instance suivante, plus forte que dans le cas homogène, prend en compte les états initiaux et les buts. Cette notion nous permet de ne conserver que les plans-*solutions* d'un problème à l'autre.

Définition 5 (Isomorphisme de sous-instance)**.** *Un* isomorphisme de sous-instance *entre P et P' est un isomorphisme de sous-instance homogène qui respecte de surcroît les conditions (3) et (4) de la Définition 3.*

Problème 3. *Isomorphisme de sous-instance STRIPS SSI*

Entrée	*Deux instances STRIPS P et P'*
Sortie	*Un isomorphisme de sous-instance (υ, ν) entre P et P', s'il en existe un*

La principale différence entre SI et SSI est que, pour SSI, la condition sur la bijectivité de υ et ν est relaxée. L'injectivité des deux fonctions est cependant toujours requise, afin d'empêcher les fluents et actions d'être fusionnés ensemble par la fonction. Les autres conditions restent inchangées.
Le résultat principal de cette partie est présenté ci-dessous. La preuve se base sur une réduction à partir du problème d'isomorphisme de sous-graphes, dont la NP-complétude est un résultat bien établi [5]. On commence par présenter le problème d'isomorphisme de sous-graphes. En essence, il consiste à trouver un isomorphisme g entre le graphe $\mathcal{G}$ et le sous-graphe $(g(V), E' \cap g(V) \times g(V))$ de $\mathcal{G}'$.

Problème 4. *Problème d'isomorphisme de sous-graphes*

Entrée	*Deux graphes non-orientés $\mathcal{G}(V, E)$, $\mathcal{G}'(V', E')$*
Sortie	*Une fonction injective $g : V \to V'$ telle que, pour tout $v_1, v_2 \in V$, $\{v_1, v_2\} \in E$ ssi $\{g(v_1), g(v_2)\} \in E'$.*

Proposition 4. *SSI est NP-complet*

Démonstration. Afin de prouver que SSI est dans NP, il suffit de remarquer que les fonctions υ et ν constituent un certificat de taille polynomial, qui peut lui-même être testé en temps polynomial. La preuve de NP-difficulté de SSI consiste en une réduction à partir du problème d'isomorphisme de sous-graphes, ce qui se fait rapidement grâce à la construction que l'on a proposée précédemment.
Soit $(\mathcal{G}, \mathcal{G}')$ une instance du problème d'isomorphisme de sous-graphes. Avec la Construction 2, on construit les instances de planification $P_{\mathcal{G}}$ et $P_{\mathcal{G}'}$. Montrons alors qu'il existe un isomorphisme de sous-graphes g entre $\mathcal{G}$ ssi il existe un isomorphisme de sous-instance entre $P_{\mathcal{G}}$ et $P_{\mathcal{G}'}$.
($\Rightarrow$) Supposons qu'il existe un isomorphisme de sous-instance $g : V \to V'$ entre $\mathcal{G}$ et $\mathcal{G}'$. Alors, par construction, comme $F = V$ et $F' = V'$, g est aussi une fonction injective de F à F'. De plus, on pose $\nu : O \to O'$, $\nu : \mathsf{move}(v_1, v_2) \mapsto \mathsf{move}(g(v_1), g(v_2))$. ν est bien défini, puisque $\{v_1, v_2\} \in E$ ssi $\{g(v_1), g(v_2)\} \in E'$, donc $\mathsf{move}(v_1, v_2) \in O$ ssi $\mathsf{move}(g(v_1), g(v_2)) \in O'$. Par ailleurs, comme g est injective, ν l'est aussi. Par conséquent, (g, ν) est bien un isomorphisme entre $P_{\mathcal{G}}$ et $P_{\mathcal{G}'}$.
($\Leftarrow$) Supposons qu'il existe un isomorphisme de sous-instance (υ, ν) entre $P_{\mathcal{G}}$ et $P_{\mathcal{G}'}$. Comme précédemment, $\upsilon : V \to V'$ est une fonction injective. De plus, on a que

$$\begin{aligned} & (v_1, v_2) \in E \\ \textit{ssi } & \mathsf{move}(v_1, v_2) \in O \\ \textit{ssi } & \nu\left(\mathsf{move}(v_1, v_2)\right) \in O' \\ \textit{ssi } & \mathsf{move}(\upsilon\left(v_1\right), \upsilon\left(v_2\right)) \in O' \\ \textit{ssi } & \left(\upsilon\left(v_1\right), \upsilon\left(v_2\right)\right) \in E' \end{aligned}$$

Par conséquent, υ est un isomorphisme de sous-graphes entre $\mathcal{G}$ et $\mathcal{G}'$. □

La preuve de l'appartenance à NP de SSI-H est immédiate : comme pour SI, un isomorphisme de sous-instance est un certificat, vérifiable en temps polynomial.
La preuve précédente de la NP-difficulté de SSI est indépendante de l'état initial et du but. Par conséquent, elle montre aussi que SSI-H est NP-dur.

Corollaire 1. *SSI-H est NP-complet*

5 Un algorithme pour SSI

Dans cette partie, nous présentons notre algorithme pour le problème SSI, dont le pseudo-code est fourni dans l'Algorithme 1. Cet algorithme repose sur la compilation du problème en une formule propositionnelle, qui est ensuite passée à un solveur SAT. La procédure est renforcée par un pré-traitement, basé sur la propagation de contraintes, qui

Algorithme 1 trouver_isomorphisme_sous-instance

Entrée : Deux instances STRIPS P et P'
Sortie : Un isomorphisme de sous-instance entre P et P' s'il en existe un

1: Initialiser_domaines(F, O)
/* Elimination des associations impossibles */
2: $Q := F \cup O$
3: **tant que** $Q \neq \varnothing$ **faire**
4: $v := Q$.Pop()
5: $r :=$ Réviser(v)
6: **si** r **alors**
7: **si** $\mathcal{D}(v) = \varnothing$ **alors retourner** UNSAT
8: **sinon** Q.Ajouter$(\{v' \mid v' \text{ lié à } v\})$
/* Phase de recherche avec SAT */
9: $\varphi :=$ Encoder_en_FNC$(P, P', \mathcal{D})$
10: **retourner** Interpréter(Solveur.Trouver_modèle(φ))

nous permet d'éliminer des associations incohérentes entre les fluents et actions respectifs de chaque instance.

Pour deux instances STRIPS P et P', l'algorithme retourne un isomorphisme de sous-instance (υ, ν), lorsqu'il en existe un. L'Algorithme 1 comporte deux étapes principales. La première étape, qui s'étend de la ligne 2 à la ligne 8, consiste à éliminer autant d'associations inconsistantes que possible entre les fluents (resp. actions) du problèmes P et les fluents (resp. actions) du problème P', dès lors que l'on a détecté une incohérence d'ordre syntaxique qui a été éventuellement propagée (comme l'on présente plus bas). La deuxième étape, qui débute à la ligne 9, consiste en une phase de recherche, ce qui passe par l'encodage du problème en une formule en FNC, qui est ensuite passée à un solveur SAT.

5.1 Élimination des associations invalides

On appelle *association* entre fluents un couple $(f, f') \in F \times F'$ tel que f' est un candidat pour la valeur de $\upsilon(f)$. De même, on appelle association entre actions un couple $(o, o') \in O \times O'$ tel que o' est un candidat pour la valeur de $\upsilon(o)$. Compte tenu de la taille des problèmes considérés, il est crucial de détecter aussi tôt que possible les associations n'appartenant à aucun isomorphisme de sous-instance valide, afin de réduire la taille de l'espace de recherche.

Afin d'éliminer autant d'associations inconsistantes que possible, nous utilisons une technique basée sur la propagation de contraintes, telle que couramment étudiée dans la littérature de programmation par contraintes. L'idée générale est de maintenir, pour chaque fluent $f \in F$ de P, un *domaine* $\mathcal{D}(f) \subseteq F'$ de fluents de P', qui représente les candidats plausibles pour la valeur de $\upsilon(f)$. De manière analogue, chaque action $o \in O$ se voit assigner un domaine $\mathcal{D}(o) \subseteq O'$. Par la suite, on appelle *variable* tout fluent ou action. La procédure que l'on présente ci-dessous a pour objectif de réduire les domaines des différentes variables, dans l'optique d'alléger la charge passée au solveur SAT.

La première étape consiste en l'initialisation des domaines. Pour chaque fluent $f \in F$, on a $\mathcal{D}(f) = F'$. Les domaines des actions sont, pour leur part, initialisés en fonction de leur *profil d'action*. Pour chaque action $o \in O \cup O'$, on définit le vecteur $\mathsf{profile}(o) \in \mathbb{N}^6$, que l'on appelle le *profil* de o. Ce vecteur est une abstraction numérique de quelques caractéristiques de l'action, pensée de sorte à ce que si $\mathsf{profile}(o) \neq \mathsf{profile}(o')$, alors les deux actions $o \in O$ et $o' \in O'$ ne peuvent être associés.

Dans la pratique, $\mathsf{profile}(o)$ compte le nombre de fluents positifs et négatifs dans les préconditions et effets de o, ainsi que dans le nombre de fluents qui sont des *additions strictes* ou des *suppressions strictes*. Un fluent f est une *addition stricte* pour l'action o si $f \in \mathsf{pre}^-(o) \wedge f \in \mathsf{eff}^+(o)$, et est une *suppression stricte* si $f \in \mathsf{pre}^+(o) \wedge f \in \mathsf{eff}^-(o)$. Finalement, on initialise le domaine de chaque action $o \in O$ tel que :

$$\mathcal{D}(o) = \{o' \in O' \mid \mathsf{profile}(o') = \mathsf{profile}(o)\}$$

La seconde étape consiste à propager les contraintes supplémentaires posées par les restrictions fraîchement identifiés du domaine. La technique que l'on propose est basée sur l'idée de cohérence d'arc, qui est omniprésente dans le domaine de la programmation par contraintes. L'idée consiste à éliminer du domaine des fluents (resp. actions) les fluents-candidats (resp. actions-candidates) qui ne sont pas supportés par le domaine d'une action (resp. fluent). Cela permet de faire la jonction entre les contraintes posées par les fluents et les contraintes posées par les actions.

Plus précisément, soit un fluent $f \in F$. Lorsqu'une action $o \in O$ est tel que f apparaît (positivement ou négativement) dans ses préconditions ou ses effets, alors on dit que o *dépend* de f. On note alors $d(f)$ l'ensemble des actions qui dépendent de f. On utilise des notations similaires pour $d(f')$ lorsque $f' \in F'$. Plaçons-nous maintenant dans le cas où $\upsilon(f) = f'$. Une conséquence de l'équation (2) de la Définition 3 est que chaque action de $d(f)$ doit avoir son image par ν dans $d(f')$. Sinon, on aurait que f apparaît dans $\mathsf{pre}(o)$ ou $\mathsf{eff}(o)$, mais $\upsilon(f)$ n'apparaîtrait ni dans $\upsilon(\mathsf{pre}(o))$, ni dans $\upsilon(\mathsf{eff}(o))$. Ainsi, si pour une action donnée $o \in d(f)$, aucun candidat pour son image n'est dans $d(f')$ (*i.e.*, $\mathcal{D}(o) \cap d(f') = \varnothing$), alors on a que f' ne peut pas être choisi pour l'image de f.

Dans la suite, on précise l'argument tout juste énoncé, en identifiant $\mathsf{pre}^+(o)$ et $\mathsf{pre}^+(o'), \ldots, \mathsf{eff}^-(o)$ et $\mathsf{eff}^-(o')$. On obtient alors les contraintes suivantes pour $\mathcal{D}(f)$, dans lesquelles on note $\mathcal{C} = \{\mathsf{pre}^+, \mathsf{pre}^-, \mathsf{eff}^+, \mathsf{eff}^-\}$:

$$\mathcal{D}(f) \subseteq \left\{ f' \;\middle|\; \begin{array}{l} \forall o \in O, \forall \mathcal{S} \in \mathcal{C} \text{ s.t. } f \in \mathcal{S}(o), \\ \exists o' \in \mathcal{D}(o) \text{ s.t. } f' \in \mathcal{S}(o') \end{array} \right\} \quad (12)$$

Le même exercice peut être fait pour les actions. Soit $o \in O$ une action quelconque, et considérons un candidat pour son image $o' \in O'$. Afin de satisfaire la propriété de morphisme, dans le cas où l'on a effectivement $\nu(o) = o'$, on doit trouver dans $\mathsf{pre}^+(o')$ un fluent qui appartient à $\mathcal{D}(f)$, et ce pour chaque fluent $f \in \mathsf{pre}^+(o')$. Plus généralement, et plus formellement, cela impose les conditions suivantes :

$$\mathcal{D}(o) \subseteq \{o' \mid \forall \mathcal{S} \in \mathcal{C}, \forall f \in \mathcal{S}(o), \exists f' \in \mathcal{D}(f) \cap \mathcal{S}(o')\} \quad (13)$$

Algorithmiquement, on s'assure de la satisfaction de ces contraintes en utilisant une version de AC3 [12, 13] adaptée à nos besoins. On retrouve en particulier la sous-routine de révision de la cohérence des domaines des variables.
Réviser une variable v consiste à vérifier que l'ensemble des éléments de son domaine sont conformes aux conditions nécessaires évoquées précédemment, c'est-à-dire l'équation (12) si v est un fluent, ou (13) si v est une action. La boucle principale, dépeinte dans l'Algorithme 1, effectue donc une révision itérative de tous les fluents et actions, en maintenant une file Q des variables à réviser (ligne 1). Cette file est initialisée avec l'ensemble des variables, afin que chacune soit révisée au moins une fois.
Si, lors de la révision d'une variable v, le domaine de v est modifié par la procédure, alors toutes les variables qui sont liées à v sont ajoutées à la file des variables à réviser (lignes 5 à 9). On dit que v' est liée à la variable v si v est un fluent et $v' \in d(v)$, ou inversement. Si le domaine d'une variable est vide, alors on est assuré qu'aucun isomorphisme de sous-instance n'existe, et l'algorithme se termine immédiatement (lignes 6 et 7). Sinon, la boucle se termine lorsque la file des variables à réviser est vide.
Cette procédure ne permet généralement pas à l'algorithme de conclure, mais allège la charge passée à la phase de recherche, que l'on présente dans la partie suivante.

5.2 Encodage en une instance SAT

Dans cette partie, nous présentons notre construction de la formule propositionnelle φ évoquée plus tôt, dont les modèles permettent de trouver un isomorphisme de sous-instance. φ est construit sur l'ensemble de variable $\mathit{Var}(\varphi)$, que l'on définit comme :

$$\mathit{Var}(\varphi) = \left\{ f_i^j \;\middle|\; i \in F, j \in F' \right\} \cup \{ o_r^s \mid r \in O, s \in O' \}$$

Une variable propositionnelle de la forme f_i^j représente l'association d'un fluent $i \in F$ à un fluent $j \in F'$. De même, o_r^s représente l'association de $r \in O$ à $s \in O'$.
Dans le reste de cette partie, nous présentons notre construction de la formule φ, qui encode une instance de SSI passée en entrée à l'Algorithme 1. φ consiste en la conjonction des formules suivantes, qui nous assurent chacune une propriété différente sur un modèle.
La formule présentée dans l'équation (14) nous assure de l'unicité de l'image de chaque fluent. On obtient la même propriété sur les images des actions en remplaçant les variables de la forme f_i^j par des variables o_i^j, et en adaptant les domaines de i et j en conséquence.

$$\bigwedge_{i \in F} \left(\bigvee_{j \in \mathcal{D}(i)} f_i^j \;\wedge \bigwedge_{\substack{j,k \in \mathcal{D}(i) \\ j \neq k}} (\neg f_i^j \vee \neg f_i^k) \right) \tag{14}$$

Afin d'assurer l'injectivité de υ et ν, on ajoute l'équation (15) pour les fluents, qui est aussi adaptée comme précédemment dans le cas des actions.

$$\bigwedge_{i \in F'} \bigwedge_{\substack{j,k \in F \\ j \neq k}} \neg f_j^i \vee \neg f_k^i \tag{15}$$

La propriété de morphisme est assurée par les formules (16) et (17), pour chaque $\mathcal{S} \in \mathcal{C} = \{\mathsf{pre}^+, \mathsf{pre}^-, \mathsf{eff}^+, \mathsf{eff}^-\}$. Plus précisément, (16) nous assure que, pour tout $\mathcal{S} \in \mathcal{C}$ et pour une action $o \in O$ quelconque, on a $\upsilon(\mathcal{S}(o)) \subseteq \mathcal{S}(\nu(o))$. Réciproquement, (17) nous assure que $\mathcal{S}(\nu(o)) \subseteq \upsilon(\mathcal{S}(o))$.

$$\bigwedge_{\substack{r \in O \\ s \in O'}} \left(o_r^s \longrightarrow \bigwedge_{i \in \mathcal{S}(r)} \bigvee_{j \in \mathcal{S}(s)} f_i^j \right) \tag{16}$$

$$\bigwedge_{\substack{r \in O \\ s \in O'}} \left(o_r^s \longrightarrow \bigwedge_{j \in \mathcal{S}(s)} \bigvee_{i \in \mathcal{S}(r)} f_i^j \right) \tag{17}$$

Finalement, il ne reste plus qu'à conserver l'état initial et le but (i.e., nous assurer du respect des équations (3) et (4)). On note alors I^+ (resp. I^-) l'ensemble des fluents positifs (resp. négatifs) dans I, et on utilise des notations similaires pour G, I' et G'. Pour chaque $\mathcal{T} \in \{I^+, I^-, G^+, G^-\}$, ainsi que pour l'ensemble $\mathcal{T}' \in \{I'^+, I'^-, G'^+, G'^-\}$ correspondant, on ajoute alors les formules suivantes :

$$\bigwedge_{i \in \mathcal{T}} \bigvee_{j \in \mathcal{T}'} f_i^j \quad \wedge \quad \bigwedge_{j \in \mathcal{T}'} \bigvee_{i \in \mathcal{T}} f_i^j \tag{18}$$

Les formules définies par (14), (15), et (18) sont immédiatement en FNC, et la taille de leur conjonction est un $\mathcal{O}(|F| \cdot |F'|^2 + |O| \cdot |O'|^2)$, avec l'hypothèse que $|F| \leq |F'|$ si $|O| \leq |O'|$ (sinon, la conclusion est immédiate). Par ailleurs, les formules définies par (16) et 17) peuvent être immédiatement converties en FNC en dupliquant les implications dans chaque clause. Elle ont alors une taille $\mathcal{O}(|O| \cdot |O'| \cdot |F| \cdot |F'|)$.
L'étape de pré-traitement présentée dans la Partie 5.1 nous permet de simplifier φ. En, effet, si l'on sait que le fluent $i \in F$ (resp. l'action $r \in O$) ne peut pas être assigné au fluent $j \in F'$ (resp. $s \in O'$), alors f_i^j (resp. o_r^s) est nécessairement faux dans tout modèle de φ. Ainsi, nos formules étant en FNC, toutes les occurrences positives de f_i^j peuvent être supprimées dans les clauses de φ, tandis que les clauses où f_i^j apparaît négativement peuvent être simplifiées.
Pour adapter l'algorithme précédent à SSI-H, il suffit de supprimer les formules de (18), et de conserver le reste des formules et de l'algorithme.

6 Évaluation empirique

L'Algorithme 1 a fait l'objet d'une implémentation en Python 3.10, ce qui nous a permis de le tester sur des instances de SSI et de SSI-H. L'analyse syntaxique a été confiée au module dédié de TouISTPlan [3], ce qui nous permet de convertir des instances de planification PDDL en une représentation STRIPS. Comme solveur SAT, nous avons eu recours à Maple LCM [11], gagnant du *main track* de la compétition SAT 2017. L'ensemble des ressources nécessaires à la reproduction des expériences présentées ici peut être trouvé sur la page web dédiée [1].

1. https://github.com/arnaudlequen/PDDLIsomorphismFinder

Les expériences ont été effectuées sur une machine fonctionnant sous Rocky Linux 8.5, dont le processeur était un Intel Xeon E5-2667 v3 processor, en utilisant au plus 8Go de mémoire vive et 4 threads par test.

Notre jeu de test est tiré de huit ensembles utilisés dans l'*International Planning Competition* : Blocks, Gripper, Hanoi, Rovers, Satellite, Sokoban, TSP and Visitall. Pour chacun de ces domaines, nous avons créé ce que l'on appelle des *instances de morphismes STRIPS*, qui sont des paires d'instances du même domaine. Nous avons effectué cette opération pour chaque paire possible d'instances de planification, et ce pour chaque domaine pris en compte. Ainsi, une instance de morphisme STRIPS est à la fois une instance de SSI et une instance de SSI-H. Nous avons donc pu évaluer notre implémentation pour chacun de ces deux problèmes, sur le même jeu de tests.

L'objectif de ces évaluations expérimentales est double. D'une part, il s'agit de montrer que, malgré la difficulté théorique du problème, il est en pratique possible d'identifier un isomorphisme (homogène) de sous-instances en temps raisonnable, et ce pour des problèmes de tailles non-triviales. D'autre part, il s'agit de tester l'efficacité de notre technique de pré-traitement présentée en Partie 5.1, afin de montrer que le coût supplémentaire ne surpasse pas l'avantage qu'il donne à l'algorithme de recherche.

La couverture absolue de notre implémentation sur notre jeu de tests est présentée dans la Table 1 pour SSI et SSI-H. Le tableau montre le nombre d'instances du jeu de tests pour lesquelles notre implémentation termine avec les contraintes spatialles et temporelles allouées. Nous ne présentons ici que les domaines pour lesquels notre algorithme s'est trouvé en mesure de résoudre au moins une instance. Pour certains domaines, même l'instance la plus petite est de taille trop conséquente pour notre implémentation. Parmi ces domaines, on peut citer, par exemple, Visitall, Barman ou Woodworking.

On remarque tout d'abord que les problèmes SSI-H et SSI sont souvent comparables en terme de difficulté, à quelques domaines près qui font figure d'exceptions. Parmi ces exceptions, on compte par exemple TSP et Gripper, pour lesquels relaxer la condition sur l'état initial et le but permet de résoudre, respectivement, 40% et 133% plus d'instances. Pour ces deux domaines, cela peut s'expliquer par les contraintes supplémentaires imposées par SSI. En effet, une de leurs conséquences immédiates est que toutes les paires d'instances différentes de TSP (ou de Gripper) sont des instances négatives de SSI, qui posent davantage de difficultés au solveur SAT que des instances positives.

De plus, les bénéfices de l'étape de pré-traitement dépassent presque toujours le coût de l'opération. En effet, l'immense majorité des instances que notre algorithme peut résoudre sans pré-traitement peuvent aussi être résolues une fois le pré-traitement activé. A plus forte raison, on peut surtout constater que le pré-traitement améliore grandement les performances générales de notre algorithme, tant et si bien que certains domaines, précédemment hors de la portée de notre algorithme, deviennent faisables. Parmi de tels cas extrêmes, on peut citer le domaine Sokoban, pour lequel notre algorithme est impuissant sans l'étape préalable d'élimination des associations inconsistantes. En effet, des 204 instances résolues par notre implémentation, aucune n'est résolue une fois le pré-traitement désactivé. Dans le cas général, cependant, on observe une augmentation du nombre d'instances résolues qui, sans pour autant changer d'ordre de grandeur, reste significatif. Par exemple, pour le domaine Satellite, dans le cas de SSI, 34 instances sont résolues lorsque la propagation de contraintes est activée, alors que seulement 22 peuvent être décidées sans pré-traitement.

Plus spécifiquement, pour presque toutes les instances de test, le pré-traitement permet une réduction non-négligeable de la taille de l'encodage propositionnel. C'est ce qui est montré dans les colonnes appelées "Simp. Moy" dans la Table 1, qui représentent la proportion moyenne de clauses simplifiées grâce à l'étape d'élimination des associations inconsistantes. Les plus grands pourcentages de clauses simplifiées sont trouvés dans les domaines qui contiennent peu de symétries. Par exemple, pour le domaine Rovers, les fluents représentent des entités qui sont souvent de types différents, et qui sont donc affectés de manières très différentes par les actions. Cela peut s'illustrer par des actions de la forme navigate(rover, x, y), qui ont un profil qui leur est propre, et qui sont peu nombreuses. Par conséquent, leurs domaines respectifs sont de tailles très réduites, ce qui

Domaine	SSI-H		
	CP	NoCP	Simp. Moy.
blocks	172	96	76.1%
gripper	210	189	74.9%
hanoi	74	75	0.2%
rovers	19	6	97.4%
satellite	34	22	79.1%
sokoban	204	0	98.6%
tsp	376	374	0.7%

Domaine	SSI		
	CP	NoCP	Simp. Moy.
blocks	166	93	76.2%
gripper	90	84	75.1%
hanoi	85	82	0.2%
rovers	16	6	97.3%
satellite	38	23	78.4%
sokoban	205	4	98.6%
tsp	265	266	1.0%

TABLE 1 – Nombre d'instances de SSI-H et SSI pour lesquelles notre algorithme termine en moins de 600 secondes. Pour chaque problème, les deux premières colonnes présentent le nombre d'instance de morphisme STRIPS qui ont pu être résolues avec et sans, respectivement, l'étape de pré-traitement basée sur une propagation des contraintes. La dernière colonne montre le pourcentage moyen de clauses qui ont pu être éliminées dans l'encodage en FNC φ, grâce à l'étape d'élimination des associations.

est très favorable à notre algorithme.
Au contraire, les domaines contenant un grand nombre de symétries ne profitent que très peu de l'élimination d'associations. C'est le cas du domaine Hanoi, pour lequel toutes les actions ont le même profil : à l'exception des informations données par l'état initial et le bug, tous les disques sont *a priori* interchangeables, ce qui ne permet pas à notre pré-traitement d'obtenir des résultats significatifs. Les seuls fragments d'informations à mêmes de guider la phase de recherche se retrouvent dans l'état initial, ce que nous pensons être un argument permettant d'expliquer la couverture légèrement supérieur de SSI, comparé à SSI-H.
Pour certaines instances de notre jeu de test, le pré-traitement est suffisant pour conclure qu'il n'existe aucun isomorphisme (homogène) de sous-instances. Cela se produit lorsqu'une variable voit son domaine réduit à l'ensemble vide. Dans ce cas, l'algorithme peut alors faire l'économie complète de la phase de recherche. Ainsi, notre algorithme est plus efficace lorsqu'il s'agit de détecter qu'il n'existe aucun isomorphisme (homogène) de sous-instances. C'est ainsi que notre étape d'élimination des associations inconsistantes nous permet d'augmenter grandement notre couverture des instances de morphismes STRIPS qui sont négatives, tandis que notre performances sur les instances positives est plus modeste, quoique significatif. Ces résultats sont compilés dans la Figure 1.
La Table 2 montre que le temps que prend l'étape de pré-traitement est négligeable devant le reste de l'algorithme. Plus précisément, que ce soit dans les domaines où elle élimine un grand nombre d'associations, ou dans les domaines où son efficacité est plus limitée, la propagation de contraintes prend rarement plus de quelques secondes. C'est ainsi que les instances où le pré-traitement permet de conclure sont résolues quasi-immédiatement.
On peut en outre relever, dans la Figure 1b, que résoudre les 500 instances négatives les plus faciles de SSI requiert une limite de temps à 10 minutes lorsque le pré-traitement n'est pas activé, alors que cette limite descend à moins d'une minute lorsque le pré-traitement est ajouté.
Dans la Table 3, nous présentons quelques résultats concernant les tailles absolues des problèmes que nous avons été en mesure de résoudre. A chaque instance de planification STRIPS $P = \langle F, I, O, G\rangle$, on associe une taille $|P| = |F| + |O|$. Dans la mesure où une instance de morphisme STRIPS a deux dimensions principales, que l'on représente par les tailles respectives des instances de planification qui la constituent, on présente deux manières différentes de mesurer la taille d'une instance de SSI.
Dans les trois premières colonnes de la Table 3, pour une instance de morphisme STRIPS donnée, on considère la somme des tailles des deux instances de planification constituant l'instance, et on présente l'instance maximisant cette somme. Cependant, avec cette mesure, P' est souvent de taille disproportionnée par rapport à P. Ce déséquilibre peut être expliqué par le fait que l'encodage en une formule propositionnelle est de temps et de taille $\mathcal{O}(|O| \cdot |O'| \cdot |F| \cdot |F'|)$, comme mentionné précédemment. Dans les deux dernières colonnes, on considère l'ordre lexicographique sur les couples $(|P|, |P'|)$. On représente, pour chaque problème, l'instance maximisant cette métrique.

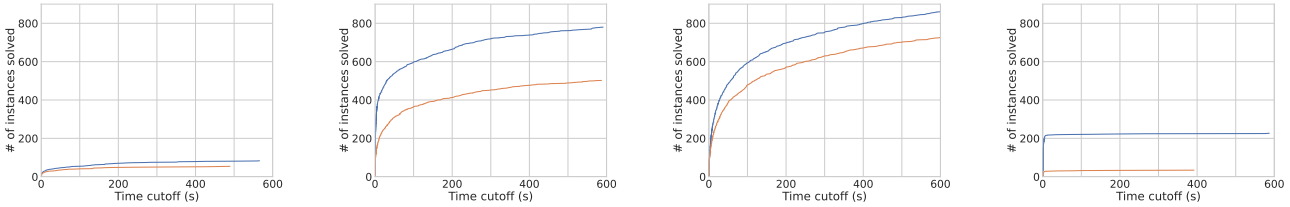
Domaine	SSI-H			
	CP	Comp.	Résol.	Temps total
blocks	0.5	93.3	76.8	170.3
gripper	0.2	23.5	9.8	33.4
hanoi	0.3	43.9	78.0	122.0
rovers	1.8	168.9	2.2	171.4
satellite	0.4	116.4	48.8	165.4
sokoban	1.7	222.7	2.3	225.2
tsp	0.2	50.7	46.7	97.5

Domaine	SSI			
	CP	Comp.	Résol.	Temps total
blocks	0.4	83.3	94.6	178.1
gripper	0.1	11.2	35.2	46.5
hanoi	0.3	70.9	47.8	118.9
rovers	1.7	180.7	2.7	183.5
satellite	0.4	85.0	10.3	95.4
sokoban	1.7	220.6	1.4	222.3
tsp	0.1	14.6	26.6	41.2

TABLE 2 – Temps moyen, en secondes, passé dans chacune des trois étapes principales de l'algorithme : pré-traitement (CP), compilation vers SAT, et résolution, respectivement. La dernière colonne résume le temps total moyen utilisé par l'algorithme. Nous ne présentons ici que les résultats pour les instances qui ont pu être décidées avec succès (positivement ou négativement) : les résultats pour SSI-H et SSI sont donc non-comparables.

7 Conclusion

Dans cet article, nous nous sommes tout d'abord intéressés au problème SI, qui traite de la synthèse d'un isomorphisme entre deux instances STRIPS, et nous avons montré sa GI-complétude. Ensuite, nous avons introduit la notion d'isomorphisme de sous-instance STRIPS, ainsi que les problèmes associés SSI et SSI-H. En sus de prouver la NP-complétude de ces deux problèmes, nous avons proposé un algorithme permettant de les résoudre, basé sur des techniques de propagation de contraintes, ainsi que sur une compilation vers SAT.
Notre évaluation expérimentale de cet algorithme a permis de montrer que des techniques classiques de propagation de contraintes, utilisées en pré-traitement, permettent d'améliorer grandement les performances d'un solveur SAT. Cela dit, bien que l'étape de pré-traitement en elle-même n'est que très peu coûteuse, tous les domaines de notre jeu de test n'en ont pas tiré le même parti.
De manière plus générale, notre travail fait office d'illustration d'un problème ouvert plus large, qui consiste à identifier quelles caractéristiques de problèmes de NP les rendent plus enclins à bénéficier de cette approche hybride entre programmation par contraintes et résolution SAT.



(a) Instances SSI positives (b) Instances SSI négatives (c) Instances SSI-H positives (d) Instances SSI-H négatives

FIGURE 1 – Nombre d'instances SSI et SSI-H que notre implémentation peut résoudre, en fonction du temps laissé à l'algorithme. La courbe bleue (resp. orange) correspond au cas où le pré-traitement est activé (resp. désactivé).

Domaine	SSI-H				
	Somme max.			$\|P\|$ max.	
	$\|P\|$	$\|P'\|$	Somme	$\|P\|$	$\|P'\|$
blocks	57	4642	4699	534	534
gripper	510	510	1020	510	510
hanoi	13	3328	3341	391	391
rovers	276	2667	2943	920	920
satellite	147	2066	2213	608	920
sokoban	2212	2286	4498	2212	2286
tsp	182	930	1112	462	462

Domaine	SSI-H				
	Somme max.			$\|P\|$ max.	
	$\|P\|$	$\|P'\|$	Somme	$\|P\|$	$\|P'\|$
blocks	57	4642	4699	534	534
gripper	510	510	1020	510	510
hanoi	13	6953	6966	391	513
rovers	276	2667	2943	920	920
satellite	147	2610	2757	608	920
sokoban	2212	2286	4498	2212	2286
tsp	90	930	1020	380	380

TABLE 3 – Tailles des plus grandes instances pouvant être résolues par notre implémentation, avec les contraintes spatiales et temporelles imposées, pour SSI-H et pour SSI. Dans les trois premières colonnes, on considère la somme des tailles des instances de planifications qui constituent le problème de morphisme STRIPS. Dans les colonnes suivantes, on considère la taille de P, plus petite instance parmi les deux formant l'instance de morphisme STRIPS.

Remerciements

Les auteurs tiennent à remercier les relecteurs anonymes de cet article, dont les commentaires nous ont permis d'améliorer la forme et le fond de ce papier.

Références

[1] Jérôme Amilhastre, Hélène Fargier, and Pierre Marquis. Consistency restoration and explanations in dynamic csps application to configuration. *Artif. Intell.*, 135(1-2) :199–234, 2002.

[2] László Babai. Group, graphs, algorithms : the graph isomorphism problem. In *Proceedings of the International Congress of Mathematicians*, pages 3319–3336. World Scientific, 2018.

[3] Djamila Baroudi, Maël Valais, and Frédéric Maris. Touistplan.

[4] Tom Bylander. The computational complexity of propositional strips planning. *Artificial Intelligence*, 69(1-2) :165–204, 1994.

[5] Stephen A. Cook. The complexity of theorem-proving procedures. In Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, editors, *3rd Annual ACM Symposium on Theory of Computing*, pages 151–158. ACM, 1971.

[6] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *J. Artif. Intell. Res.*, 17 :229–264, 2002.

[7] Richard Fikes and Nils J. Nilsson. STRIPS : A new approach to the application of theorem proving to problem solving. *Artif. Intell.*, 2(3/4) :189–208, 1971.

[8] Malte Helmert. Complexity results for standard benchmark domains in planning. *Artificial Intelligence*, 143(2) :219–262, 2003.

[9] Henry A. Kautz and Bart Selman. Planning as satisfiability. In Bernd Neumann, editor, *ECAI 92*, pages 359–363. John Wiley and Sons, 1992.

[10] Songtuan Lin and Pascal Bercher. Change the world - how hard can that be ? On the computational complexity of fixing planning models. In Zhi-Hua Zhou, editor, *IJCAI-21*, pages 4152–4159, 8 2021.

[11] Mao Luo, Chu-Min Li, Fan Xiao, Felip Manyà, and Zhipeng Lü. An effective learnt clause minimization approach for cdcl sat solvers. In *IJCAI-17*, pages 703–711, 2017.

[12] Alan K Mackworth. Consistency in networks of relations. *Artificial intelligence*, 8(1) :99–118, 1977.

[13] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of Constraint Programming*. Elsevier, 2006.

[14] Viktor N Zemlyachenko, Nickolay M Korneenko, and Regina I Tyshkevich. Graph isomorphism problem. *Journal of Soviet Mathematics*, 29(4) :1426–1481, 1985.

Recherche à Large Voisinage avec des Diagrammes de Décision

X. Gillard[1], P. Schaus[2]

[1] Université Catholique de Louvain, UCLouvain/ICTEAM/INGI

{xavier.gillard, pierre.schaus}@uclouvain.be

Résumé

La recherche locale est une technique largement utilisée afin de trouver rapidement de bonnes solutions à un problème d'optimisation. Afin de ne pas rester bloqué sur un minimum local, on utilise généralement des méta-heuristiques ou des techniques visant à explorer de larges voisinages de la meilleure solution courante. C'est ce qu'on appelle la recherche à large voisinage et qui est la base de ces travaux. Cet article propose un algorithme générique pour explorer de larges voisinages autour d'une solution donnée en utilisant des diagrammes de décision.

Mots-clés

Recherche Opérationelle, Optimisation combinatoire, Diagrammes de Décision,

Abstract

Local search is a well-known technique to solve combinatorial optimization problems efficiently. To escape local minima one generally uses metaheuristics or try to design large neighborhoods around the current best solution. A somewhat more black box approach consists in using an optimization solver to explore a large neighborhood. This is the large-neighborhood search (LNS) idea that we reuse in this work. We introduce a generic neighborhood exploration algorithm based on restricted decision diagrams (DD) constructed from the current best solution.

Keywords

Operations Research, Combinatorial Optimization, Decision Diagrams,

1 Introduction

La recherche locale est une approche largement utilisée pour obtenir rapidement de bonnes solutions à des problèmes d'optimisation combinatoire [HM09, HS04]. Malheureusement, une simple descente de gradient basée sur des petites perturbations comme des bi-intervertissements peut vite se trouver bloquée dans un minimum local. Dans ce cadre, les méta-heuristiques comme la recherche tabou ou le recuit-simulé peuvent permettre une certaine amélioration. Une autre alternative pour sortir des minima locaux consiste à explorer de grands voisinages autour de la meilleure solution connue dans l'espoir de trouver une solution encore meilleure. Ce faisant, le besoin de développer des méta-heuristiques fines et complexes se fait moins pressant, car la recherche est moins "myope". La construction de ces grands voisinages nécessite cependant souvent une grande expertise. Un exemple de ce type de voisinage qui permet d'obtenir d'excellents résultats sur les problèmes de routage de véhicules s'appelle le voisinage de Lin-Kernighan [LK73] et généralise les approches 2-opt et k-opt. Pour d'autres problèmes, des voisinages de taille exponentielle peuvent être explorés grâce à des algorithmes en temps polynomiaux tels les algorithmes de chemins ou flux maximaux [AEOP02]. On parle alors de voisinages *très larges*. Ceux-ci sont malheureusement souvent très fortement liés au problème qu'ils visent à résoudre et sont difficilement transposables, même dans le cas où le problème ne change que légèrement. Une approche plus générique basée sur la même idée utilise un solveur d'optimisation afin d'explorer ces voisinages larges [Sha98]. Le principal avantage de cette approche tient au fait que l'expertise de l'utilisateur peut se limiter à la modélisation du problème sans requérir une connaissance approfondie d'algorithmes nécessaires au développement de voisinages complexes. Cette approche alterne entre deux phases : (1) le relâchement d'une fraction des variables de décision et (2) l'assignation d'une valeur à ces variables à l'aide dudit solveur d'optimisation. Cette approche a été utilisée avec succès en combinant recherche locale et programmation par contrainte (PPC) pour la résolution de problèmes d'ordonnancement [LRSV18] et des problèmes de routage de véhicules [JVH11]. Ce type de techniques s'est aussi développé sous le nom de *branchement local* en utilisant des solveurs de programmation linéraire en nombres entiers (PLNE) [FL10]. Récemment, certains problèmes d'optimisation combinatoire ont été résolus de façon très efficace en utilisant des approches basées sur les diagrammes de décision [BCvHH16]. Des solveurs génériques basés sur celles-ci ont même été développés [GSC20]. Il est donc naturel de chercher à exploiter ce type de solveurs pour explorer de larges voisinages.

Contributions Cet article montre comment utiliser les idées développées dans [BCvHH16] afin d'explorer de larges voisinages autour d'une solution. L'efficacité de cette approche est étayée par une étude expérimentale portant sur deux problèmes d'optimisation sous contrainte à savoir le problème de séquençage de pigments (PSP) [GW99, Problème 58] et le problème du voyageur de commerce avec fenêtres de temps (PVC-FT) [GCLR20]. En dépit de sa simplicité et de son aspect générique, notre approche s'est avérée être compétitive vis-à-vis de l'état de l'art pour le

1er problème [PW06] et a permis de trouver de nouvelles meilleures solutions connues pour certaines instances du second [GCLR20].

2 Optimisation discrète

Un problème d'optimisation discrète est un problème de satisfaction auquel on a adjoint une fonction d'objectif à minimiser. Plus formellement, on dit qu'un problème d'optimisation $\mathcal{P}$ se définit comme $\min\{f(x) \mid x \in D \wedge C(x)\}$ dans lequel C est un ensemble de contraintes, $x = \langle x_0, \dots, x_{n-1}\rangle$ est un ensemble d'affectations de valeurs à des variables de décisions ; lesquelles sont chacune associées à un domaine D_i s.t. $D = D_0 \times \cdots \times D_{n-1}$ dont les valeurs sont tirées. La fonction $f : D \to \mathbb{R}$ est la fonction d'objectif à minimiser.

Parmi toutes les solutions réalisables $Sol(\mathcal{P}) \subseteq D$ (c-à-d qui satisfont toutes les contraintes de C), on dénote la *meilleure* solution par x^*. D'où, $x^* \in Sol(\mathcal{P})$ et $\forall x \in Sol(\mathcal{P}) : f(x^*) \leq f(x)$.

3 Programmation dynamique

La programmation dynamique (PD) a été introduite par Bellman au milieu des années 50 [Bel54]. Cette stratégie de résolution de problème est si populaire qu'elle constitue le coeur de nombre d'algorithmes classiques, notamment les algorithmes de Dijkstra [CLRS09, p.658] et Bellman-Ford [CLRS09, p.651].

Bien qu'on envisage souvent la programmation dynamique via le prisme de la récursion, il est aussi assez naturel de considérer ce modèle en le considérant comme un système de transitions labellisées. En adoptant cette perspective, le problème d'optimisation $\mathcal{P}$ est constitué de :

- un ensemble $S = \{S_0, \dots, S_n\}$ d'espaces d'états parmi lesquels on distingue l'état initial r, l'état terminal t et l'état infaisable $\perp$.
- un ensemble τ de fonctions de transitions t.q. $\tau_i : S_i \times D_i \to S_{i+1}$ emmenant le système d'un état s^i à l'état s^{i+1} suivant, pour tout $i = 0, \dots, n-1$; sur base de la valeur d affectée à la variable x_i. (Cette transition peut aussi amener à l'état $\perp$ si l'affectation $x_i = d$ viole une des contraintes de C). Ces fonctions ne doivent jamais permettre de "réparer" une infaisabilité [1] ($\tau_i(\perp, d) = \perp$) peu importe la valeur de $d \in D_i$).
- un ensemble de fonctions de couts de transitions $h_i : S_i \times D_i \to \mathbb{R}$ qui représentent le gain immédiat qu'il y a à affecter une valeur dßD_i à la variable x_i (pour $i = 0, \dots, n-1$).
- une valeur initiale v_r.

Sur cette base, on exprime la fonction d'objectif $f(x)$ du problème $\mathcal{P}$ comme :

$$\text{minimiser } f(x) = v_r + \sum_{i=0}^{n-1} h_i(s^i, x_i)$$

tel que

$$s^{i+1} = \tau_i(s^i, x_i) \text{ pour } i = 0, \dots, n-1; x_i \in D_i \wedge C(x_i)$$
$$s^i \in S_i \text{ pour } i = 0, \dots, n$$

où $C(x_i)$ est un prédicat qui est $vrai$ lorsqu'une affectation partielle $\langle x_0, \dots, x_i\rangle$ ne viole aucune contrainte de C.
L'attrait de ce type de formulation vient de sa simplicité et de son expressivité qui permet d'aisément capturer la structure du problème à résoudre. De plus, cette formulation se prête naturellement à une représentation sous forme de diagramme de décisions multivalué, lequel représente alors l'intégralité de $Sol(\mathcal{P})$.

4 Diagrammes de Décision

En toute généralité, un diagramme de décision (DD) est un type d'automate à couches t.q. un chemin entre la source et un nœud terminal passe par exactement un nœud de chaque couche. Dans cette structure, les libellés des arcs sont interprétés comme l'affectation d'une valeur à une variable donnée. De ce fait, le DD pris comme un tout peut être vu comme un encodage compact de l'ensemble des solutions d'un problème donné.
Formellement, un DD $\mathcal{B}$ est un graphe sans circuit à couches $\mathcal{B} = \langle n, U, A, l, d, v, \sigma\rangle$ où n est le nombre de variables du problème, U est l'ensemble des nœuds du graphe ; chacun desquels étant associé à un état $\sigma(u)$. Le partitionnement $l : U \to \{0 \dots n\}$ groupe les nœuds de U en couches disjointes $L_0 \dots L_n$ t.q. $L_i = \{u \in U : l(u) = i\}$ de nœuds appartenant aux mêmes espaces d'état ($\forall u \in L_i : \sigma(u) \in S_i$ pour $i = 0, \dots, n$). Aussi, les états de tous les nœuds appartenant à une même couche sont distincts ($\forall u_1, u_2 \in L_i : u_1 \neq u_2 \implies \sigma(u_1) \neq \sigma(u_2)$; $i = 0, \dots, n$).
L'ensemble $A \subseteq U \times U$ de ce modèle formel est un ensemble d'arcs qui joignent les nœuds de U. Un tel arc $a = (u_1, u_2)$ connecte des nœuds appartenant à deux couches successives ($l(u_1) = l(u_2) - 1$) et doit être vu comme la matérialisation d'une décision de branchement sur la variable $x_{l(u_1)}$. C'est pourquoi tous les arcs sont annotés avec libellés $d : A \to D$ et $v : A \to \mathbb{R}$ qui associent respectivement une décision et une valeur (poids) à un arc donné.

Exemple 1. *Un arc a connectant le nœud $u_1 \in L_3$ au nœud $u_2 \in L_4$, annoté avec $d(a) = 6$ et $v(a) = 42$ doit être comprise comme l'affectation $x_3 = 6$ effectuée depuis l'état $\sigma(u_1)$. Cela indique de plus que $\tau_3(\sigma(u_1), 6) = \sigma(u_2)$ et le coût de cette affectation est $v(a) = h_3(\sigma(u_1), 6) = 42$.*

Puisque chaque r-t chemin d'un DD $\mathcal{B}$ décrit une affectation satisfaisant le problème $\mathcal{P}$, on appellera $Sol(\mathcal{B})$ l'ensemble des solutions encodées dans $\mathcal{B}$. Aussi, puisqu'ils ne sont pas réparables, les chemins irréalisables sont omis du DD. Il s'ensuit que trouver la solution x^* dans un DD revient à

1. Au sens d'une condition de sureté

trouver le plus court r-t chemin dans $\mathcal{B}$ (selon les poids v des arcs).

DD Exact Étant donné un problème $\mathcal{P}$, un DD $\mathcal{B}$ est exact s'il encode exactement l'ensemble des solutions $Sol(\mathcal{B}) = Sol(\mathcal{P})$ de $\mathcal{P}$. En d'autres termes, non seulement tous les r-t chemin de $\mathcal{B}$ encodent des solutions réalisables de $\mathcal{B}$; mais toutes les solutions de $\mathcal{P}$ sont aussi présentes dans $\mathcal{B}$. Un DD exact pour le problème $\mathcal{P}$ peut être compilé via une procédure de haut en bas qui découle naturellement de la définition ci-dessus. Pour ce faire, il suffit de "dérouler" la relation de transition de façon répétée jusqu'à ce que toutes les variables aient reçu une valeur (Algorithme 1).

Algorithme 1 : Compilation d'un DD exact de haut en bas

```
1  Entrée : un PD P = ⟨S, r, t, ⊥, v_r, τ, h⟩ ;
2  L_0 ← {r} ;
3  pour i ∈ {0 ... n − 1}, u ∈ L_i, d ∈ D_i faire
4      u' ← τ_i(σ(u), d) ;
5      si u' ≠ ⊥ alors
6          L_{i+1} ← L_{i+1} ∪ {u'} ;
7          a ← (u, u') ;
8          v(a) ← h_i(σ(u), d) ;
9          d(a) ← d ;
10         A ← A ∪ a ;
```

DD Restreint Bien que leur encodage soit compact, la construction des DD peut requérir une quantité de mémoire exponentielle dans le pire des cas. De ce fait, il n'est souvent pas raisonnable de chercher à encoder l'ensemble complet des solutions d'un problème dans un DD exact. C'est pourquoi des approximations ont été inventées pour ces DD exacts; lesquelles garantissent de n'utiliser qu'une quantité de mémoire *bornée*. Cet article n'utilise qu'une seule variante de ces approximations de taille bornée : les *DD restreints*. L'Algorithme 2 montre comment compiler un tel DD restreint. En pratique, il s'agit d'insérer un appel à une procédure de bornage de la largeur du DD. Celle-ci (qu'on appellera `restreindre`) garantit que la largeur (le nombre $|L_i|$ de nœuds distincts au sein de la couche L_i) de L_i n'excède pas une limite W fixée à priori. Pour ce faire, `restreindre` conserve les nœuds les plus prometteurs (selon une heuristique) et supprime les autres. Les choix posés par la fonction `restreindre` sont d'un importance évidente car ce sont eux qui déterminent les nœuds – et par là même les chemins = les solutions représentées – qui seront présentes dans le DD compilé.

Formellement, si le DD exact $\mathcal{B}$ encode l'ensemble des solutions du problème $\mathcal{P}$, son homologue restreint $\overline{\mathcal{B}}$ en encode un sous-ensemble $Sol(\overline{\mathcal{B}}) \subseteq Sol(\mathcal{B})$. Par conséquent, le plus court r-t chemin de $\overline{\mathcal{B}}$ est aussi une solution du problème $\mathcal{P}$ et sa longueur donne une borne supérieure sur la valeur de l'optimum x^*.

Borne grossière Les bornes grossières ont récemment été proposées par [GCSC21] afin d'accélérer la compilation des DD et de resserrer les bornes dérivables d'approximations de taille bornée. L'intuition qui sous-tend cette technique est la suivante : en supposant que l'on connaisse une borne supérieure $\overline{v}$ sur la valeur de la solution optimale, et en supposant qu'on soit aussi capable de rapidement calculer une borne inférieure $\underline{v_s}$ (assez grossière, mais facile à calculer) sur la valeur du plus court r-t chemin passant par un nœud s donné; on peut décider d'ignorer le développement de s et tous ses descendants lorsque $\underline{v_s} \geq \overline{v}$. C'est-à-dire, lorsqu'on a la garantie qu'aucun chemin passant par s ne pourra améliorer l'objectif global.

L'Algorithme 2 montre comment calculer un DD restreint en utilisant une borne grossière. En pratique il suffit d'inclure à cet effet, un test $grossiere(u') \leq f(s^*)$ permettant d'ignorer le développement d'un nœud u' lorsque sa borne grossière ne permet pas d'améliorer la meilleure solution connue. Le reste de cet article utilise `CompilerDDRestreint(`$\mathcal{P}$`, `s^*`)` pour signifier une invocation de l'Algorithme 2 où $\mathcal{P}$ et s^* dénotent respectivement le (sous-)problème considéré et sa meilleure solution actuellement connue.

Algorithme 2 : Compil. d'un DD restreint de haut en bas

```
1  Entrée : un PD P = ⟨S, r, t, ⊥, v_r, τ, h⟩ ;
2  L_0 ← {r} ;
3  pour i ∈ {0 ... n − 1} faire
4      pour u ∈ L_i, d ∈ D_i faire
5          u' ← τ_i(σ(u), d) ;
6          si u' ≠ ⊥ ∧ grossiere(u') ≤ f(s*) alors
7              L_{i+1} ← L_{i+1} ∪ {u'} ;
8              a ← (u, u') ;
9              v(a) ← h_i(σ(u), d) ;
10             d(a) ← d ;
11             A ← A ∪ a ;
12     si |L_{i+1}| > W alors
13         L_{i+1} ← restreindre(L_{i+1}) ;
```

5 Recherche à Large Voisinage

Comme cela a été dit dans l'introduction, la recherche à large voisinage (RLV) est une méthode d'optimisation incomplète qui vise à permettre un échappement des minima locaux. Le tout sans nécessiter la maitrise de méta heuristiques avancées et complexes. Pour ce faire, la RLV tente d'établir un équilibre entre l'intensification (utiliser des algorithmes d'inférence avancés) et la diversification (explorer des voisinages différents). C'est pourquoi, partant d'une solution initiale s^*, la RLV oscille entre deux phases. La première consiste à relâcher une petite partie des décisions faites dans s^*. La seconde phase tente alors de réoptimiser le sous-problème résiduel à l'aide d'un solveur utilisé comme un oracle. Dès que celui-ci identifie une solution s'^* améliorant l'objectif ($f(s'^*) < f(s^*)$); la meilleure solution courante est mise à jour.

Notre Approche Nous proposons d'utiliser des DD restreints pour explorer des ensembles de solutions dans un grand voisinage de s^*. L'Algorithme 3 montre comment

procéder. À l'instar d'une RLV traditionnelle, notre méthode doit trouver un équilibre entre intensification et diversification. En l'occurrence, l'objectif de diversification est atteint via la compilation d'un DD restreint (Algorithme 3 lignes 10-11).

Algorithme 3 : RLV avec des DD

1 **Entrée :** un PD $\mathcal{P} = \langle S, r, t, \bot, v_r, \tau, h \rangle$;
2 **Entrée :** $s^* \leftarrow$ la meilleure solution connue, ou rien ;
3 $ProfondeurMax \leftarrow |S| - 2$;
4 $d \leftarrow ProfondeurMax$;
5 **tant que** *critere de fin non satisfait* **faire**
6 $r' \leftarrow r$;
7 **si** $s^* \neq rien$ **alors**
8 $r' \leftarrow s^*_d$;
9 $\mathcal{P}' \leftarrow \langle S, r', t, \bot, v(r'), \tau, h \rangle$;
10 $voisinage \leftarrow CompilerDDRestreint(\mathcal{P}', s^*)$;
11 $voisin \leftarrow PlusCourtChemin(voisinage)$;
12 **si** $f(voisin) < f(s^*)$ **alors**
13 $s^* \leftarrow voisin$;
14 $d \leftarrow ProfondeurMax$;
15 **sinon si** $d = 0$ **alors**
16 $d \leftarrow ProfondeurMax$;
17 **sinon**
18 $d \leftarrow d - 1$;

Notre méthode met en oeuvre trois mécanismes d'intensification. Le premier consiste en l'utilisation (optionnelle) d'une procédure de bornage grossière afin d'éliminer les nœuds qui ne peuvent amener aucune amélioration. Les second et troisième mécanismes découlent du comportement de la procédure `restreindre`. Comme le montre l'Algorithme 4, elle effectue une première partition des nœuds de la couche L_i afin de séparer ceux qui *doivent* être conservés des autres (Algorithme 4 lignes 5-11). Cette décision se base sur le prédicat $doitGarder$ (Definition 1), lequel exprime qu'un nœud n de la i^{eme} couche doit être gardé si la valeur affectée à la variable x_i du meilleur r-n chemin (noté $p^*_{r-n}(i)$) est la même que la valeur affectée à x_i dans s^* (Algorithme 3 ligne 10). Ceci permet de garantir que la meilleure solution connue ne sera *jamais* supprimée du DD restreint.

Définition 1.

$$doitGarder(n, s^*, i) \iff p^*_{r-n}(i) = s^*(i)$$

Notre dernier mécanisme d'intensification tient à l'utilisation de la borne grossière associée à chaque nœud comme heuristique pour sélectionner les autres nœuds qui pourront rester dans la couche après restriction. Ceci est visible aux lignes 12-13 de l'Algorithme 4.
Il y a deux mécanismes qui interviennent afin d'assurer une diversification suffisante à notre méthode. Le premier consiste à sélectionner une racine différente pour la compilation de chaque DD restreint. Cette racine est choisie de façon systématique en choisissant opportunément une racine en bas du DD ayant fourni la meilleure solution. Puis en progressant vers les niveaux supérieurs (Algorithme 3 lignes 6-9, 14-18). Le second mécanisme consiste à choisir quelques nœuds ne satisfaisant pas le prédicat $doitGarder$ de façon aléatoire, et décider de les ranger dans la partition des nœuds qui *doivent* être gardés avec une faible probabilité (ligne 7).

Algorithme 4 : Restrict Procedure

1 **Entrée :** L_i : la couche à restreindre ;
2 **Entrée :** s^* : la meilleure solution connue ou rien ;
3 **Entrée :** W : la largeur de couche maximale ;
4 **Entrée :** p : une faible probabilitié (p/ex. 10%) ;
5 $frontiere \leftarrow 0$;
6 **pour** $k \in \{0..|L_i|\}$ **faire**
7 **si** $doitGarder(L_i[k], s^*, i) \vee hasard() \leq p$ **alors**
8 $echanger(L_i, frontiere, k)$;
9 $frontiere \leftarrow 1 + frontiere$;
10 $garder \leftarrow noeuds[0..frontiere[$;
11 $candidats \leftarrow noeuds[frontiere..|noeuds|[$;
12 $trier(candidats$ sur la base de leur borne grossière) ;
13 $tronquer(candidats, \max(0, W - |garder|)$;
14 $L_i \leftarrow concatener(garder, candidats)$;

Points forts Il y a plusieurs points forts à notre approche : les DD permettent de s'appuyer sur un PD afin d'explorer de nouveaux voisinages. De plus, contrairement à la RLV classique, notre approche est parfois capable de prouver l'optimalité des instances qu'elle résout ; ce qui n'est habituellement possible qu'en utilisant des méthodes exactes t.q. la programmation linéaire en nombres entiers ou un système de séparation et évaluation[BCvHH16]. En effet, notre méthode génère des ensembles de solutions complètes à chaque itération. Et, étant donné que la valeur de la meilleure solution connue s'améliore avec le temps, il en va de même pour la force d'élagage des bornes grossières. De ce fait, il est possible qu'au fil du temps, cette force d'élagage devienne suffisante pour compiler le DD sans avoir recours à la procédure `restreindre`. Dans ce cas, le DD compilé est un DD exact, ce qui suffit à prouver l'optimalité de la meilleure solution qu'il contient. Naturellement, cette capacité résulte d'un compromis entre la force d'élagage des bornes grossières et la largeur du DD compilé. C'est pourquoi il n'est pas toujours possible d'arriver à une preuve d'optimalité. Nous pensons néanmoins que cette possibilité est suffisamment remarquable pour qu'elle soit mentionnée à propos d'une approche incomplète.

Un autre point fort de notre méthode tient à l'aspect configurable de la compilation des DD. Ce qui signifie qu'il est possible de choisir la largeur des DDs et par là, le temps nécessaire à les compiler ; arbitrant de ce fait un compromis entre la diversification et l'intensification.

6 Étude expérimentale

Nous avons utilisé deux problèmes d'ordonnancement afin d'évaluer l'efficacité de notre méthode de RLV avec DD : un problème de dimensionnement et ordonnancement de lots discrets (PSP) ainsi que le problème du voyageur de commerce avec fenêtres temporelles (PVC-FT).

PSP Le problème du séquençage de pigments est un problème d'ordonnancement de lots multi-items sous contrainte de capacité. Il est décrit dans la CSPLib [GW99, Problème 58] et est étudié en détail dans [PW06]. C'est un problème de planification de production dans lequel un item doit être produit sur une machine à chaque pas de temps afin de satisfaire la demande pour chacun des produits tout en minimisant les couts de stockage et de reconfiguration de la machine. Le PSP est caractérisé par un 5-tuple $\langle \mathcal{I}, \mathcal{H}, \mathcal{S}, \mathcal{C}, \mathcal{Q} \rangle$ où :

- $\mathcal{I} = \{0, \dots, n-1\}$ est l'ensemble des items à produire,
- $\mathcal{H}$ est l'horizon temporel,
- $\mathcal{S}$ est un vecteur de couts de stockage t.q. $\mathcal{S}_i$ est le cout à payer pour stocker une unité de type i pendant un pas de temps.
- $\mathcal{C}$ est une matrice de couts de reconfiguration t.q. $\mathcal{C}_{i,j}$ est le cout à payer pour passer de la production d'un item de type i à un item de type j.
- $\mathcal{Q}$ est un vecteur de demande par item. Étant donné une demande $0 \leq t < \mathcal{H}$ et un item $i \in \mathcal{I}$, on utilise $\mathcal{Q}_t^i$ pour dénoter le nombre d'items de type i qui doivent être livrées au temps t. Les sections qui suivent supposent que les demandes sont normalisées : $\mathcal{Q}_t^i \in \{0, 1\}\, \forall t, i$.

Programmation Linéaire en Nombres Entiers Les équations (1) – (6) donnent une formulation du PSP sous forme de programme en nombres entiers. En accord avec la terminologie de [PW06], ce modèle est appelé PIG-A-1.

$$\textbf{minimiser} \sum_{i,j,t} \mathcal{C}_{i,j} \mathbf{c}_{i,j}^t + \sum_{i,t} \mathcal{S}_i \mathbf{s}_i^t \tag{1}$$

tel que

$$\mathbf{s}_i^0 = 0 \;\; \forall i \in \mathcal{I} \tag{2}$$

$$\mathbf{x}_i^t + \mathbf{s}_i^{t-1} = \mathcal{Q}_t^i + \mathbf{s}_i^t \;\; \forall i \in \mathcal{I}; 0 \leq t < \mathcal{H} \tag{3}$$

$$\mathbf{x}_i^t \leq \mathbf{y}_i^t \;\; \forall i \in \mathcal{I}, 0 \leq t < \mathcal{H} \tag{4}$$

$$\sum_{i,t} \mathbf{y}_i^t = 1 \;\; \forall i \in \mathcal{I}, 0 \leq t < \mathcal{H} \tag{5}$$

$$\mathbf{c}_{i,j}^t \geq \mathbf{y}_i^{t-1} + \mathbf{y}_j^t - 1 \;\; \forall i, j \in \mathcal{I}; 0 \leq t < \mathcal{H} \tag{6}$$

où $\mathbf{x}_i^t$ est une variable de production binaire (1 lorsqu'on produit un item de type i au temps t, 0 sinon). $\mathbf{y}_i^t$ est une variable binaire d'installation (1 ssi la machine est prête à produire un item de type i au temps t). $\mathbf{c}_{i,j}^t$ est une variable de reconfiguration (1 ssi la configuration de la machine a changé de i vers j au temps t). $\mathbf{s}_i^t$ est une variable entière de comptant le nombre d'items de type i stockés au temps t.

Dans ce modèle, (1) est la fonction d'objectif à minimiser. (2) est une contrainte imposant que le stock de chaque type d'item soit vide au démarrage. L'équation (3) est une contrainte de *conservation* indiquant qu'un item est soit stocké soit livré une fois qu'il a été produit. La contrainte (4) impose une consistance entre la production et les variables de configuration de la machine. (5) est une contrainte de *capacité* imposant qu'au plus une unité d'un seul type d'item soit produite par pas de temps. Enfin, (6) est une contrainte forçant la consistance entre les variables d'installation $(\mathbf{y}_i^{t-1}, \mathbf{y}_j^t)$ et les variables de reconfiguration $(\mathbf{c}_{i,j}^t)$.

Programme Dynamique[2] Afin de formuler un PD pour PSP, il est utile de définir $\mathcal{T}_t^i$ sur base des données du problème. $\mathcal{T}_t^i$ exprime le moment où la demande *précédente* d'un item de type i doit avoir été satisfaite (pour un item de type i et un moment $0 \leq t \leq \mathcal{H}$).

$$\mathcal{T}_0^i = -1 \;\Big|\; \mathcal{T}_t^i = \begin{cases} t-1 & \text{si } \mathcal{Q}_i^{t-1} > 0 \\ \mathcal{T}_{t-1}^i & \text{sinon} \end{cases}$$

Les éléments du PD pour PSP sont les suivants :

- un état $s^t \in \mathcal{S}_{\mathcal{H}-t}$ est un tuple $\langle k, u \rangle$ dans lequel k représente le type d'item produit au temps $t+1$, et u est un vecteur reprenant la date de livraison précédente pour chaque type d'item. On a donc : $r = \left\langle -1, \left(\mathcal{T}_{\mathcal{H}}^0, \mathcal{T}_{\mathcal{H}}^1, \dots, \mathcal{T}_{\mathcal{H}}^{n-1}\right)\right\rangle$
- $\tau_t(\langle k, u \rangle, d) = \begin{cases} \left\langle d, \left(u_0, \dots u_{d-1}, \mathcal{T}_{u_d}^d, u_{d+1}, \dots u_{n-1}\right)\right\rangle & \\ \qquad\qquad \text{lorsque } u_d \geq t & \\ \bot \qquad\qquad \text{sinon} & \end{cases}$
- $h_t(\langle k, u \rangle, d) = \begin{cases} \mathcal{S}_d \cdot (u_d - t) & \text{si } k = -1 \\ \mathcal{C}_{k,d} + \mathcal{S}_d \cdot (u_d - t) & \text{sinon} \end{cases}$
- $v_r = 0$.

Borne grossière Puisque le PSP est un problème de Wagner-Within (WW) [PW06] en l'absence des couts de reconfiguration, une borne grossière pour un état $s^t = \langle k, u \rangle \in \mathcal{S}_{\mathcal{H}-t}$ est donnée par le cout WW de s^t plus le poids total d'un plus petit arbre couvrant l'ensemble des couts de reconfiguration entre les items i t.q. $u_i \geq 0$.

PVC-FT Le PVC-FT est une variante du PVC dans laquelle les clients du voyageur de commerce ne peuvent être visités que pendant des fenêtres de temps données. Ce problème est connu pour être difficile à résoudre. En effet, il a été prouvé que la satisfaisabilité de ce problème est NP-complète [Sav85]. Formellement, le PVC-FT est caractérisé par le nombre N de clients à visiter, $\mathcal{D}$ la matrice carrée de distance entre les différents clients t.q. $\mathcal{D}_{i,j}$ dénote la distance entre les clients i et j ; $\mathcal{H}$ est l'horizon temporel considéré et $\mathcal{TW}$ est le vecteur de fenêtre de temps t.q. $\mathcal{TW}_i = (e_i, l_i)$ où e_i correspond au moment auquel le voyageur peut visiter le client i au plus tôt et l_i le dernier moment auquel cette visite peut avoir lieu.

2. Pour autant que nous sachions, ce problème n'a jamais été résolu via la programmation dynamique avant cet article. Ceci est donc une contribution supplémentaire de notre article

Modèle de PPC Nous fournissons ci-dessous un modèle de PPC avec Minizinc pour le PVC-FT. La valeur de la variable x_i indique le client qui est visité en i^{eme} position du tour. Les variables auxiliaires a_i représentent les moments auxquels le voyageur de commerce visite le i^{eme} client du tour. Les contraintes assurent i) que le tour du voyageur de commerce démarre et se termine au dépôt ii) que chaque client soit visité exactement une fois iii) que les fenêtres de temps soient respectées et iv) que le voyageur ne puisse pas de déplacer plus rapidement que spécifié dans la matrice de distance (mais il a le droit d'attendre). Enfin, on minimise l'objectif de temps de parcours.

```
% Le tour doit commencer/terminer dans la ville 0
constraint (x[0] = 0 /\ x[N] = 0 /\ a[0] = 0);
constraint alldifferent_except_0(x);
% On force le respect des fenetres de temps
constraint forall(i in 0..N)(PlusTot[x[i]] <= a[x[i]]);
constraint forall(i in 0..N)(a[x[i]] <= PlusTard[x[i]]);
constraint forall(i in 0..N-1)(
  a[x[i+1]] >= a[x[i]] + Distance[x[i],x[i+1]]
);
% Objectif de temps de trajet a minimiser
int: trajet = sum(i in 0..N-1)(Distance[x[i],x[i+1]]);
solve minimize trajet;
```

Programme Dynamique De nombreux modèles de PD ont déjà été proposés pour le PVC-FT [Sav85, DDGS95, MBR97]. Dans cet article, il est modélisé comme suit :

— un état $s^t \in \mathcal{S}_t$ est un tuple $\langle t, c, \rho \rangle$ où t denote le pas de temps actuel, c donne la position courante du voyageur de commerce et ρ est l'ensemble des clients qui doivent encore être visités. En particulier, on a $r = \langle 0, 0, \{1 \dots N-1\} \rangle$.

— $\tau_t(\langle t, c, \rho \rangle, d) = \begin{cases} \langle \max(e_d, t + \mathcal{D}_{c,d}),\ d,\ \rho \setminus \{d\} \rangle & \\ \qquad \text{quand } l_d \geq t + \mathcal{D}_{c,d} & \\ \bot \qquad \text{sinon} & \end{cases}$

— $h_t(\langle t, c, \rho \rangle, d) = \mathcal{D}_{c,d}$

— $v_r = 0$

Borne grossière Une borne inférieure – simple, mais néanmoins effective – sur la valeur optimale atteignable depuis un état $s^t = \langle t, c, \rho \rangle$ est donnée par la somme de $\min\{D_{c,o} \mid o \in \rho\}$, du cout d'un arbre minimum couvrant la distance entre les clients de ρ, et $\min\{D_{o,0} \mid o \in \rho \wedge D_{o,0} \leq l_0\}$. Intuitivement, ces trois termes représentent respectivement la plus courte distance entre la position courante du voyageur et n'importe quel autre client à visiter ; une estimation du plus petit temps de parcours possible entre les clients encore à visiter et la plus courte distance jusqu'au dépôt depuis n'importe quel client encore à visiter dont la fenêtre de temps permet qu'il soit visité juste avant le retour au dépôt.

6.1 Protocole expérimental

Toutes nos expériences ont été réalisées sur la même machine physique, laquelle est équipée de deux processeurs Intel(R) Xeon(R) CPU E5-2687W v3 et 128Go de mémoire vive. Sur cette machine, on a donné à chaque solveur un budget temporel de 10 minutes d'exécution sur un seul fil et un quota de mémoire utilisable de 2Go maximum pour résoudre chaque instance.

PSP Nous avons utilisé notre propre implémentation d'un cadre générique de résolution de problèmes via RLV-DD et y avons branché le PD décrit ci-dessus. Nous avons comparé l'efficacité du solveur ainsi obtenu avec celle de l'état de l'art pour la résolution de ce problème (modèle PLNE nommé PIG-A-3 dans [PW06]). Étant donné la simplicité de notre modèle, et puisque PIG-A-3 a bénéficié des améliorations amenées par des experts du sujet durant plus d'une dizaine d'années ; nous avons aussi inclus les modèles PIG-A-1 et PIG-A-2 dans notre comparaison. Ceux-ci devraient donner une idée de ce qu'un modélisateur pourrait raisonnablement produire comme modèle pour le PSP. Les trois modèles PLNE, proviennent directement de [PW06] et sont programmés et résolus avec FICO Xpress Mosel v8.11. Nos expériences portent sur 500 instances générées aléatoirement qu'il était impossible de résoudre en utilisant uniquement un PD ou une approche brancher-et-borner à base de DD [BCvHH16, GCSC21]. Ces 500 instances ont $|\mathcal{I}| = 10$, $\mathcal{H} \in \{50, 100\}$. Les valeurs de leurs $\mathcal{S}_i$ et $\mathcal{C}_{i,j}$ varient entre 10 et 50.

PVC-FT Les expériences que nous avons réalisées à propos du PVC-FT utilisent le même cadre générique de celles pour PSP ; et se basent sur le PD décrit ci-dessus. Nous avons comparé l'efficacité du solveur ainsi obtenu avec une implémentation du modèle par PPC de la section 6 dans Choco 4.10.6 [3]. Ce dernier a été configuré pour permettre la ré-optimisation de petites séquences de 5 variables de décision $x_i \dots x_{i+4}$ choisies de façon aléatoires à chaque redémarrage [4].

Nos expériences portent sur les 467 instances des suites de référence généralement utilisées pour évaluer la performance de nouveaux solveurs pour le PVC-FT.

Résultats La Table 1 montre pour chaque solveur, le nombre d'instances du PSP dont la meilleure solution connue a été identifiée. Elle montre aussi le nombre d'instances pour lesquelles l'objectif de la meilleure solution trouvée se trouvait à moins d'1% de la meilleure solution connue ($\frac{trouve - meilleure\_connue}{meilleure\_connue} \leq 1\%$). Cette table nous montre assez clairement que l'approche RLV-DD est très efficace pour trouver de bonnes solutions. En effet, cette méthode surclasse toujours les modèles PIG-A-1 et PIG-A-2, même lorsque la largeur maximale d'une couche (des DD compilés) est limitée à 10 nœuds seulement. De plus, bien que le PD sous-jacent soit très simple, l'efficacité de notre approche est comparable à celle qu'on peut obtenir avec le modèle PIG-A-3 qui est nettement plus complexe.

Comme la satisfaisabilité du PVC-FT est NP-complete et afin d'établir une comparaison honnête entre RLV-PPC et RLV-DD ; nous avons initialisé la résolution de tous les solveurs avec une solution faisable calculée hors ligne (la même pour toutes les techniques). Ces solutions initiales ont été obtenues à l'aide d'une variante de l'algorithme de Da Silva et Urrutia [DSU10]. La Table 2 montre pour chaque solveur, le nombre d'instances pour lesquelles l'ob-

3. https ://choco-solver.org/

4. Plusieurs relaxations alternatives ont été testées, et ces paramètres sont ceux qui donnaient les résultats les plus favorables à la PPC.

Méthode	W	Meill. Sol.	1% de la meilleure sol.
PIG-A-1	N.A.	35	232
PIG-A-2	N.A.	97	226
PIG-A-3	N.A.	475	499
RLV-DD	10	167	449
RLV-DD	100	276	489
RLV-DD	1000	368	490

TABLE 1 – Nombre d'instances PSP pour lesquelles la meilleure solution connue a été identifiée et pour lesquelles l'objectif de la meilleure solution trouvée est à moins d'un pourcent de la meilleure solution connue.

Méthode	W	Meill. Sol.	1% de la meilleure sol.
RLV-PPC	N.A.	144	184
RLV-DD	10	197	246
RLV-DD	100	217	261
RLV-DD	1000	217	249

TABLE 2 – Nombre d'instances PVC-FT pour lesquelles l'objectif de la meilleure solution trouvée correspond à celui de la meilleure solution publiée ; ainsi que le nombre d'instances pour lesquelles l'objectif de la meilleure solution trouvée est à moins d'un pourcent du meilleur objectif connu.

jectif de la meilleure solution trouvée correspond à celui de la meilleure solution connue (publiée). De plus, elle montre le nombre d'instances pour lesquelles l'objectif de la meilleure solution trouvée se trouve à moins d'un pourcent de l'objectif de la meilleure solution connue. Cette table montre que les deux méthodes sont très efficaces pour trouver de bonnes solutions pour le PVC-FT ; avec un léger avantage en faveur de RLV-DD par rapport à RLV-PPC. Durant cette phase de nos expériences, nous avons identifié 75 nouvelles solutions ayant un objectif égal à celui de la meilleure solution publiée.

Afin d'évaluer la capacité de ces méthodes à optimiser le PVC-FT indépendamment de la solution initiale, nous avons répété l'expérience en initialisant cette fois les solveurs avec la meilleure solution publiée. Les deux techniques ont été capables de trouver de nouvelles solutions améliorant l'objectif par rapport aux meilleures solutions connues. En pratique, RLV-DD a permis d'identifier 8 nouvelles solutions améliorantes dans deux suites de référence (AFG et OhlmannThomas) et RLV-PPC a permis d'en identifier 10 dans la suite de référence OhlmannThomas.

Il est intéressant de noter que nos résultats expérimentaux indiquent que la largeur maximale W des DD compilés permet de facilement configurer le niveau de diversification de la recherche. En effet, on observe aussi bien sur la Table 1 que la Table 2 qu'accroitre la largeur maximum W permet d'améliorer la performance du solveur (jusqu'à un certain point où cela devient contre-productif).

7 Travaux Connexes

Notre approche peut être considérée comme une hybridation des techniques d'optimisation à base de DD (ODD), de recherche par faisceaux, RLV et de l'heuristique de sauvegarde de phase couramment utilisée dans les solveurs SAT [PD07]. D'autres ont récemment proposé des techniques qui combinent certains de ces ingrédients. Toutefois, et pour autant que nous sachions, aucune approche n'a jamais proposé de les combiner tous. Par exemple, [LIB10] propose une hybridation de la recherche par faisceaux avec une optimisation par colonie de fourmis afin de résoudre le PVC-FT. Contrairement à notre approche, cette technique se base sur une optimisation par colonie de fourmis plutôt qu'un programme dynamique.

Les travaux très récents [DCS18] et [BFP$^+$20] cherchent à atteindre un objectif similaire au nôtre. Pour ce faire, ils combinent programmation par contrainte, sauvegarde de phase et RLV en vue de résoudre des problèmes combinatoires difficiles. Leurs travaux se concentrent toutefois sur des propagateurs alors que nos travaux cherchent uniquement à exploiter une formulation du problème sous forme de programme dynamique.

8 Conclusions

Dans cet article, nous avons proposé et évalué une méthode qui combine la recherche à larges voisinages avec des diagrammes de décisions afin de résoudre des problèmes d'optimisation combinatoire complexes formulables comme des programmes dynamiques. La simplicité et les bonnes performances (évaluées sur deux problèmes) de cette technique peuvent être appréciables pour un praticien. En particulier lorsqu'il est nécessaire d'être capable de prendre rapidement de bonnes décisions ; comme par exemple lorsqu'il faut adapter un planning de production sur base d'un carnet de commandes en constante évolution.

Références

[AEOP02] Ravindra K Ahuja, Özlem Ergun, James B Orlin, and Abraham P Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1-3) :75–102, 2002.

[BCvHH16] David Bergman, Andre A. Cire, Willem-Jan van Hoeve, and J. N. Hooker. Discrete optimization with decision diagrams. *INFORMS Journal on Computing*, 28(1) :47–66, 2016.

[Bel54] Richard Bellman. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6) :503–515, 11 1954.

[BFP$^+$20] Gustav Björdal, Pierre Flener, Justin Pearson, Peter J Stuckey, and Guido Tack. Solving satisfaction problems using large-neighbourhood search. In *International Conference on Principles and Practice of Constraint Programming*, pages 55–71. Springer, 2020.

[CLRS09] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[DCS18] Emir Demirović, Geoffrey Chu, and Peter J Stuckey. Solution-based phase saving for cp : A value-selection heuristic to simulate local search behavior in complete solvers. In *International Conference on Principles and Practice of Constraint Programming*, pages 99–108. Springer, 2018.

[DDGS95] Yvan Dumas, Jacques Desrosiers, Eric Gelinas, and Marius M Solomon. An optimal algorithm for the traveling salesman problem with time windows. *Operations research*, 43(2) :367–371, 1995.

[DSU10] Rodrigo Ferreira Da Silva and Sebastián Urrutia. A general vns heuristic for the traveling salesman problem with time windows, 2010.

[FL10] Matteo Fischetti and Andrea Lodi. Heuristics in mixed integer programming. *Wiley Encyclopedia of Operations Research and Management Science*, 2010.

[GCLR20] Jaime E Gonzalez, Andre A Cire, Andrea Lodi, and Louis-Martin Rousseau. Integrated integer programming and decision diagram search tree with an application to the maximum independent set problem. *Constraints*, pages 1–24, 2020.

[GCSC21] X. Gillard, V. Coppé, P. Schaus, and A. Ciré. Improving the filtering of branch-and-bound mdd solvers, 2021.

[GSC20] X. Gillard, P. Schaus, and V. Coppé. Ddo, a generic and efficient framework for mdd-based optimization. International Joint Conference on Artificial Intelligence (IJCAI-20) ; DEMO track, 2020.

[GW99] Ian P Gent and Toby Walsh. CSPLib : a benchmark library for constraints. In *International Conference on Principles and Practice of Constraint Programming*, pages 480–481. Springer, 1999.

[HM09] Pascal Van Hentenryck and Laurent Michel. *Constraint-based local search*. 2009.

[HS04] Holger H Hoos and Thomas Stützle. *Stochastic local search : Foundations and applications*. Elsevier, 2004.

[JVH11] Siddhartha Jain and Pascal Van Hentenryck. Large neighborhood search for dial-a-ride problems. In *International Conference on Principles and Practice of Constraint Programming*, pages 400–413. Springer, 2011.

[LIB10] Manuel López-Ibáñez and Christian Blum. Beam-aco for the travelling salesman problem with time windows. *Computers & operations research*, 37(9) :1570–1583, 2010.

[LK73] Shen Lin and Brian W Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2) :498–516, 1973.

[LRSV18] Philippe Laborie, Jérôme Rogerie, Paul Shaw, and Petr Vilím. Ibm ilog cp optimizer for scheduling. *Constraints*, 23(2) :210–250, 2018.

[MBR97] Aristide Mingozzi, Lucio Bianco, and Salvatore Ricciardelli. Dynamic programming strategies for the traveling salesman problem with time window and precedence constraints. *Operations research*, 45(3) :365–377, 1997.

[PD07] Knot Pipatsrisawat and Adnan Darwiche. A lightweight component caching scheme for satisfiability solvers. In *International conference on theory and applications of satisfiability testing*, pages 294–299. Springer, 2007.

[PW06] Yves Pochet and Laurence A. Wolsey. *Production Planning by Mixed Integer Programming*. Springer, 2006.

[Sav85] Martin WP Savelsbergh. Local search in routing problems with time windows. *Annals of Operations research*, 4(1) :285–305, 1985.

[Sha98] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *International conference on principles and practice of constraint programming*, pages 417–431. Springer, 1998.

Identifier des *Soft Cores* dans des Formules Propositionnelles

G. Audemard[1], J-M. Lagniez[1], Marie Miceli[1] *, Olivier Roussel[1]

[1] CRIL, Université d'Artois & CNRS, Lens, France

{audemard,lagniez,miceli,roussel}@cril.fr

Résumé

Cet article résume le travail accepté à la conférence ICAART de 2022 [1]. Bien que le problème SAT soit fondamental, il reste difficile pour un utilisateur de comprendre les résultats retournés par un solveur SAT. Nous présentons une nouvelle notion, les soft cores, dont le but est d'expliquer à des utilisateurs pourquoi une formule CNF n'admet pas les solutions souhaitées. Nous montrons également comment utiliser le problème Max#SAT pour extraire un soft core, et proposons un nouveau solveur pour résoudre Max#SAT de manière exacte.

Mots-clés

SAT, #SAT, Max#SAT, Explication

1 Motivation

Ces dernières années, beaucoup de nouvelles notions ont pour but d'expliquer aux utilisateurs les décisions et résultats des systèmes. La modélisation d'un problème en une formule CNF, en plus de potentiellement complexifier la résolution du problème, est difficilement interprétable pour un utilisateur. Il est donc laborieux pour celui-ci de déterminer si la modélisation du problème est correcte ou de comprendre pourquoi certaines (ou toutes) solutions sont impossibles. Lorsque la formule est insatisfiable, une méthode connue pour localiser les raisons de l'incohérence est d'extraire un MUS (pour *Minimally Unsatisfiable Subformula*), un sous-ensemble lui-même insatisfiable mais dont tous les sous-ensembles propres sont satisfiables. Néanmoins, une formule peut admettre un nombre exponentiel de MUS par rapport à sa taille et il est parfois nécessaire de tous les extraire. Même dans le cas où un seul MUS est suffisant, sa taille peut atteindre celle de la formule et il ne peut donc pas être manipulé par un utilisateur. La recherche du plus petit MUS (*Smallest* MUS) ne garantit pas une taille adaptée, en plus d'être un problème difficile car dans $\text{FP}^{\Sigma_2^{\text{PP}}}$ (contre FP^{NP} pour l'extraction d'un MUS). Dans le cas où la formule est satisfiable, un solveur renvoie un modèle, mais ne donne pas plus d'informations sur les autres modèles possibles. Un autre type d'explication est le MES (pour *Minimal Equivalent Subformula*) ou un SMES (*Smallest* MES) : une formule minimale (pour l'inclusion) équivalente à la formule initiale. Cependant, comme pour les MUS (ou SMUS), un MES peut être trop grand. Nous proposons alors une nouvelle notion, les *soft cores*, qui peut s'appliquer aux formules qu'elles soient satisfiables ou non, et qui correspondent à une partie très contrainte de la formule, tout en étant d'assez petite taille pour être compréhensible par un utilisateur.

2 Définition et Complexité

Étant donné une formule Ψ, nous notons $\| \Psi \|$ son nombre de modèles. Un *soft core* est un sous-ensemble d'une formule propositionnelle, petit en taille et en nombre de modèles admis sur l'alphabet de la formule. Clairement, extraire un *soft core* est un problème d'optimisation bicritères, dont les fonctions objectifs à minimiser portent sur le nombre d'éléments de l'ensemble et sur le nombre de modèles. Afin de palier au manque de structure des formules CNF, nous pouvons regrouper les clauses liées sémantiquement (par exemple, celles issues d'une même contrainte de plus haut niveau). Nous obtenons ainsi une formule GCNF (pour *Group* CNF) $\Phi = D \cup \mathcal{G}$, avec $\mathcal{G} = \{G_1, \ldots, G_m\}$ et D permettant de rassembler toutes les contraintes d'intégrité. Nous pouvons alors définir formellement les *soft cores* :

Définition 1 (*soft core*)**.** *Soit une formule GCNF $\Phi = D \cup \mathcal{G}$, $\mathcal{G}'$ est un soft core de Φ si et seulement si $\mathcal{G}' \subseteq \mathcal{G}$ et $\forall \mathcal{G}'' \subseteq \mathcal{G}$, avec $\mathcal{G}' \neq \mathcal{G}''$ et $|\mathcal{G}''| \leq |\mathcal{G}'|$, $\| D \cup \mathcal{G}'' \| \geq \| D \cup \mathcal{G}' \|$.*

Nous pouvons noter que les formules traitées ne sont pas tautologiques. Un *soft core* peut être vu comme une généralisation des notions de SMUS et SMES : s'il est suffisamment grand, son nombre de modèles est égal à celui de la formule (0 pour une formule insatisfiable). Le *soft core* correspond donc à un SMUS ou à un SMES suivant la satisfiabilité de la formule. Dans l'article, nous nous sommes focalisés sur le problème restreint des k-*soft cores*, où la taille, k, est donnée par l'utilisateur.

Définition 2 (k-*soft core*)**.** *Soit une formule GCNF $\Phi = D \cup \mathcal{G}$, $\mathcal{G}'$ est un k-soft core de Φ si et seulement si $\mathcal{G}' \subseteq \mathcal{G}$ avec $|\mathcal{G}'| = k$, et $\forall \mathcal{G}'' \subseteq \mathcal{G}$, avec $\mathcal{G}' \neq \mathcal{G}''$ et $|\mathcal{G}''| = k$, $\| D \cup \mathcal{G}'' \| \geq \| D \cup \mathcal{G}' \|$.*

Nous avons prouvé dans l'article que le problème de décision lié au k-*soft core* est NP^{PP}-difficile en proposant une réduction au problème NP^{PP}-complet E-MajSAT. Nous écrivons $\Psi(X, Y)$ pour spécifier une formule dont l'ensemble

*Papier doctorant

des variables est décomposé en deux ensembles disjoints X et Y. $\| \Psi(X,Y) \|_Z$ dénote le nombre de modèles admis par Ψ sur un ensemble de variables Z.

Définition 3 (E-MajSAT)**.** *Soit une formule CNF $\Psi(X,Y)$. Le problème E-Maj*SAT *consiste à déterminer s'il existe une interprétation ω de l'ensemble X telle que $\| \Psi|_\omega \|_{Var(\Psi)} > \frac{1}{2} \times 2^{|Y|}$, c'est-à-dire que la majorité des extensions sur Y doit satisfaire Ψ.*

Dans la section suivante, nous décrivons comment encoder le problème d'extraction d'un *k-soft core* en une instance Max#SAT, un problème d'optimisation défini comme une extension du problème E-MajSAT.

3 Extraction d'un k-*Soft Core*

Pour une formule CNF quantifiée existentiellement, $\exists Y.\Psi(X,Y)$, avec Y les variables à oublier, calculer le nombre de modèles projetés revient à calculer le nombre d'interprétations sur X qui ont au moins une interprétation sur Y qui satisfait Ψ. Cette notion est nécessaire à la compréhension du problème Max#SAT :

Définition 4 (Max#SAT)**.** *Soit une formule CNF quantifiée $\exists Z.\Psi(X,Y,Z)$. Le problème Max#SAT est de déterminer une interprétation ω sur l'ensemble X qui maximise $\| \Psi|_\omega \|_Y$, soit le nombre d'interprétations sur Y qui ont au moins une extension sur Z satisfaisant Ψ.*

Notre objectif est d'utiliser la proximité de notre problème avec le problème Max#SAT afin de pouvoir extraire un *k-soft core* en résolvant une instance Max#SAT.

3.1 Traduction vers Max#SAT

Nous voulons transformer linéairement une formule GCNF Φ en une formule CNF $\exists Z.\Psi(X,Y,Z)$, de manière à sélectionner, grâce à une interprétation des variables de X, un ensemble de k groupes minimisant le nombre de modèles admis sur les variables du problème Y.
Au départ, nous avons $Var(\Phi) = Y$, X et Z étant vides. Pour extraire des groupes grâce à X, nous associons à chaque clause d'un même groupe G_i une nouvelle variable x_i dite sélecteur, dont l'affectation à vrai (resp. à faux) active (resp. désactive) le groupe. Ces sélecteurs, ayant le même comportement que les identificateurs de groupes, permettent également de considérer une CNF $\Psi(X,Y)$ plutôt qu'une GCNF, avec X l'ensemble des sélecteurs et Y les variables de Φ.
Afin de sélectionner un *k-soft core* tout en maximisant le nombre de modèles, nous utilisons le fait que minimiser le nombre de modèles d'une formule propositionnelle revient à maximiser son nombre de contre-modèles. Nous appliquons alors deux transformations linéaires : la première afin d'effectuer la négation de Ψ, qui résulte en une DNF, et la seconde pour transformer la formule obtenue en une nouvelle CNF. Pour les besoins du problème, il faut que cette dernière transformation préserve le nombre de modèles entre les deux formules après oubli des variables de Z. Pour cela, nous pouvons appliquer la transformation de Tseytin qui garantit cette propriété (les variables d'encodage peuvent être alors placées soit dans Y soit dans Z) ou la transformation de Plaisted&Greenbaum, qui introduit moins de clauses (mais toutes les variables doivent être dans Z, pour ne pas introduire de nouveaux modèles).
Pour finir, afin qu'uniquement k groupes de clauses soient extraits, une contrainte de cardinalité $\sum_{i=1}^{m} x_i = k$ sur les sélecteurs est ajoutée à la formule. Toutes les variables additionnelles de cardinalité sont mises dans Z afin de garder le bon nombre de modèles.
Toute interprétation falsifiant la contrainte de cardinalité entraîne un nombre de modèles égal à zéro ne peut donc pas être une solution du problème Max#SAT, le but étant de maximiser ce nombre. Ainsi, un solveur Max#SAT retourne l'interprétation des X qui maximise le nombre de modèles sur Y après oubli de Z, toutes les clauses dont le sélecteur est activé constituant un *k-soft core*.

3.2 Résolution exacte de Max#SAT

Nous avons également proposé un solveur pouvant résoudre de manière exacte le problème Max#SAT, basé sur le compteur de modèles `d4`. Il prend en entrée une formule CNF Ψ ainsi qu'une tri-partition de ses variables (X,Y,Z), et retourne l'interprétation des X qui maximise le nombre de modèles projetés sur Z. Nous avons tiré avantage de `d4`, en reprenant les fonctionnalités de *caching* et de décomposition dynamique, et utilisé une recherche arborescente descendante similaire. Cependant, la recherche est séparée en deux parties en fonction du type de la variable sélectionnée. La partie haute correspond à l'affectation ω des variables de X. La partie basse équivaut à un comptage de modèles classique, où est calculé, pour l'affectation complète ω décidée durant la partie haute, le nombre de modèles projetés sur Z de $\Psi|_\omega$.
Nous avons comparé notre solveur exact à la méthode approchée `maxcount`, et nous avons pu constater que notre approche est assez compétitive sur le problème d'extraction de *k-soft core*, bien que la contrainte de cardinalité nous ait empêchés de considérer des instances de grande taille.

4 Perspectives

Plusieurs perspectives sont envisageables. Pour l'instant, notre approche reste dédiée au problème Max#SAT. Nous pouvons la spécialiser à la recherche des *k-soft cores* en minimisant les modèles au lieu de maximiser les contre-modèles, ce qui nous éviterait de devoir transformer la formule. Nous pouvons aussi considérer la contrainte de cardinalité directement dans le solveur et non plus globalement (via un encodage CNF). De plus, nous pouvons approcher le problème en appliquant par exemple la recherche locale, de façon à traiter des instances plus grandes.

Références

[1] Gilles Audemard, Jean-Marie Lagniez, Marie Miceli, and Olivier Roussel. Identifying soft core in propositional formulae. In *Proceedings of ICAART'22*, 2022.

Session 7 : Modélisation

Optimiser l'agencement d'une fabrique grâce aux diagrammes de décision

Vianney Coppé*, Xavier Gillard, Pierre Schaus

UCLouvain

{vianney.coppe, xavier.gillard, pierre.schaus}@uclouvain.be

Résumé

Cet article présente un modèle exact pour résoudre un problème d'agencement de départements dans une fabrique avec des contraintes de position, d'ordre et d'adjacence. Ce problème, communément appelé le Constrained Single-Row Facility Layout Problem, a été précédemment résolu avec la programmation linéaire mixte en nombres entiers. L'approche proposée est basée sur les diagrammes de décision. Les expériences montrent qu'elle est plus rapide et permet de résoudre des instances de plus grande taille.

Mots-clés

Optimisation discrète, Diagrammes de décision, Constrained Single-Row Facility Layout Problem

Abstract

This paper presents an exact model for the Constrained Single-Row Facility Layout Problem, an extension of the Single-Row Facility Layout Problem with positioning, ordering and adjacency constraints. A mixed-integer programming model was already proposed for this problem. The proposed approach is based on decision diagrams. It is shown to be faster and allows to solve larger instances.

Keywords

Discrete Optimization, Decision Diagrams, Constrained Single-Row Facility Layout Problem

1 Introduction

Le *Constrained Single-Row Facility Layout Problem* (cSRFLP), est un problème d'agencement de départements au sein d'une fabrique organisée sur une seule rangée, avec des contraintes de position, d'ordre et d'adjacence. Ce problème a été introduit récemment par [9] en ajoutant ces contraintes au *Single-Row Facility Layout Problem* (SRFLP). La seule approche exacte présentée pour résoudre le cSRFLP est le modèle de programmation linéaire mixte en nombres entiers (MIP) de [10]. Dans cet article, le modèle de programmation dynamique (DP) pour le SRFLP [12] est étendu pour incorporer les contraintes du cSRFLP. Il est ensuite expliqué comment ce modèle peut être intégré dans un algorithme de séparation et évaluation, basé sur les diagrammes de décision (DD) [4]. Après quoi, une comparaison entre l'approche existante et celle introduite est faite, sur base d'expériences utilisant des instances de référence pour le SRFLP, auxquelles des contraintes sont ajoutées. Le modèle présenté est nettement plus rapide et permet de résoudre des instances de plus grande taille. L'article est conclu par un résumé des contributions ainsi que des pistes d'amélioration.

*L'auteur principal est doctorant.

2 Définition du problème

Soit un ensemble de départements $N = \{1, \dots, n\}$ dans une fabrique, dont chaque département $i \in N$ possède une longueur l_i. À chaque paire de départements $i, j \in N$ est associée une quantité de trafic c_{ij} les reliant. L'objectif du cSRFLP est de placer les départements côte à côte sur une seule rangée de manière à minimiser la distance totale parcourue dans la fabrique. L'agencement des départements est soumis à trois types de contraintes définies ci-après. Les contraintes de *position* assignent à un département une position fixe dans l'arrangement. Nous utiliserons deux fonctions pour représenter ces contraintes : $position : N \rightarrow N \cup \{0\}$ et $department : N \rightarrow N \cup \{0\}$. La première donne la position à laquelle chaque département est fixé, ou 0 si la position du département n'est pas imposée. La deuxième est la relation inverse, reliant les positions aux départements. Les contraintes d'*ordre* imposent une précédence entre deux départements dans l'arrangement. La fonction $predecessors : N \rightarrow 2^N$ lie chaque département à l'ensemble de départements qui doivent être placés à gauche de celui-ci. Enfin, les contraintes d'*adjacence* sont similaires aux contraintes d'ordre, à la différence que les départements concernés doivent cette fois-ci être adjacents. La fonction $previous : N \rightarrow N \cup \{0\}$ envoie donc chaque département vers son voisin gauche direct, s'il existe, et 0 sinon. Soit la bijection $\pi : N \rightarrow N$ reliant les départements à leur position dans l'arrangement, la fonction objectif du problème est définie comme suit :

$$cSRFLP(\pi) = \sum_{k=1}^{n} l_k \sum_{\substack{i=1 \\ \pi(i)<\pi(k)}}^{n} \sum_{\substack{j=1 \\ \pi(k)<\pi(j)}}^{n} c_{ij} + K \quad (1)$$

$$K = \sum_{i=1}^{n} \sum_{\substack{j=1 \\ i<j}}^{n} c_{ij} \frac{l_i + l_j}{2} \quad (2)$$

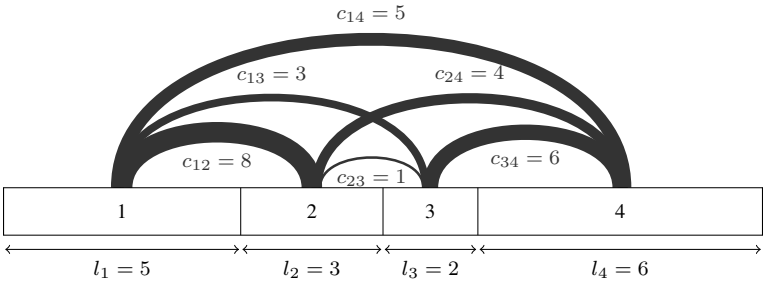


FIGURE 1 – Une instance du cSRFLP dont les départements sont arrangés de manière optimale.

à minimiser en respectant les contraintes suivantes :

$$\pi(i) = j \quad \forall i, j \in N, position(i) = j \neq 0 \quad (3)$$
$$\pi(i) > \pi(j) \quad \forall i \in N, \forall j \in predecessors(i) \quad (4)$$
$$\pi(i) = \pi(j) + 1 \quad \forall i, j \in N, previous(i) = j \neq 0. \quad (5)$$

et où K est une constante englobant les contributions des demi-longueurs de département à la fonction objectif.

3 Modèle DP

Avant de présenter comment intégrer les contraintes du cSRFLP au modèle DP pour le SRFLP [12], nous rappelons d'abord celui-ci. L'idée est de placer les départements séquentiellement de gauche à droite et de maintenir comme état l'ensemble des départements qui restent à placer. De cette manière, il est possible d'ajouter la contribution individuelle d'un département à la fonction objectif lorsqu'il est placé. Formellement, le modèle DP est composé des *variables de décision* $x_j \in D_j$ avec $j \in \{0, \ldots, n-1\}$. La variable x_j correspond au département placé à la position $j+1$. Toutes les variables ont le même domaine $D_j = N$ puisque tous les départements sont candidats à être placés à chaque position. L'ensemble des états S contient tous les sous-ensembles de N, dont l'état *racine* $\hat{r} = N$ et *terminal* $\hat{t} = \emptyset$. On définit également un état *non-admissible* $\hat{0}$. L'ensemble S est partitionné en $n+1$ ensembles $S_0, \ldots, S_n$, où l'ensemble S_j contient tous les états avec j variables assignées. Les *fonctions de transition* $t_j : S_j \times D_j \to S_{j+1}$ pour chaque $j = 0, \ldots, n-1$ spécifient l'état résultant d'une décision supplémentaire sur un état :

$$t_j\left(s^j, x_j\right) = \begin{cases} s^j \setminus \{x_j\}, & \text{si } x_j \in s^j \\ \hat{0}, & \text{sinon.} \end{cases} \quad (6)$$

Celles-ci vont de paire avec les *fonctions de coût de transition* $h_j : S_j \times D_j \to \mathbb{R}$ pour $j = 0, \ldots, n-1$ qui associent une valeur à chaque transition :

$$h_j\left(s^j, x_j\right) = \begin{cases} l_{x_j} \sum_{i \in \overline{s}^j} \sum_{k \in s^j \setminus \{x_j\}} c_{ik}, & \text{si } x_j \in s^j \\ 0, & \text{sinon.} \end{cases} \quad (7)$$

Cette formule découle de l'équation (1) puisque $\overline{s}^j$ – le complément de s^j – et $s^j \setminus \{x_j\}$ contiennent respectivement les départements placés à gauche et à droite du département considéré. Le dernier élément est la valeur donnée à la racine : $v_r = K$, voir équation (2).

La solution de ce modèle DP peut ensuite être trouvée en résolvant la formule de récurrence suivante :

$$\begin{aligned} &\min \hat{f}(x) = v_r + \sum_{j=0}^{n-1} h_j\left(s^j, x_j\right) \\ &\text{sujet à } s^{j+1} = t_j\left(s^j, x_j\right), \forall x_j \in D_j, j = 0, \ldots, n-1 \\ &\qquad s^j \in S_j, j = 0, \ldots, n. \end{aligned} \quad (8)$$

Afin d'intégrer au modèle les contraintes mentionnées dans la section 2, nous introduisons les prédicats $valid_j : S_j \times D_j \to \{true, false\}$ pour $j = 0, \ldots, n-1$. Ils servent à filtrer les solutions non-admissibles dans les fonctions de transition :

$$t_j\left(s^j, x_j\right) = \begin{cases} s^j \setminus \{x_j\}, & \text{si } x_j \in s^j \wedge valid_j(s^j, x_j) \\ \hat{0}, & \text{sinon.} \end{cases} \quad (9)$$

Pour être satisfaits, ces prédicats nécessitent que toutes les contraintes soient respectées :

$$valid_j(s^j, x_j) = p_j(s^j, x_j) \wedge o_j(s^j, x_j) \wedge r_j(s^j, x_j) \quad (10)$$

où p_j, o_j et r_j concernent respectivement les contraintes de position, ordre et adjacence :

$$\begin{aligned} &p_j(s^j, x_j) = (position(x_j) = j+1) \\ &\vee (position(x_j) = 0 \wedge department(j+1) = 0) \end{aligned} \quad (11)$$
$$o_j(s^j, x_j) = predecessors(x_j) \subseteq \overline{s}^j \quad (12)$$
$$\begin{aligned} &r_j(s^j, x_j) = (previous(x_j) \in \overline{s}^j) \vee \\ &(previous(x_j) = 0 \wedge \nexists k \in s^j : previous(k) \in \overline{s}^j). \end{aligned} \quad (13)$$

L'équation (11) vérifie que le département x_j doit être placé à la position $j+1$, ou bien que le département et la position en question ne soient pas impliqués dans une contrainte de position. Pour les contraintes d'ordre, l'équation (12) s'assure que tous les prédécesseurs du département x_j sont déjà placés. Dans l'équation (13), on s'assure que le voisin direct du département x_j soit déjà placé, ou qu'aucun des départements qu'il reste à placer n'a de contrainte d'adjacence.

Calcul des coûts de transition Pour calculer les coûts de transition plus rapidement, nous maintenons également dans chaque état les *valeurs de coupe* de chaque département libre, c'est-à-dire la somme des quantités de trafic

reliant les départements placés et le département libre en question. Pour l'état s^j et chaque département $i \in N$, on définit :

$$s^j_{cut}[i] = \begin{cases} \sum_{j \in \overline{s}^j} c_{ij}, & \text{si } i \in s^j \\ 0, & \text{sinon} \end{cases} \quad (14)$$

qui est mis à jour en $\mathcal{O}(n)$ lors des transitions $t_j\left(s^j, x_j\right)$:

$$s^{j+1}_{cut}[i] = \begin{cases} s^j_{cut}[i] + c_{ix_j}, & \text{si } i \in s^j \setminus \{x_j\} \\ 0, & \text{sinon.} \end{cases} \quad (15)$$

La formule du coût de transition devient donc :

$$h_j\left(s^j, x_j\right) = \begin{cases} l_{x_j} \sum_{i \in s^j \setminus \{x_j\}} s^j_{cut}[i], & \text{si } x_j \in s^j \\ 0, & \text{sinon} \end{cases} \quad (16)$$

qui peut être calculée en $\mathcal{O}(n)$ au lieu de $\mathcal{O}\left(n^2\right)$. Les valeurs de coupe seront également utilisées dans la section 5.2 pour obtenir une borne inférieure.

4 Représentation DD

En partant d'un modèle DP, il est aisé de construire le DD exact correspondant. En effet, un DD est un *graphe orienté acyclique* $B = (U, A, d, v, \sigma)$ dont l'ensemble de *nœuds* U est partitionné en plusieurs *couches* $L_0, \ldots, L_n$. Les *arcs* $a \in A$ reliant les nœuds de couches consécutives L_j et L_{j+1} encodent l'assignation d'une valeur à la variable de décision x_j. De plus, la fonction σ fait correspondre chaque nœud à un état du modèle DP. Ainsi, la première couche contient un seul nœud, le nœud racine, logiquement associé à l'état racine du modèle DP. Ensuite, pour chaque nœud d'une couche donnée L_j, on génère la couche L_{j+1} en ajoutant un nœud pour chaque état *distinct* obtenu grâce à la fonction de transition. Lors d'une transition, la valeur assignée à la variable de décision et le coût correspondant sont donnés respectivement par le *label* $d(a)$ et la *longueur* $v(a)$ de l'arc associé. Dès lors, un chemin $p = \left(a^{(0)}, \ldots, a^{(n-1)}\right)$ reliant le nœud racine r au nœud terminal t représente une solution au problème, avec $x_j = d\left(a^{(j)}\right)$, et dont la longueur totale $v(p) = v_r + \sum_{j=0}^{n-1} v\left(a^{(j)}\right)$ équivaut au coût. Dans un DD *exact*, qui contient toutes les solutions admissibles du problème associé, trouver la solution optimale revient donc à résoudre un problème de plus court chemin.

5 Séparation et évaluation

L'attrait des DDs pour l'optimisation discrète vient surtout de l'algorithme de séparation et évaluation basé sur ceux-ci, présenté dans [4]. Au lieu de générer un DD exact pour l'entièreté du problème considéré, l'idée est d'explorer celui-ci intelligemment en générant des DDs approximés, permettant à la fois de trouver des solutions admissibles de bonne qualité et de couper certaines branches.

5.1 Borne supérieure

Pour calculer une borne supérieure, les DDs *restreints* sont utilisés. Leur fonctionnement est simple : on impose au DD développé à partir du nœud donné une *largeur* maximale, qui correspond au nombre maximal de nœuds contenus dans une de ses couches. Pendant la création du DD en question, le surplus de nœuds générés dans chaque couche est alors supprimé. De cette manière, le DD résultant contient un sous-ensemble des solutions admissibles du problème, et fournit donc une borne supérieure. La qualité de cette borne est directement impactée par le choix des nœuds retirés de chaque couche, en pratique on retire donc ceux qui semblent les moins prometteurs.

5.2 Borne inférieure

À la façon des DDs restreints, les DDs *relaxés* permettent de calculer des bornes inférieures. Une largeur maximale est également imposée, mais cette fois-ci le surplus de nœuds est fusionné en un unique nœud. Celui-ci est réintroduit dans le DD, de manière à n'exclure aucune solution admissible. Ce faisant, il est probable que des solutions non-admissibles soient introduites dans le DD. L'opérateur de fusion mentionné doit être spécifié pour chaque problème étudié. Pour le cSRFLP, nous en avons essayé plusieurs mais avons obtenu les meilleurs résultats avec une borne inférieure heuristique.
Étant donné un nœud u et la valeur $v^*(u)$ du plus court chemin reliant le nœud racine à celui-ci, le théorème 1 montre que le coût restant à ajouter à $v^*(u)$ pour obtenir le coût complet de l'arrangement $\pi^*|_u$ peut être divisé en deux termes, que nous allons ensuite approximer séparément.

Théorème 1 *Soit un nœud u et son état $\sigma(u) = s$, soit $\pi^*|_u$ l'arrangement de coût minimal qui traverse u. Par souci de concision, on pose $\pi = \pi^*|_u$. On a alors l'égalité :*

$$SRFLP(\pi) - v^*(u) = \underbrace{\sum_{k \in s} l_k \sum_{\substack{i \in s \\ \pi(i) < \pi(k)}} \sum_{\substack{j \in s \\ \pi(k) < \pi(j)}} c_{ij}}_{\text{coût d'arrangement des départements libres}} + \underbrace{\sum_{k \in s} l_k \sum_{\substack{j \in s \\ \pi(k) < \pi(j)}} s_{cut}[j]}_{\text{coût lié aux départements fixés}}. \quad (17)$$

Coût d'arrangement des départements libres En observant le premier terme de l'équation (17), on remarque qu'il vaut exactement le coût de l'arrangement considérant uniquement les départements libres. Celui-ci peut être sous-évalué comme spécifié par l'algorithme 1. On commence par trier les quantités de trafic par ordre décroissant. De même, on trie les longueurs des départements par ordre croissant. On remarque ensuite qu'en plaçant n départements côte à côte, $n - k$ paires de départements seront séparés par $k - 1$ départements (voir figure 1). Une borne inférieure est obtenue en supposant que les plus importantes intensités de trafic reliant des départements libres sont multipliées par 0, n'ayant aucun département les séparant. Les $n - k$ plus grandes intensités de trafic suivantes sont multipliées par la longueur des $k - 1$ départements libres les

Algorithme 1 Calcul de $LB_{edge}(u)$.

Require: $edge = sorted_{\geq}(\{\langle c : c_{ij}, dep1 : i, dep2 : j\rangle \mid 1 \leq i < j \leq n\})$
Require: $length = sorted_{\leq}(\{\langle l : l_i, dep : i\rangle \mid 1 \leq i \leq n\})$
$s \leftarrow \sigma(u), lb \leftarrow 0, cumul\_l \leftarrow 0, i \leftarrow 1, j \leftarrow 1$
for $k \leftarrow 1$ **to** $|s| - 1$ **do**
  for $l \leftarrow 1$ **to** k **do**
    while $edge[i].dep1 \notin s \vee edge[i].dep2 \notin s$ **do**
      $i \leftarrow i + 1$
    $lb \leftarrow lb + cumul\_l \cdot edge[i].c$
    $i \leftarrow i + 1$
  while $length[j].dep \notin s$ **do**
    $j \leftarrow j + 1$
  $cumul\_l \leftarrow cumul\_l + length[j].l$
  $j \leftarrow j + 1$
return lb

Algorithme 2 Calcul de $LB_{cut}(u)$.

$s \leftarrow \sigma(u), lb \leftarrow 0, cumul\_l \leftarrow 0, order \leftarrow [\,]$
for all $i \in s$ **do**
  $order.append(\langle ratio : s_{cut}[i]/l_i, dep : i\rangle)$
$sort_{\geq}(order)$
for $k \leftarrow 1$ **to** $|s|$ **do**
  $lb \leftarrow lb + cumul\_l \cdot s_{cut}[order[k].dep]$
  $cumul\_l \leftarrow cumul\_l + l_{order[k].dep}$
return lb

moins longs. Cette borne inférieure peut être vue comme une généralisation de la *Edges method* [11] pour le *Minimum Linear Arrangement Problem*, un cas particulier du SRFLP.

Coût lié aux départements fixés Le second terme de l'équation (17) tient compte des valeurs de coupe, dont la valeur sera encore multipliée par la distance séparant chaque département du dernier département placé. Une borne inférieure est donnée par la *borne de première génération* introduite dans [13]. Comme formalisé par l'algorithme 2, elle consiste à trier les départements libres par ordre décroissant de ratio entre valeur de coupe et longueur. L'arrangement optimal des départements libres par rapport au coût lié aux valeurs de coupe est donné par le résultat du tri, sa valeur est donc une borne inférieure sur le deuxième terme de l'équation (17).
Finalement, la borne inférieure d'un nœud u est donnée par $LB(u) = v^*(u) + LB_{edge}(u) + LB_{cut}(u)$. Celle-ci peut être utilisée pour filtrer des nœuds à la fois dans l'algorithme de séparation et évaluation, et dans la génération des DDs [6].

6 Résultats expérimentaux

L'approche présentée a été implémentée en C++, en basant largement le code sur la librairie DDO [5] pour le langage Rust. Nous la comparons avec le modèle MIP introduit par Liu et al. [10] implémenté avec Gurobi [7]. Nos expériences utilisent des instances classiques du SRFLP [1, 2, 3, 8, 13] avec jusque 25 départements, auxquelles nous ajoutons des ensembles valides de contraintes, à savoir :

— des ensembles de contrainte avec $2, 4, 6, 8$ et 10 contraintes de position, d'ordre ou d'adjacence.
— des ensembles de contrainte avec $0, 2, 4, 6, 8$ et 10 contraintes de position, d'ordre et d'adjacence.

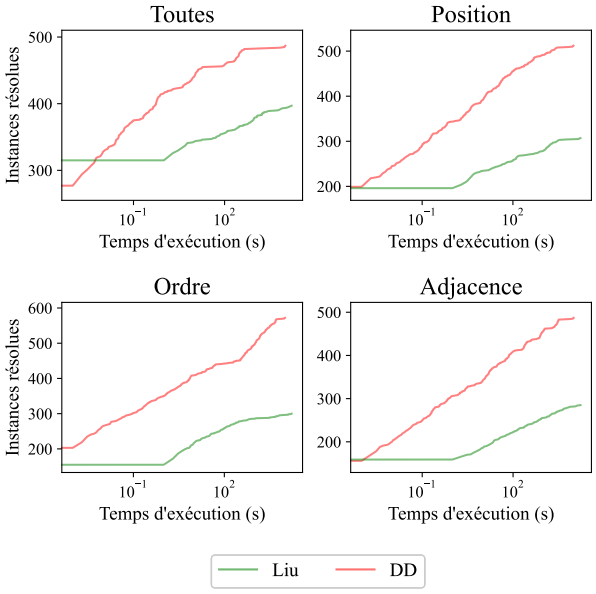


FIGURE 2 – Nombre d'instances résolues par chaque algorithme en fonction du type de contraintes imposé.

Pour chacune de ces configurations, cinq ensembles de contraintes ont été générés de façon aléatoire. Les deux algorithmes ont ensuite été confrontés à chaque combinaison valide d'instance et ensemble de contraintes, avec un temps maximal fixé à cinq heures. La figure 2 montre le résultat de ces expériences : quel que soit le type de contraintes imposé, l'approche DD est capable de résoudre toutes les instances tandis que le modèle MIP est incapable d'en résoudre presque la moitié. De plus, cette supériorité est aussi observée en termes de vitesse de résolution pour les instances résolues par les deux algorithmes.

Détails d'implémentation Pour l'approche DD, nous utilisons des DDs de largeur maximale égale à 3. Nous avons en effet remarqué que de très bonnes solutions étaient trouvées assez tôt dans la recherche malgré cette faible largeur, et avec les adaptations décrites précédemment, la largeur n'a pas d'influence sur la qualité des bornes inférieures. La sélection des nœuds à supprimer pour créer les DDs restreints se fait sur base de la borne inférieure. Enfin, nous avons adapté l'algorithme de séparation et évaluation pour parcourir les nœuds en largeur. Cela permet d'éviter d'explorer plusieurs nœuds correspondant au même état.

7 Conclusion

Dans cet article, le modèle DP du SRFLP a été étendu pour couvrir les contraintes introduites par le cSRFLP. Une approche DD basée sur ce modèle a ensuite été décrite, y compris une formule de borne inférieure. Les expériences effectuées sur un large panel d'instances ont permis de montrer la rapidité de l'approche présentée. Celle-ci pourrait certainement être améliorée dans le futur, notamment en tenant compte des contraintes dans les calculs de borne inférieure.

Références

[1] André R. S. Amaral. On the exact solution of a facility layout problem. *European Journal of Operational Research*, 173(2) :508–518, 2006.

[2] André R. S. Amaral. An exact approach to the one-dimensional facility layout problem. *Operations Research*, 56(4) :1026–1033, 2008.

[3] Miguel F. Anjos and Anthony Vannelli. Computing globally optimal solutions for single-row layout problems using semidefinite programming and cutting planes. *INFORMS Journal on Computing*, 20(4) :611–617, 2008.

[4] David Bergman, Andre A. Cire, Willem-Jan van Hoeve, and John N. Hooker. Discrete optimization with decision diagrams. *INFORMS Journal on Computing*, 28(1) :47–66, 2016.

[5] Xavier Gillard, Pierre Schaus, and Vianney Coppé. Ddo, a generic and efficient framework for mdd-based optimization. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20)*, pages 5243–5245, 2020.

[6] Xavier Gillard, Pierre Schaus, Vianney Coppé, and André A. Cire. Improving the filtering of branch-and-bound mdd solver. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 2021.

[7] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2022.

[8] Sunderesh S. Heragu and Andrew Kusiak. Efficient models for the facility layout problem. *European Journal of Operational Research*, 53(1) :1–13, 1991.

[9] Zahnupriya Kalita and Dilip Datta. A constrained single-row facility layout problem. *The international journal of advanced manufacturing technology*, 98(5) :2173–2184, 2018.

[10] Silu Liu, Zeqiang Zhang, Chao Guan, Lixia Zhu, Min Zhang, and Peng Guo. An improved fireworks algorithm for the constrained single-row facility layout problem. *International Journal of Production Research*, 59(8) :2309–2327, 2021.

[11] Jordi Petit. Experiments on the minimum linear arrangement problem. *Journal of Experimental Algorithmics*, 8, 2003.

[12] Jean-Claude Picard and Maurice Queyranne. On the one-dimensional space allocation problem. *Operations Research*, 29(2) :371–391, 1981.

[13] Donald M. Simmons. One-dimensional space allocation : an ordering algorithm. *Operations Research*, 17(5) :812–826, 1969.

Des encodages PB pour la résolution de problèmes CSP

Thibault Falque[1,2], Romain Wallon[2]

[1] Exakis Nelite
[2] CRIL, Univ Artois & CNRS

{falque,wallon}@cril.univ-artois.fr

Résumé

Une approche possible pour résoudre un problème CSP est d'encoder ce problème sous la forme d'une formule CNF, et ensuite utiliser un solveur SAT pour la résoudre. Le principal avantage de cette technique est qu'elle permet de bénéficier de l'efficacité pratique des solveurs SAT modernes, fondés sur l'architecture CDCL. Cependant, le pouvoir d'inférence de ces solveurs est assez « faible », car limité par celui du système de preuve par résolution utilisé pendant l'analyse de conflit. Cette observation a conduit au développement de solveurs pseudo-booléen (PB), qui implantent le système de preuve plus puissant des plans-coupes, ainsi que de nombreuses autres techniques héritées des solveurs SAT. De plus, les solveurs PB sont capables de raisonner nativement sur des contraintes PB, c'est-à-dire, des équations ou inéquations linéaires en variables booléennes. Ces contraintes sont plus concises que les clauses, de sorte qu'une seule contrainte PB peut représenter un nombre exponentiel de clauses. Dans cet article, nous tirons parti à la fois de cette concision et du pouvoir d'inférence des solveurs PB pour résoudre des problèmes CSP. Pour ce faire, nous définissons des encodages PB pour différentes contraintes populaires, et confions leur résolution à des solveurs PB pour comparer leurs performances à celles d'autres solveurs CSP existants.

Mots-clés

programmation par contraintes, encodages SAT, solveurs PB

Abstract

One of the possible approaches for solving a CSP is to encode the input problem into a CNF formula, and then use a SAT solver to solve it. The main advantage of this technique is that it allows to benefit from the practical efficiency of modern SAT solvers, based on the CDCL architecture. However, the reasoning power of SAT solvers is somehow "weak", as it is limited by that of the resolution proof system they use internally. This observation led to the development of so called pseudo-Boolean (PB) solvers, that implement the stronger cutting planes proof system, along with many of the solving techniques inherited from SAT solvers. Additionally, PB solvers can natively reason on PB constraints, i.e., linear equalities or inequalities over Boolean variables. These constraints are more succinct than clauses, so that a single PB constraint can represent exponentially many clauses. In this paper, we leverage both this succinctness and the reasoning power of PB solvers to solve CSPs by designing PB encodings for different common constraints, and feeding them into PB solvers to compare their performance with that of existing CP solvers.

Keywords

constraint programming, PB solving, SAT encodings

1 Introduction

Le problème de satisfaction de contraintes (CSP) consiste à déterminer si un ensemble de contraintes est cohérent et, si tel est le cas, d'identifier une solution satisfaisant l'ensemble de ces contraintes. Afin de résoudre ce type de problèmes, plusieurs approches ont été proposées [18]. L'une de ces approches consiste à travailler nativement sur les contraintes du problème, à l'aide de structures de données efficaces permettant de représenter ces contraintes et la structure du problème, à l'image de solveurs tels que `Choco` [23], `Nacre` [11] ou encore `ACE` [1].

Une autre approche possible consiste à tirer parti de l'efficacité pratique des solveurs SAT modernes, fondés sur l'architecture CDCL [20, 21, 8] pour résoudre les problèmes CSP, qui sont eux-même NP-complets. Comme ces solveurs prennent en entrée une formule propositionnelle en forme normale conjonctive, il est nécessaire d'utiliser des *encodages* permettant de représenter les domaines des variables ainsi que les contraintes du problème initial sous forme de clauses. Dans ce cadre, de nombreuses solutions ont été proposées, comme celles décrites dans [26, 24, 10, 6, 3, 25, 1].

Malgré leur efficacité pratique, les solveurs SAT souffrent d'un problème majeur : le système de preuves par résolution, utilisé lors de l'analyse de conflit, présente un pouvoir d'inférence relativement faible, qui empêche de résoudre efficacement des problèmes en apparence simple, tel que celui du *pigeonhole principle* (ou *principe des tiroirs* en français) [13]. Plus généralement, les problèmes nécessitant de « savoir compter » sont souvent difficile à résoudre pour ce type de solveurs. Cette observation a conduit au développement de solveurs dit pseudo-booléens (PB) [7, 16, 9], qui héritent de nombreuses fonctionnalités des solveurs SAT

1. `https://github.com/xcsp3team/ace`

tout en implantant le système de preuve des plans-coupes plus puissant que celui de la résolution [12, 14, 22].
Ces solveurs sont par ailleurs capables de gérer nativement des contraintes PB, c'est-à-dire, des équations ou inéquations linéaires en variables booléennes. Or, parmi les encodages SAT existants, nombreux sont ceux qui utilisent en pratique une représentation intermédiaire des contraintes sous la forme de contraintes PB, avant d'encoder ces même contraintes sous la forme de clauses. Cette étape est nécessaire à l'utilisation d'un solveur SAT, mais requiert l'introduction de variables additionnelles, et l'augmentation du nombre de contraintes à donner au solveur (une unique contrainte PB pouvant représenter un nombre exponentiel de clauses).
Dans cet article, nous présentons donc des encodages tirant parti à la fois de la concision des contraintes PB et du pouvoir d'inférence des solveurs PB. Ces encodages exploitent notamment des approches classiques de représentation des domaines, tels que le *direct-encoding*, le *log-encoding* ou encore l'*order-encoding* pour ensuite encoder sous la forme de contraintes PB différentes contraintes populaires. Pour cela, nous commençons par introduire quelques préliminaires relatifs à la résolution de problèmes CSP et PB, avant de présenter formellement nos encodages et de les évaluer empriquement sur différents jeux d'instances.

2 Préliminaires

2.1 Programmation par contraintes

Un *réseau de contraintes* (ou *CN* pour *Constraint Network*) se compose d'un ensemble fini de variables et d'un ensemble fini de contraintes. Chaque variable X peut prendre sa valeur dans un ensemble fini appelé *domaine* de X, et noté $\texttt{dom}(X)$. Chaque contrainte est défini par une relation sur un ensemble de variables. Une *solution* d'un CN est une affectation de valeurs à toutes les variables telle que toutes les contraintes soient satisfaites. Un CN est *cohérent* s'il admet au moins une solution, et le problème de décision correspondant, appelé *CSP* (pour *Constraint Satisfaction Problem*), consiste à déterminer si un CN est cohérent ou non.

2.2 Solveurs SAT et CSP

Une variable x est dite *booléenne* lorsque $\texttt{dom}(X) = \{0, 1\}$. Nous appelons *littéral* ℓ une variable booléenne x ou sa négation $\bar{x} = 1 - x$. Le littéral ℓ est dit *satisfait* lorsque ℓ est affecté à 1, et *falsifié* dans le cas contraire. Une *clause* est une disjonction de littéraux, qui impose la satisfaction d'au moins l'un de ses littéraux, et un problème est dit en *Forme Normale Conjonctive* (*CNF*, pour *Conjunctive Normal Form*) lorsqu'il est constitué d'une conjonction de clauses. Le *problème de cohérence propositionnel* (*SAT*) consiste à déterminer si une telle conjonction possède une solution.
Le problème SAT étant le problème NP-complet de référence [5], il est possible d'utiliser des solveurs SAT pour résoudre des problèmes CSP, en utilisant différents encodages. En particulier, pour représenter le domaine d'une variable CSP X, nous pouvons dans un premier temps utiliser le *direct-encoding* (voir par exemple [26]). Celui-ci consiste à utiliser une variable booléenne x_v pour chacune des valeurs $v \in \texttt{dom}(X)$. Dans ce cas, la valeur affectée à la variable X peut être obtenue en identifiant (l'unique) variable booléenne x_v satisfaite.
Cette représentation est particulièrement commode dans le cas d'un domaine où les valeurs sont énumérées et ne constituent pas un intervalle de valeurs. Dans ce dernier cas, une autre solution consiste à représenter la variable à l'aide du *log-encoding*, qui utilise la décomposition en base 2 de la variable X à l'aide de variables booléennes b_i représentant les bits de X (voir par exemple [26]). Cette représentation est définie par l'égalité suivante :

$$X = \min(\texttt{dom}(X)) + \sum_{i=0}^{\lceil \log_2(X) \rceil} 2^i b_i$$

Notons ici l'ajout de $\min(\texttt{dom}(X))$, qui vise à garantir que la décomposition en base 2 encode toujours une valeur valant au minimum 0.
Enfin, une dernière représentation possible est celle fondée sur l'*order-encoding* [25], qui utilise une variable booléenne $x_{\geq v}$ pour chacune des valeurs $v \in \texttt{dom}(X) \setminus \{\min(\texttt{dom}(X))\}$, qui est satisfaite si et seulement si $X \geq v$. Dans ce cas, la valeur affectée à X peut être obtenue en identifiant deux variables $x_{\geq v}$ et $x_{\geq v+1}$ telle que la première est satisfaite et la seconde est falsifiée, auquel cas la variable X est affectée à v.

2.3 Contraintes pseudo-booléennes (PB)

Une *contrainte pseudo-booléenne* (*PB*) est une contrainte de la forme $\sum_{i=1}^{n} \alpha_i \ell_i \ \triangle \ \delta$, où n est un entier naturel, les *poids* (ou *coefficients*) α_i et le *degré* δ sont des entiers, les ℓ_i sont des littéraux et $\triangle \in \{<, \leq, =, \geq, >\}$.
Une contrainte PB est dite *normalisée* lorsque les coefficients et le degré de cette contrainte sont strictement positifs, et $\triangle$ est $\geq$. Toute contrainte PB peut être écrite sous la forme d'une conjonction de contraintes PB normalisées, ce qui peut être particulièrement commode dans le cas des encodages que nous présentons plus loin.
Une *contrainte de cardinalité PB* est une contrainte PB normalisée dont tous les coefficients sont égaux à 1, et une *clause* est une contrainte de cardinalité PB de degré 1. Cette définition coïncide avec la définition de clause en tant que disjonction de littéraux, et illustre le fait que les solveurs PB généralisent les solveurs SAT.

3 Encodages purement PB

Dans cette section, nous exploitons le *direct-encoding*, le *log-encoding* et l'*order-encoding* pour représenter le domaine des variables d'un CN, et utilisons cette représentation pour encoder sous forme de contraintes PB plusieurs contraintes CSP couramment utilisées, en considérant tout particulièrement celles reconnues par les *mini-solvers* de la compétition XCSP3, ainsi des contraintes de comptage.

3.1 (Dés)activation de contraintes PB

Pour encoder un problème CSP (qu'il s'agisse du domaine de ses variables ou de ses contraintes) sous la forme de contraintes PB, il est souvent nécessaire de pouvoir activer (ou désactiver) une contrainte. Pour ce faire, il est commode d'utiliser un *sélecteur*, c'est-à-dire, une variable fraîche s dont la satisfaction entraîne celle de la contrainte considérée. Dans le cas des contraintes PB, un tel sélecteur s pourrait avoir la sémantique suivante (où $\Rightarrow$ représente l'implication matérielle) :

$$s \Rightarrow \sum_{i=1}^{n} \alpha_i \ell_i \geq \delta$$

La forme particulière des contraintes PB fournit une manière simple et concise de représenter cette implication sous la forme d'une unique contrainte PB, donnée ci-dessous :

$$\delta \bar{s} + \sum_{i=1}^{n} \alpha_i \ell_i \geq \delta$$

Rappelons que, dans le cas présenté ci-dessus, la satisfaction de la contrainte ne garantit pas la satisfaction de la variable s. Lorsque cette garantie est nécessaire, il faut ajouter l'implication réciproque :

$$s \Leftarrow \sum_{i=1}^{n} \alpha_i \ell_i \geq \delta$$

Cette implication peut elle aussi être représentée sous la forme d'une unique contrainte pseudo-booléenne, à savoir :

$$\left(\sum_{i=1}^{n} \alpha_i - \delta + 1\right) s + \sum_{i=1}^{n} \alpha_i \bar{\ell_i} \geq \sum_{i=1}^{n} \alpha_i - \delta + 1$$

Dans la suite, nous noterons s un sélecteur pour lequel uniquement la première implication est définie, et $\mathcal{S}$ un sélecteur pour lequel les deux implications sont définies (si nécessaire, ces symboles pourront être indicés).
Pour illustrer l'utilisation des sélecteurs, considérons la contrainte de diséquation suivante :

$$\sum_{i=1}^{n} \alpha_i \ell_i \neq \delta$$

Notons que cette contrainte ne peut pas être normalisée directement, comme l'opérateur $\neq$ n'est pas autorisé par les contraintes PB. En revanche, nous pouvons observer que cette contrainte est en fait équivalente à la disjonction :

$$\left(\sum_{i=1}^{n} \alpha_i \ell_i \leq \delta - 1\right) \vee \left(\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta + 1\right)$$

Nous définissons alors deux nouveaux sélecteurs, $s_{\leq}$ et $s_{\geq}$, ayant respectivement les sémantiques suivantes :

$$s_{\leq} \Rightarrow \sum_{i=1}^{n} \alpha_i \ell_i \leq \delta - 1$$

$$s_{\geq} \Rightarrow \sum_{i=1}^{n} \alpha_i \ell_i \geq \delta + 1$$

Il suffit alors d'ajouter la définition de ces deux sélecteurs, ainsi que la disjonction $s_{\leq} \vee s_{\geq}$, pour représenter la diséquation présentée plus haut.

3.2 Les variables et leurs domaines

Afin de nous assurer que les encodages introduit dans la section précédente représente effectivement le domaine des variables du problème original, nous avons besoin d'ajouter un certain nombre de contraintes. Dans le cas du *direct-encoding*, il suffit d'ajouter la contrainte PB suivante, qui garantit que la variable X peut prendre exactement une valeur parmi celles de son domaine :

$$\sum_{v \in \mathtt{dom}(X)} x_v = 1$$

Il est alors possible de représenter la variable X à l'aide de l'égalité suivante :

$$X = \sum_{v \in \mathtt{dom}(X)} v x_v$$

Dans le cas du *log-encoding*, la représentation de la variable X étant donnée par l'égalité suivante :

$$X = \min(\mathtt{dom}(X)) + \sum_{i=0}^{\lceil \log_2(X) \rceil} 2^i b_i$$

la contrainte à ajouter pour s'assurer que le domaine de X soit bien respecté est donnée ci-dessous (rappelons que cet encodage est utilisé ici uniquement pour représenter des domaines correspondant à des intervalles) :

$$\sum_{i=0}^{\lceil \log_2(X) \rceil} 2^i b_i \leq \max(\mathtt{dom}(X)) - \min(\mathtt{dom}(X))$$

Enfin, dans le cas de l'*order-encoding*, les contraintes à ajouter sont exactement les mêmes clauses que dans l'article original introduisant cette approche [25]. Ainsi, pour toute valeur $v \in \mathtt{dom}(X) \setminus \{\min(\mathtt{dom}(X))\}$, l'implication matérielle suivante est ajoutée au solveur :

$$x_{\geq v} \Rightarrow x_{\geq v-1}$$

Lorsque le domaine de X n'est pas un intervalle, il est possible d'interdire une valeur v en ajoutant, en plus, l'implication :

$$x_{\geq v} \Rightarrow x_{\geq v+1}$$

Notons de plus qu'il est possible, en utilisant l'*order-encoding*, de représenter X grâce à l'égalité suivante :

$$X = \min(\mathtt{dom}(X)) + \sum_{v \in \mathtt{dom}(X) \setminus \{\min(\mathtt{dom}(X))\}} x_{\geq v}$$

Comme dans le cas du *log-encoding*, remarquons ici l'ajout de $\min(\texttt{dom}(X))$ pour se ramener à l'encodage d'un domaine dont le minimum est 0. Grâce à cette représentation, il est donc possible, sans perte de généralité, de représenter toute variable CSP X sous la forme d'une somme pondérée de variables booléennes, et plus généralement de littéraux, auxquels peut s'ajouter une constante μ :

$$X = \mu + \sum_{i=1}^{n} \alpha_i \ell_i$$

Notons par ailleurs que, pour encoder des contraintes CSP, il est souvent nécessaire de pouvoir déterminer si une variable est affectée à une valeur v donnée. Dans le cas du *direct-encoding*, il suffit de regarder la valeur affectée à x_v. Pour obtenir une telle variable avec l'*order-encoding*, observons dans un premier temps que la variable X est affectée à la valeur v si, et seulement si, la conjonction $x_{\geq v} \wedge \overline{x_{\geq v+1}}$ est satisfaite. Pour obtenir une variable x_v équivalente à celle utilisée dans le *direct-encoding*, il est possible de définir cette variable à l'aide de l'équivalence suivante :

$$x_v \Leftrightarrow x_{\geq v} \wedge \overline{x_{\geq v+1}}$$

qui peut être encodée sous la forme de contraintes PB, comme illustré dans la section précédente.
La situation est un peu plus complexe pour le *log-encoding*, car il est nécessaire de regarder l'affectation de *toutes* les variables booléennes utilisées dans la représentation de la variable pour connaître la valeur de la variable X. Nous proposons dans ce cas d'utiliser une forme paresseuse de *direct-encoding*, où la variable X est initialement encodée *via* le *log-encoding*, puis sous la forme du *direct-encoding* uniquement si la variable x_v a besoin d'être utilisée. Dans ce cas, la contrainte suivante assure la correspondance de la valeur affectée à X par dans les deux encodages :

$$\sum_{v \in \texttt{dom}(X)} v x_v = \min(\texttt{dom}(X)) + \sum_{i=0}^{\lceil \log_2(X) \rceil} 2^i b_i$$

Grâce à cette approche, il est toujours possible d'obtenir, pour toute variable CSP X et toute valeur $v \in \texttt{dom}(X)$ une unique variable x_v représentant l'affectation $X = v$.

3.3 Contrainte `cardinality`

Comme nous l'avons déjà mentionné, l'un des principaux avantages des solveurs PB comparés aux solveurs SAT et leur capacité à « savoir compter » de manière relativement efficace. Pour bénéficier de cet avantage, nous proposons tout d'abord un encodage pour les contraintes `cardinality`.
Une première forme de cette contrainte consiste à imposer des bornes sur le nombre de variables parmi un ensemble $\{X_1, \ldots, X_N\}$ pouvant être affectées à une valeur v donnée. Notons m et M le nombre minimum et le nombre maximum de variables X_i pouvant être affectées à v, respectivement, et notons x_v^i la variable booléenne représentant l'affectation $X_i = v$. Clairement, le nombre de x_v^i pouvant être satisfaits doit être compris entre m et M. Ainsi, la contrainte `cardinality` peut être représentée par la contrainte

$$m \leq \sum_{i=1}^{N} x_v^i \leq M$$

qui peut facilement se décomposer en deux contraintes PB. Notons que ces contraintes peuvent à elles seules permettre de représenter un nombre exponentiel de clauses suivant les valeurs de m et M [2], sans ajouter de nouvelles variables, contrairement à ce que nécessiterait un encodage CNF de ces contraintes, par exemple.
Une seconde forme de contrainte `cardinality` vise à s'assurer qu'une variable C est affectée au nombre de variables X_i qui sont affectées à une valeur v donnée. Soit $\mu + \sum_{i=1}^{n} \alpha_i \ell_i$ la représentation PB de la variable C. Cette variable doit être égale au nombre de x_v^i satisfait. Alors, en reprenant les mêmes notations que dans le cas précédent, cette seconde forme de contrainte de cardinalité peut être représentée à l'aide de l'égalité ci-dessous :

$$\sum_{i=1}^{N} x_v^i = \mu + \sum_{i=1}^{n} \alpha_i \ell_i$$

qui peut s'écrire sous la forme de la contrainte PB :

$$\sum_{i=1}^{N} x_v^i - \mu - \sum_{i=1}^{n} \alpha_i \ell_i = 0$$

Il existe par ailleurs des variantes des contraintes décrites dans cette section, où les valeurs ne sont pas des constantes mais des variables Z. Pour encoder de telles contraintes, il suffit de remplacer les variables booléennes x_v^i dans les contraintes PB définies ici par des variables x_Z^i, telles que :

$$x_Z^i \Leftrightarrow (X_i - Z = 0)$$

Notons que, en utilisant les décompositions de X_i et Z sous la forme de sommes pondérées de variables booléennes, il est possible de représenter cette contrainte à l'aide des sélecteurs présentés dans la Section 3.1.

Ajoutons qu'à l'image de la contrainte `cardinality`, des encodages PB peuvent être proposés pour d'autres contraintes de comptage, telles que `count` ou `nValues`, mais ils sont omis ici. En effet, aucune des instances XCSP3 que nous avons pu trouver ne comportant de telles contraintes, nous n'avons pas été en mesure d'évaluer ces encodages, et nous ne pouvons donc fournir aucune conclusion à leur sujet à ce jour.

3.4 Contraintes *mini-solvers*

Nous proposons maintenant des encodages pour une partie des contraintes autorisées pour les *mini-solvers* ayant participé à la compétition XCSP'19.

3.4.1 Contrainte `sum`

Une contrainte `sum` est une contrainte de la forme suivante :

$$\sum_{i=1}^{N} A_i X_i \odot k$$

où $\odot \in \{<, \leq, =, \neq, \geq, <\}$. Il est clair qu'une telle contrainte peut facilement s'écrire sous la forme d'une contrainte PB. En effet, si chacun des X_i s'écrit sous la forme $X_i = \mu^i + \sum_{j=1}^{n_i} \alpha_j^i x_j^i$, où les x_j^i sont des variables booléennes, alors la contrainte ci-dessous est équivalente à :

$$\sum_{i=1}^{N} A_i \left(\mu^i + \sum_{j=1}^{n_i} \alpha_j^i x_j^i \right) \odot k$$

qui, après développement des facteurs constants, peut être écrite sous la forme d'une contrainte PB (le cas de l'opérateur $\neq$ est traité de manière analogue à celle présentée dans la Section 3.1).

3.4.2 Contrainte `allDifferent`

La contrainte globale `allDifferent` vise à garantir que, parmi un ensemble de variables, les valeurs prises par ces variables sont toutes différentes. Nous illustrons l'encodage de cette contrainte sur l'exemple suivant :

$$\texttt{allDifferent}(X_1, \ldots, X_n)$$

Soit $\mathcal{D} = \bigcup_{i=1}^{N} \texttt{dom}(X_i)$. La sémantique de la contrainte `allDifferent` impose que chaque valeur $v \in \mathcal{D}$ soit utilisée au plus une fois parmi les variables X_i. Soit donc $v \in \mathcal{D}$, et notons x_v^i la variable booléenne représentant l'égalité $X_i = v$ pour tout $i \in 1..N$. Nous pouvons représenter cette contrainte sur v par la contrainte suivante, qui est déjà une contrainte PB :

$$\sum_{\substack{i=1 \\ v \in \texttt{dom}(X_i)}}^{N} x_v^i \leq 1$$

Cette contrainte devant s'appliquer à toutes les valeurs $v \in \mathcal{D}$, il faut ensuite répéter l'opération pour chacune des valeurs possibles. Observons que cet encodage n'est pas sans rappeler celui du problème du *pigeonhole-principle*, qui est connu pour être difficile pour les solveurs SAT utilisant la résolution [13]. Ici, l'encodage sous forme de contraintes PB permet donc, grâce à l'utilisation des solveurs PB, d'améliorer l'efficacité du raisonnement sur le sous-ensemble de contraintes correspondant à la contrainte `allDifferent`.

3.4.3 Contrainte `extension` pour les supports

Un *support* permet de lister explicitement les solutions possibles d'une contrainte. Pour encoder de telles contraintes, commençons par noter qu'une contrainte PB de la forme $\sum_{i=1}^{n} \ell_i \geq n$ permet de représenter la conjonction des littéraux ℓ_i, i.e., $\bigwedge_{i=1}^{n} \ell_i$.

Partant de cette observation, nous pouvons encoder un support (unique) ayant la forme suivante :

$$(X_i \mid 1 \leq i \leq N) = (v_i \mid 1 \leq i \leq N)$$

à l'aide de la contrainte PB :

$$\sum_{i=1}^{N} x_{v_i}^i \geq N$$

où $x_{v_i}^i$ est la variable booléenne représentant l'égalité $X_i = v_i$ pour tout $i \in 1..N$. Dans le cas (plus fréquent) où plusieurs tuples t sont autorisés par une contrainte de type support, il nous faut ajouter un sélecteur s_t pour les contraintes associées à chacun de ces tuples, pour obtenir des contraintes de la forme :

$$s_t \Rightarrow \sum_{i=1}^{N} x_{v_i}^i \geq N$$

En effet, l'ensemble des tuples autorisés doit être vu comme une disjonction. En particulier l'un des tuples doit nécessairement être affecté : il faut alors ajouter la clause $\bigvee s_t$ pour terminer l'encodage du support.

Remarquons enfin que, si le symbole $\star$ est utilisé dans les tuples autorisés à la place d'une valeur v_i (pour symboliser la possibilité d'utiliser une valeur quelconque pour la variable X_i), il suffit d'omettre le littéral correspondant à la variable X_i dans la contrainte ci-dessus.

4 Résultats expérimentaux

Cette section présente quelques résultats expérimentaux relatifs à l'utilisation des encodages pseudo-booléens présenté dans cet article sur différents ensembles de problèmes de satisfaction. Afin d'évaluer les performances de notre approche, nous avons implanté nos encodages dans le solveur `Sat4j` [16], et exécuté plusieurs variantes de ce solveur, notées `Sat4j` + $\mathcal{S}$ dans la suite de cette section, où $\mathcal{S}$ représente le nom la variante utilisée. Lorsque aucun encodage n'est spécifié, c'est la combinaison du *direct-encoding* et du *log-encoding* qui est utilisé pour représenter les domaines des variables. Le solveur `Sat4j` + `OrderEncodingBothPOS2020` utilise quant à lui l'*order-encoding*, tandis que `Sat4j` + `OrderEncodingPrimitiveBothPOS2020` exploite en plus cet encodage pour encoder les contraintes primitives de manière plus efficaces (à la manière de ce qui est proposé dans [25]). Dans le même temps, nous avons exécuté le solveur PB `RoundingSat` [9] sur l'encodage PB fournit par notre implantation dans `Sat4j`.

Nous comparons ces implantations avec différents solveurs CSP de l'état de l'art, à savoir `ACE`[2] – le nouvel avatar d'`AbsCon` – `Choco` [23] et `sCOP` [25]. Nous avons également exécuté le solveur CSP proposé avec la bibliothèque `Sat4j`, et noté `Sat4j` + `SAT` [6].

2. https://github.com/xcsp3team/ace

Tous les solveurs ont été lancés sur un cluster de machines équipées 32 Go de RAM et de 2 processeurs quadricœur Intel Xeon X5550 cadencé à 2.66 GHz. Le temps d'exécution était limité à 1200 secondes.

Dans le cadre de nos expérimentations, nous avons utilisé deux benchmarks, composés de problèmes de satisfaction provenant de la distribution XCSP [4, 19]. Le premier, noté $\mathcal{I}_{\texttt{card}}$, se compose de 5 familles de problèmes et de 145 instances. Il est composé d'instances CSP comportant des contraintes de type `cardinality` provenant du site XCSP[3]. Pour obtenir cet ensemble, nous avons dû retirer certaines instances qui comportaient des contraintes qui n'étaient pas gérées par notre approche (en particulier, des contraintes `lex`). Le second, noté $\mathcal{I}_{\texttt{XCSP18-19}}$, correspond à l'ensemble des instances des compétitions XCSP18 et XCSP19 utilisées dans le *track mini-solvers*, résultant en 32 familles de problèmes et 371 instances.

Nous présentons ci-dessous l'étude expérimentale de notre approche sur ces deux benchmarks. Ces analyses ont été réalisées avec Metrics[4], un outil d'analyse expérimentale garantissant la reproductibilité des résultats.

4.1 Etude de $\mathcal{I}_{\texttt{card}}$

La Figure 1 présente un aperçu des performances des différents solveurs exécutés sur l'ensemble d'instance $\mathcal{I}_{\texttt{card}}$.

Cette figure est un *cactus-plot*. Chacune des lignes correspond à un solveur, et permet de connaître le nombre d'instances résolues en un temps donné par ce solveur. Ici, nous pouvons observer que le solveur `RoundingSat` parvient à résoudre plus d'instances que les autres solveurs, en résolvant par exemple 16 instances de plus que le solveur `ACE` dans la limite des 1200 secondes. Nous pouvons de plus remarquer que les solveurs utilisant un encodage SAT, en l'occurence, `Sat4j` + `SAT` et `sCOP`, qui utilisent respectivement le *direct-encoding* et l'*order-encoding* ont des difficultés sur ces instances qui requiert des capacités de comptage (rappelons que ces instances contiennent des contraintes `cardinality`).

Afin de comparer plus précisément les deux premiers solveurs du *cactus-plot*, nous traçons à la Figure 2 un *scatter-plot* comparant `ACE` et `RoundingSat`.

Ici, chaque point représente une instance, et sa couleur la famille de problèmes à laquelle elle appartient. L'abscisse d'un point correspond au temps d'exécution de `RoundingSat` sur cette instance, et son ordonnée celui de `ACE` sur cette même instance. Cette figure montre clairement que `RoundingSat` parvient à résoudre très efficacement les instances de la famille `CarSequencing` (qui comportent essentiellement des contraintes `cardinality` et `sum`). De manière générale, nous avons pu observer que ce comportement se généralisait aux différents solveurs PB considérés dans notre étude, même si les différentes variantes de `Sat4j` restent moins efficaces que `RoundingSat`.

3. `https://xcsp.org`
4. `https://github.com/crillab/metrics`

4.2 Etude de $\mathcal{I}_{\texttt{XCSP18-19}}$

Afin d'évaluer les capacités de notre approche à résoudre des problèmes plus généraux, nous l'évaluons maintenant sur les instances de l'ensemble $\mathcal{I}_{\texttt{XCSP18-19}}$, qui contient une plus large variété de problèmes et de contraintes. Pour les contraintes des *mini-solvers* que nous n'avons pas présentées dans la Section 3, nous utilisons soit une combinaison classique des contraintes présentées dans cet article, soit un encodage sous forme de clauses (qui reprend donc un encodage existant, dépendant de l'encodage choisi pour les domaines des variables concernée). De cette manière, notre implantation est capable d'encoder toutes les instances de l'ensemble $\mathcal{I}_{\texttt{XCSP18-19}}$, La Figure 3 nous présente une vue globale des solveurs exécutés sur cet ensemble.

Le premier solveur, avec une belle avance sur les autres, est `ACE`. Il est suivi par `Choco` et `sCOP` respectivement en seconde et troisième place. Le premier solveur `Sat4j` arrive en quatrième position. Il s'agit de `Sat4j`+`Resolution` : ce solveur PB réalise en fait une analyse de conflit à la manière d'un solveur SAT classique, en inférant une clause de manière paresseuse chaque fois qu'une contrainte PB est rencontrée pendant l'analyse. L'avantage de cette approche est qu'elle permet de combiner la concision des contraintes PB et l'efficacité des structures de données des solveurs SAT.

Néanmoins, l'efficacité de ce solveur ne permet pas de généraliser les bonnes performances observées sur l'ensemble $\mathcal{I}_{\texttt{card}}$. Cela peut en partie s'expliquer par le fait que de (trop) nombreuses contraintes doivent encore être encodées à l'aide de clauses pour les problèmes considérés, ce qui ne permet pas d'exploiter la pleine puissance du système de preuves implanté par les solveurs PB.

Nous pouvons de plus observer que l'ordre des solveurs PB n'est pas le même que pour $\mathcal{I}_{\texttt{card}}$. Cela peut s'expliquer par la complémentarité des approches implantées par les différents solveurs comme cela a notamment été décrit dans [15, 17].

5 Conclusion

Dans cet article nous avons proposé d'exploiter différents encodages booléens des domaines des variables d'un problème de satisfaction de contraintes pour définir des encodages sous forme de contraintes pseudo-booléennes. Nous avons tout particulièrement considéré les contraintes reconnues par les *mini-solvers* de la compétition XCSP3, ainsi que différentes variantes de la contrainte `cardinality`. Le principal avantage des encodages proposés est qu'ils permettent d'exploiter le pouvoir d'inférence des solveurs PB, et notamment leur capacité à compter efficacement. L'analyse expérimentale a montré que nos encodages, combinés à l'utilisation de solveurs PB, permet en effet de résoudre efficacement des problèmes composés principalement de contraintes `sum` et de contraintes `cardinality`. Néanmoins, ces bonnes performances ne se généralisent pas à des problèmes comportant d'autres types de contraintes, pour lesquels les solveurs CP natifs et les solveurs fondés sur SAT restent plus performant.

A ce jour, notre approche ne permet de résoudre que

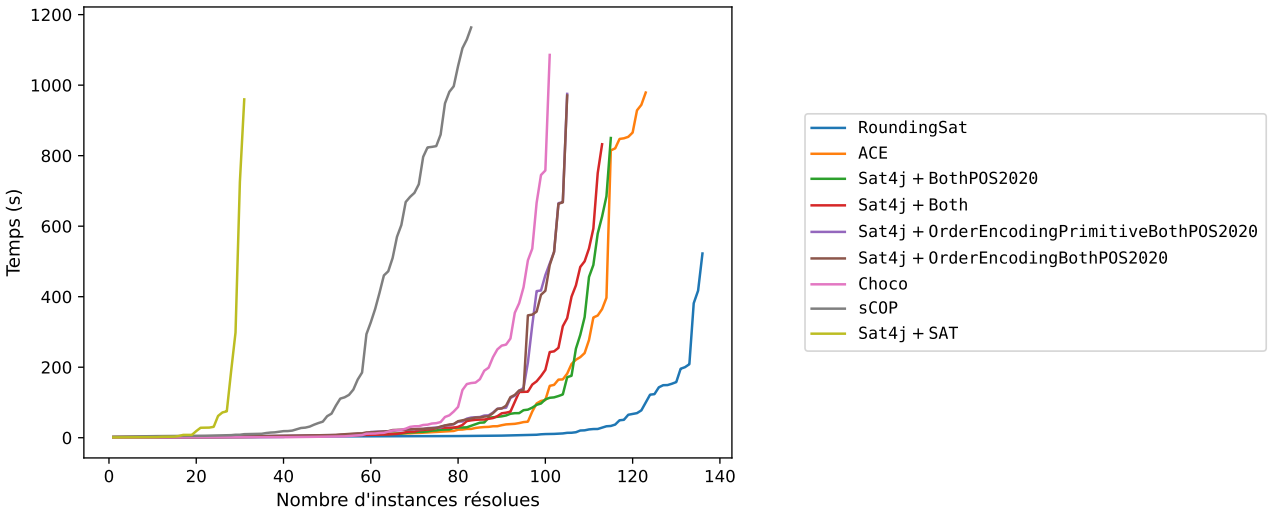


FIGURE 1 – *Cactus-plot* des solveurs sur l'ensemble d'instances $\mathcal{I}_{\texttt{card}}$.

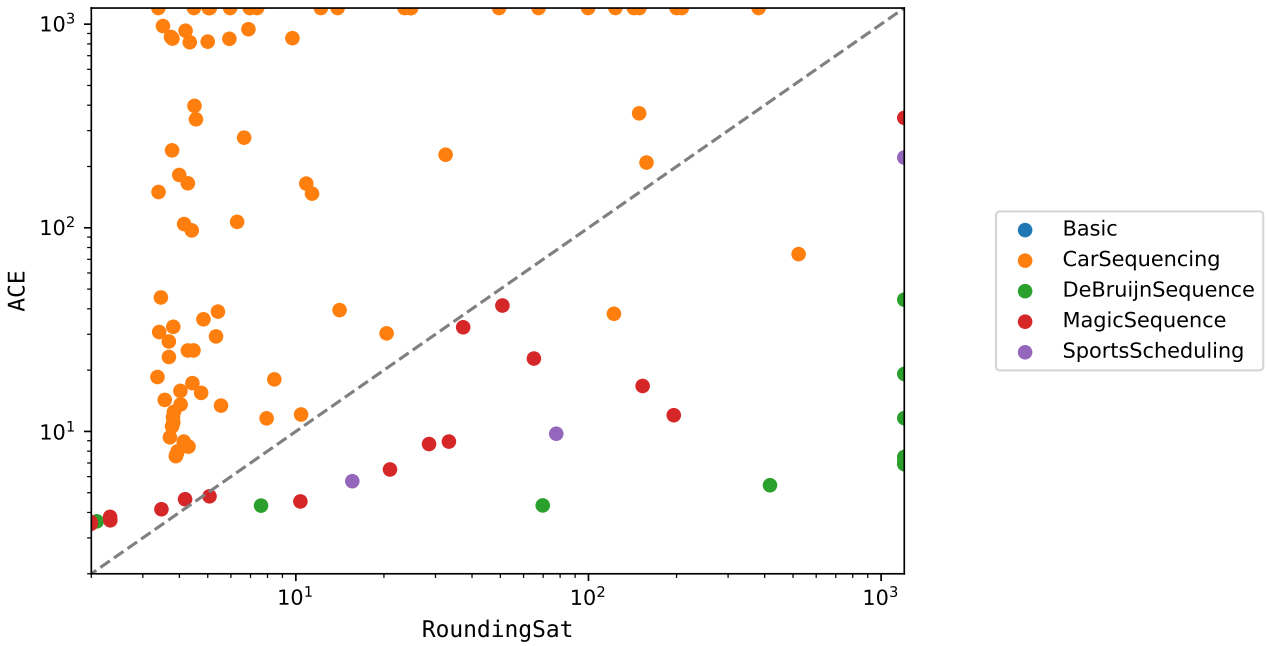


FIGURE 2 – *Scatter-plot* comparant les temps d'exécution (en secondes) d'`ACE` et de `RoundingSat` sur l'ensemble $\mathcal{I}_{\texttt{card}}$.

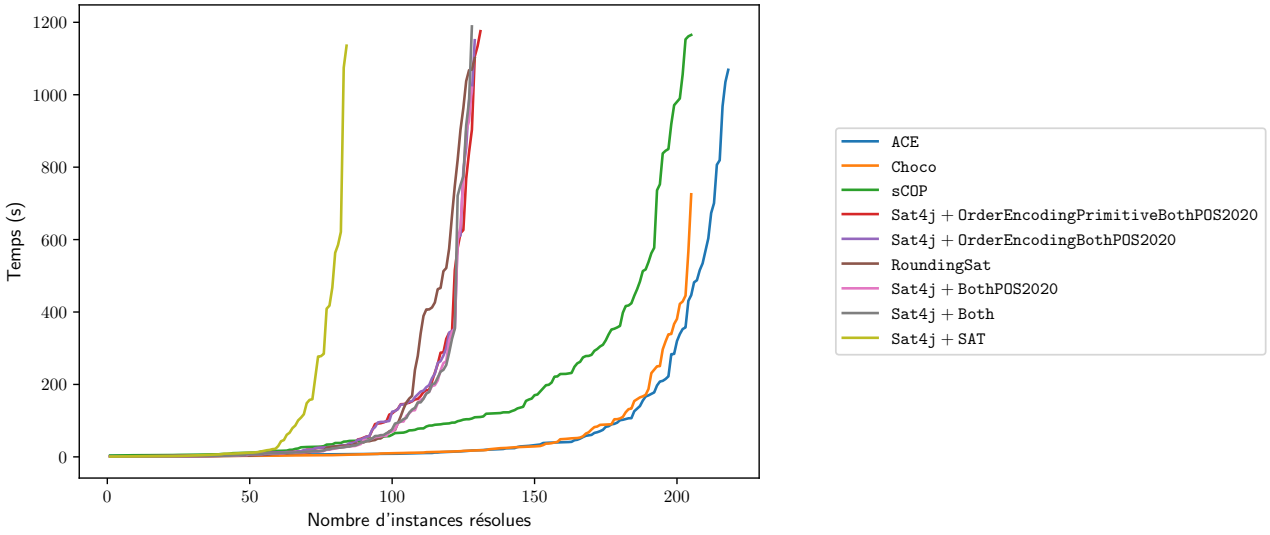


FIGURE 3 – *Cactus-plot* des solveurs sur l'ensemble d'instances $\mathcal{I}_{\texttt{XCSP18-19}}$

des problèmes comportant un sous-ensemble restreint de contraintes. A court terme, nous envisageons de définir des encodages pour d'autres types de contraintes, afin de pouvoir soumettre notre approche à la prochaine compétition XCSP. Nous souhaiterions de plus étudier de nouveaux encodages pour améliorer les performances des solveurs PB dans la résolution de problèmes CSP. A plus long terme, nous envisageons d'exploiter la complémentarité des différents paradigmes de résolution de CSP (natif, fondé sur SAT ou fondé sur PB) pour tirer le meilleur de chacune de ces approches.

Références

[1] Ramón Béjar, Cèsar Fernández, and Francesc Guitart. Encoding basic arithmetic operations for sat-solvers. In René Alquézar, Antonio Moreno, and Josep Aguilar-Martin, editors, *Artificial Intelligence Research and Development - Proceedings of the 13th International Conference of the Catalan Association for Artificial Intelligence, l'Espluga de Francolí, Tarragona, Spain, 20-22 October 2010*, volume 210 of *Frontiers in Artificial Intelligence and Applications*, pages 239–248. IOS Press, 2010.

[2] Belaid Benhamou, Lakhdar Sais, and Pierre Siegel. Two proof procedures for a cardinality based language in propositional calculus. In Patrice Enjalbert, Ernst W. Mayr, and Klaus W. Wagner, editors, *STACS 94, 11th Annual Symposium on Theoretical Aspects of Computer Science, Caen, France, February 24-26, 1994, Proceedings*, volume 775 of *Lecture Notes in Computer Science*, pages 71–82. Springer, 1994.

[3] Christian Bessiere, George Katsirelos, Nina Narodytska, Claude-Guy Quimper, and Toby Walsh. Decompositions of all different, global cardinality and related constraints. In Craig Boutilier, editor, *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 419–424, 2009.

[4] Frédéric Boussemart, Christophe Lecoutre, Gilles Audemard, and Cédric Piette. Xcsp3-core : A format for representing constraint satisfaction/optimization problems. *CoRR*, abs/2009.00514, 2020.

[5] Stephen A. Cook. The Complexity of Theorem-proving Procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[6] Ines Lynce Daniel Le Berre. Csp2sat4j : A simple csp to sat translator. In *Proceedings of the second CSP Competition*, 2008.

[7] Heidi E. Dixon and Matthew L. Ginsberg. Inference methods for a pseudo-boolean satisfiability solver. In *AAAI'02*, pages 635–640, 2002.

[8] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *Theory and Applications of Satisfiability Testing*, pages 502–518, 2004.

[9] Jan Elffers and Jakob Nordström. Divide and conquer : Towards faster pseudo-boolean solving. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 1291–1299, 2018.

[10] Marco Gavanelli. The log-support encoding of CSP into SAT. In Christian Bessiere, editor, *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings*, volume 4741 of *Lecture Notes in Computer Science*, pages 815–822. Springer, 2007.

[11] Gaël Glorian. Nacre. In *Solver Descriptions of XCSP3 Competition 2018*, 2018.

[12] Ralph E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, pages 275–278, 1958.

[13] Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39 :297 – 308, 1985. Third Conference on Foundations of Software Technology and Theoretical Computer Science.

[14] J. N. Hooker. Generalized resolution and cutting planes. *Annals of Operations Research*, 12(1) :217–239, 1988.

[15] Daniel Le Berre, Pierre Marquis, and Romain Wallon. On weakening strategies for PB solvers. In Luca Pulina and Martina Seidl, editors, *Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*, volume 12178 of *Lecture Notes in Computer Science*, pages 322–331. Springer, 2020.

[16] Daniel Le Berre and Anne Parrain. The SAT4J library, Release 2.2, System Description. *Journal on Satisfiability, Boolean Modeling and Computation*, 7 :59–64, 2010.

[17] Daniel Le Berre and Romain Wallon. On dedicated cdcl strategies for pb solvers. In *Theory and Applications of Satisfiability Testing - SAT 2021 - 24th International Conference, Proceedings*, page to appear, 2021.

[18] C. Lecoutre. *Constraint Networks : Techniques and Algorithms*. ISTE/Wiley, 2009.

[19] Christophe Lecoutre and Nicolas Szczepanski. PYCSP3 : modeling combinatorial constrained problems in python. *CoRR*, abs/2009.00326, 2020.

[20] Joao Marques-Silva and Karem A. Sakallah. Grasp : A search algorithm for propositional satisfiability. *IEEE Trans. Computers*, pages 220–227, 1999.

[21] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff : Engineering an Efficient SAT Solver. In *Proceedings of the 38th Annual Design Automation Conference*, DAC '01, pages 530–535, New York, NY, USA, 2001. ACM.

[22] Jakob Nordström. On the Interplay Between Proof Complexity and SAT Solving. *ACM SIGLOG News*, 2(3) :19–44, August 2015.

[23] Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. Choco solver documentation. *TASC, INRIA Rennes, LINA CNRS UMR*, 6241, 2016.

[24] Naoyuki Tamura, Akiko Taga, Satoshi Kitagawa, and Mutsunori Banbara. Compiling finite linear CSP into SAT. In Frédéric Benhamou, editor, *Principles and Practice of Constraint Programming - CP 2006, 12th International Conference, CP 2006, Nantes, France, September 25-29, 2006, Proceedings*, volume 4204 of *Lecture Notes in Computer Science*, pages 590–603. Springer, 2006.

[25] Naoyuki Tamura, Akiko Taga, Satoshi Kitagawa, and Mutsunori Banbara. Compiling finite linear CSP into SAT. *Constraints An Int. J.*, 14(2) :254–272, 2009.

[26] Toby Walsh. SAT v CSP. In Rina Dechter, editor, *Principles and Practice of Constraint Programming - CP 2000, 6th International Conference, Singapore, September 18-21, 2000, Proceedings*, volume 1894 of *Lecture Notes in Computer Science*, pages 441–456. Springer, 2000.

Décodage Optimal de Modèle de Markov Cachés avec Contraintes de Cohérence

Alexandre Dubray[1], Guillaume Derval[2], Siegfried Nijssen[1], Pierre Schaus[1]

[1] ICTEAM, UCLouvain, Belgique
{prenom}.{nom}@uclouvain.be
[2] Montefiore, ULiège, Belgique
gderval@uliege.be

Abstract

Les modèles de Markov cachés (MMC) sont des modèles statistiques interprétables qui spécifient des distributions sur des séquences de symboles en les générant à partir d'états cachés. En pratique, cela se fait en trouvant le chemin le plus long dans un graphe pondérés dirigé, en couche et sans cycle, à l'aide de la programmation dynamique. Dans certaines applications, bien que les états cachés soient inconnus, nous soutenons qu'il est connu que certains éléments observables doivent avoir le même état caché. Nous proposons une approche de programmation en nombres entiers (PE), de programmation dynamique (PD) et de Séparation et Évaluation (S&E) pour résoudre le problème. Nos expériences montrent que l'approche PD ne s'adapte pas bien ; S&E s'adapte mieux à un petit nombre de contraintes imposées à de nombreux éléments et PE est l'approche la plus robuste lorsque de nombreuses petites contraintes sont imposées.

1 Introduction

Les modèles de Markov cachés (MMC) sont une classe de modèles probabilistes dans lesquels on suppose que les symboles des séquences sont générés à partir d'états cachés. Pour une séquence de données observées, on suppose qu'il existe une séquence d'états cachés qui l'a générée avec une probabilité donnée ; déterminer les états cachés qui ont généré les symboles observés est utile pour comprendre les données. Les MMCs ont été utilisés dans diverses applications telles que la prédiction de structure des protéines [5], l'extraction de données de trajectoires [6] ou la reconnaissance vocale [4]. Le problème de décodage des MMCs consiste à trouver la séquence la plus probable d'états cachés, pour une séquence observée, et est généralement résolu par l'algorithme de Viterbi [7], qui a un temps d'exécution polynomial.

Dans ce travail, nous soutenons que dans de nombreuses applications, un meilleur décodage peut être fait en exploitant des connaissances de base entre les séquences qui ne sont pas prises en compte dans le décodage classique des MMCs, dans lequel plusieurs séquences sont décodées indépendamment. Nous nous concentrons sur les contraintes de cohérence entre deux observations de certaines séquences, éventuellement la même, qui imposent que le même état caché soit sélectionné pour les éléments. En pratique, de telles connaissances existent. Par exemple, dans l'assignation de partie du discours, dans lequel les mots de phrases sont annotés en fonction de leur nature, il est probable que, dans une même phrase, plusieurs occurrences du même mot peu commun doivent recevoir la même annotation. Dans un problème d'affectation de l'état d'un camion, où la tâche consiste à déterminer l'état d'un certain nombre de camions qui sont suivis (sont-ils en train de rouler, de se reposer, ...), de telles contraintes peuvent être imposées aux camions qui se trouvent dans la même zone au même moment. En imposant des contraintes selon lesquelles les symboles de la séquence doivent provenir du même état, l'algorithme de Viterbi ne peut plus être utilisé.

2 Définition du Problème

Dans cette section, nous formalisons le problème de la recherche du chemin le plus long dans un graphe dirigé sans cycle (GDSC) en couches sous contraintes de cohérence. La résolution de ce problème permet de résoudre également le problème du décodage des MMCs. D'abord, nous introduisons les notations et notions de couche et contrainte de cohérence dans un GDSC, puis nous exprimons le problème de trouver le plus long chemin cohérent dans celui-ci.

2.1 Plus Long Chemin dans un GDSC à niveaux avec Contraintes de Cohérence

Nous définissons le problème du décodage dans les MMCs sur des GDSC annotés. Soit $G = (V, E)$ un graphe avec V l'ensemble des nœuds et E l'ensemble des arêtes. Chaque nœud $v \in V$ a une annotation provenant d'un ensemble $\mathcal{L}$, noté l_v et V est divisé en T couches $L_1, \ldots, L_T$ telles que $V = \bigcup_{j=1,\ldots,T} L_j$ et $L_i \cap L_j = \emptyset \; \forall i \neq j$. Dans chaque couche, deux nœuds n'ont pas la même annotation. Ainsi, lorsqu'il n'y a pas d'ambiguïté, un nœud peut être identifié par son annotation. Nous désignons par $e = (l, l', t) \in E$ une arête allant du nœud l à la couche L_t au nœud l' à la couche L_{t+1} $(1 \leq t < T)$ avec un poids w_e, où les poids peuvent être à la fois positifs et négatifs. Dans le problème du décodage de MMC, dans un tel graphe, il y a un nœud source et un nœud cible, notés s et s', et chaque couche

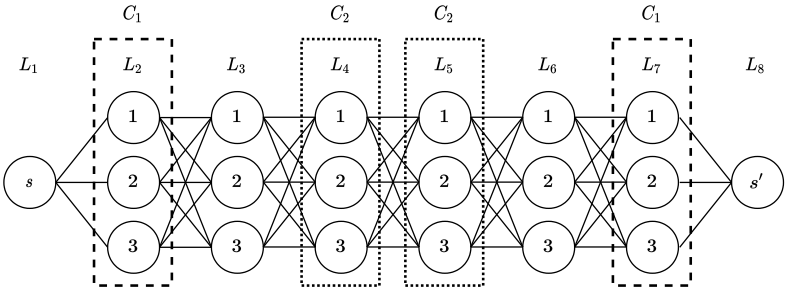


FIGURE 1 – Exemple d'un GDSC en couches pour le problème de décodage dans un MMC avec trois états cachés et deux contraintes de cohérence. Les arêtes du GDSC sont orientées de gauche à droite et les annotations sur les nœuds représentent les états cachés ou les nœuds source et cible.

intermédiaire a le même nombre de nœuds, représentant les états cachés, comme montré dans la Figure 1.

Un chemin dans G de L_1 à L_T sélectionne un nœud par couche et peut être identifié par la séquence de nœuds sur le chemin. Plus formellement, soit $P = \langle P_1, \ldots, P_T \rangle \in \mathcal{L}^T$ un chemin de L_1 à L_T tel que $P_i \in L_i$. Le coût de P est la somme des poids des arcs du chemin : $\sum_{t=1}^{T-1} w_{(P_t, P_{t+1}, t)}$.

Une contrainte de cohérence est spécifiée en identifiant un ensemble de couches pour lesquelles des nœuds avec la même annotation doivent être sélectionnés dans chaque couche du chemin. Plus formellement, $C = \{C_1, \ldots, C_k\}$ sont k contraintes de cohérence avec $C_i = \{c_1^i, \ldots, c_{k_i}^i\} \subseteq \{1, \ldots, T\}$ et $C_i \cap C_j = \emptyset$ pour $i \neq j$. L'ensemble de toutes les couches contraintes est noté $L_C = \bigcup_{i=1}^{k} C_i$. Nous définissons également un vecteur $c \in \{0, \ldots, k\}^T$ qui donne pour chaque couche l'indice de sa contrainte ou 0 si la couche n'est pas contrainte. Par exemple, dans la Figure 1 nous avons $c = \langle 0, 1, 0, 2, 2, 0, 1, 0 \rangle$. Un chemin P est dit cohérent si toutes les contraintes de cohérence sont respectées. Le problème de la recherche du plus long (i.e. de poids maximum) chemin cohérent est donc formalisé comme suit :

$$P^\star = \arg\max_{P \in \mathcal{L}^T} \sum_{t=1}^{T-1} w_{(P_t, P_{t+1}, t)} \quad (1)$$

$$\text{s.t.} P_{c_1^i} = \ldots = P_{c_{k_i}^i} \qquad \forall C_i \in C \quad (2)$$

L'importance de ce problème pour le décodage des MMCs est qu'une instance de ce problème, y compris le GDSC et ses poids, peut être construite pour un problème spécifique de décodage HMM sur une séquence de symboles. Dans chaque couche du GDSC, tous les états cachés du modèle de Markov sont répétés ; les poids correspondent aux probabilités logarithmiques de générer un symbole en passant d'un état caché à un autre. Finalement, notons que le problème définit par les Equations (1)-(2) est NP-difficile, mais la preuve, qui consiste à réduire 3-SAT à notre problème, est omise pour plus de brièveté.

3 Résolution du Problème

Dans cette section, nous expliquons trois approches pour résoudre le problème définit par les équations (1)-(2). Nous fournissons d'abord une formulation du programme à l'aide de la programmation en nombres entiers (PE). Ensuite, une approche de programmation dynamique (PD) est expliquée, suivie d'une méthode de Séparation et Évaluation (S&E).

3.1 Formulation en Programmation Entière

En programmation en nombres entiers, le problème consistant à trouver le plus long chemin dans le GDSC peut être formulé en utilisant une variable binaire pour chaque arête du graphe. Soit $z_{ij}^t \in \{0, 1\}$ une variable binaire indiquant si $e_{(i,j,t)} \in E$ fait partie du chemin ou non. Nous dénotons aussi $I_i^t = \sum_{j \in L_{t-1}} z_{ji}^{t-1}$ le flux entrant dans le nœud $i \in L_t$ et $O_i^t = \sum_{j \in L_{t+1}} z_{ij}^t$ son flux sortant. Le modèle d'optimisation est le suivant :

$$\max \sum_{t=1}^{T-1} \sum_{\substack{i \in L_t \\ j \in L_{t+1}}} w_{(i,j,t)} z_{ij}^t$$

$$\text{s.t. } I_i^t = O_i^t \qquad \forall 1 < t < T,\ \forall i \in L_t \quad (3)$$

$$\sum_{\substack{i \in L_t \\ j \in L_{t+1}}} z_{ij}^t = 1 \qquad \forall 1 \leq t < T \quad (4)$$

$$I_i^t = I_i^{t'} \qquad \forall i \in L_t;\ t, t' \in C_k; C_k \quad (5)$$

La Contrainte (3) garantit que si une arête entrante est sélectionnée dans un nœud, l'une de ses arêtes sortantes est également sélectionnée. La Contrainte (4) garantit qu'une arête entre chaque couche est sélectionnée, y compris L_1 (qui peut avoir plusieurs noeuds). La Contrainte (5) est la contrainte de cohérence.

3.2 Programmation Dynamique

Pour résoudre le problème sans contrainte, l'algorithme de Viterbi [7] est l'approche classique de programmation dynamique. La relation de récurrence calcule la valeur du plus long chemin de L_1 à un nœud $i \in L_t$ à partir de la couche L_{t-1} et la stocke dans une table de taille $T \times |\mathcal{L}|$. Les entrées de la table, $V[T, i]$, sont calculées comme suit :

$$\begin{cases} 0 & \text{if } t = 1 \\ \max\limits_{j \in L_{t-1}} V[t-1, j] + w_{(j,i,t-1)} & \text{autrement} \end{cases} \quad (6)$$

Proportion de Contraintes	0.00	0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.80	0.90	1.00
PD	6.00	360.10	324.40	299.30	278.60	255.70	232.70	211.90	192.20	173.90	157.80
PE	588.20	841.67	911.33	926.60	927.20	1007.20	1001.25	1169.67	1061.67	1141.00	1047.00
S&E	**2.63**	**3.59**	**3.59**	**3.55**	**3.43**	**3.42**	**3.34**	**3.31**	**3.27**	**3.21**	**2.93**

TABLE 1 – Temps d'exécution en seconde des méthodes, en fonction de la proportion de contraintes dans le modèle, pour le problème d'assignation d'état aux camions

et la valeur du plus long chemin est donnée par $\max_{i \in L_T} V[T, i]$.

Cette équation utilise le fait que le graphe est organisé en couches et qu'un chemin se terminant à la couche L_t provient toujours de la couche L_{t-1}. Ainsi, le plus long chemin vers un nœud $i \in L_t$ est l'un des plus longs chemins vers un nœud dans L_{t-1} plus l'arête vers i. Cependant, lorsqu'on ajoute des contraintes de cohérence, cette équation ne fonctionne plus, car elle ne prend pas en compte la cohérence entre les couches. Nous résolvons ce problème en ajoutant des affectations de nœud aux contraintes de cohérence dans la relation de récurrence ; en affectant un nœud à une contrainte, nous affectons le même nœud à toutes les couches de la contrainte. Soit $P_C = \langle P_{C_1}, \ldots, P_{C_k} \rangle \in (\mathcal{L} \cup \{\})^k$ une affectation de nœud aux contraintes de cohérence avec $P_{C_i} = \perp$ si aucun nœud n'est affecté à C_i. Ensuite, si $P_{C_i} \neq \perp$, le chemin de L_1 à L_T doit passer par P_{C_i} pour chaque couche L_t avec $t \in C_i$. Nous définissons l'opérateur d'affectation $P_C|_{j,l}$ qui affecte l à P_{C_j}. Les valeurs des plus longs chemins sont maintenant être stockées dans un tableau de taille $T \times |\mathcal{L}| \times k^{|\mathcal{L}|}$, en tenant compte des affectations possibles des nœuds aux contraintes. Les entrées $V[T, i, P_C]$ de la table sont calculées comme suit :

$$\begin{cases} 0 & \text{si } t = 1 \\ \max\limits_{l \in L_{t-1}} V[t-1, l, P_C] + w_{(l,i,t-1)} & \text{si } L_{t-1} \notin L_C \\ V[t-1, P_{C_{c[t]}}, P_C] + w_{(P_{C_{c[t]}}, i, t-1)} & \text{si } L_{t-1} \in L_C \\ & \quad \wedge P_{C_{c[t]}} \neq \perp \\ \max\limits_{l \in L_{t-1}} V[t-1, l, P_C|_{c[t],l}] + w_{(l,i,t-1)} & \text{si } L_{t-1} \in L_C \\ & \quad \wedge P_{C_{c[t]}} = \perp \end{cases}$$

et la valeur du plus long chemin peut être calculée par $\max_{i \in L_T} V[T, i, \langle \perp, \ldots, \perp \rangle]$ si $L_T \notin L_C$ et $\max_{i \in L_T} V[T, i, \langle \perp, \ldots, \perp \rangle|_{c[T],i}]$ sinon. Les deux premiers cas de cette équation sont les mêmes que pour l'Equation (6) car il n'y a pas de contraintes à considérer. Cependant, lorsque la couche L_{t-1} est contrainte, il y a deux situations. S'il existe un choix pour cette contrainte dans P_C, alors pour être cohérent avec P_C, le chemin doit passer par le nœud choisi. Dans ce cas, il n'est pas nécessaire de considérer les autres nœuds de la couche. Cependant, lorsqu'il n'y a pas encore de nœud affecté à cette contrainte, alors chaque nœud $j \in L_{t-1}$ doit être considéré pour calculer le plus long chemin. Dans ce cas, le vecteur P_C est mis à jour pour refléter le choix effectué.

3.3 Séparation et Évaluation

Soit $P_C \in (\mathcal{L} \cup \{\perp\})^k$, comme pour le PD, un vecteur d'affectations de nœuds pour la contrainte de cohérence. La recherche commence à partir du vecteur $\langle \perp, ..., \perp \rangle$. L'idée de cette méthode est d'assigner les valeurs P_{C_i} et de calculer le plus long chemin de L_1 à L_T tout en étant cohérent avec P_C. Initialement, certains P_{C_i} ne sont pas assignés ; tant que la contrainte n'est pas assignée, on l'ignore et le coût est une borne supérieure de la solution optimale dans la branche. Par exemple, sur la Figure 1, si nous avons $P_C = \langle 2, \perp \rangle$, le plus long chemin de L_1 à L_T peut être décomposé en le plus long chemin de L_1 à L_2, puis de L_2 à L_7 et finalement de L_7 à L_8.

En pratique, un tableau de taille $T \times |\mathcal{L}|$, noté V, est utilisé pour stocker les valeurs des chemins les plus longs des couches de L_C vers les autres couches. À la racine de l'arbre de recherche, le tableau V est rempli avec l'Équation (6) puisqu'aucune contrainte de cohérence n'est imposée. Lorsqu'une valeur P_{C_i} est attribuée, le tableau n'a pas besoin d'être recalculé en entier. Regardons la Figure 1 comme un exemple. Lorsque la recherche attribue $P_{C_1} = 2$, les couches contraintes par C_1 agissent comme de nouvelles couches sources. Les valeurs calculées, dans V, pour les couches L_1 à L_2 représentent toujours les valeurs qu'une équation récursive calcule pour le plus long chemin de L_1 à L_2 et ne doivent donc pas être recalculées. Supposons maintenant que la recherche attribue $P_{C_2} = 1$. Les valeurs dans V pour L_1 à L_4 et pour L_8 sont toujours valides, et seules les valeurs de L_5 à L_7 doivent être mises à jour.

Notons que lorsque toutes les arêtes ont un poids négatif, comme pour le problème de décodage d'un MMC, alors les valeurs du tableau V peuvent être calculées entre les contraintes de cohérence, même si elles ne sont pas assignées. Dans l'exemple de la Figure 1, la conséquence est que lorsque P_{C_1} est fixé à 2, les valeurs à partir de L_2 ne sont calculées que jusqu'à L_4 et non L_7. Puisque les arêtes n'ont que des poids négatifs, cela donne toujours une borne supérieure (moins forte) sur la solution optimale, mais plus rapide à calculer.

4 Résultats Expérimentaux

Dans cette section, nous analysons les trois méthodes présentées dans la Section 3 sur deux applications de MMC ayant des caractéristiques différentes en termes de longueur de séquence et de nombre de contraintes de cohérence. Le PE est résolu avec le solveur Gurobi [3] et pour le S&E nous utilisons la variation de l'algorithme qui ne supporte que les poids négatifs, car nous n'expérimentons qu'avec des MMCs et cela donne, dans nos expériences, les meilleurs résultats. [1].

1. Le code source peut être trouvé à `https://github.com/AlexandreDubray/consistent-viterbi`.

Jeu de Donnée	conll2000					treebank						brown		
Nombre de Contraintes	2	3	4	5	6	2	3	4	5	6	7	2	3	4
PD	153.2	128.7	O.O.M	O.O.M	O.O.M	56.0	1132.0	O.O.M	O.O.M	O.O.M	O.O.M	1047.4	O.O.M	O.O.M
PE	97.6	137.6	**86.0**	**128.1**	**130.6**	34.6	**33.4**	**32.1**	**32.2**	**47.6**	**48.9**	485.3	**480.9**	**790.1**
S&E	**18.1**	**29.1**	280.7	T.O.	T.O.	**14.0**	63.5	113.1	745.7	T.O	T.O	**77.4**	1052.2	T.O.

TABLE 2 – Temps d'exécution, en seconde, des méthodes en fonction du nombre de contraintes de cohérence pour le problème d'annotation de PDD. Le temps d'exécution maximum a été mis à 1 heure et son dépassement est indiqué par T.O. tandis que les saturations de mémoire sont indiquées par O.O.M.

Identification d'arrêts de Camions Les MMCs ont été utilisés pour identifier les arrêts d'activité dans les trajectoires de camions [6]. Quatre états cachés représentent si le camion roule, se trouve dans un embouteillage, se repose ou effectue des actions liées au travail. Dans ce contexte, il est naturel de supposer que les camions situés dans des zones géographiques similaires effectuent la même activité, ce qui est traduit en contrainte de cohérence. Nous expérimentons sur des trajectoires de camions ayant utilisé le réseau routier belge décrit dans [1]. Les contraintes de cohérence sont imposées aux points de conduite sur l'autoroute et aux arrêts tombant dans i) les zones de repos, ii) les zones industrielles ainsi que iii) les autoroutes. Seule une fraction d'entre eux est contrainte, entre 0% et 100%.
La Table 1 montre le temps d'exécution des méthodes avec différentes proportions. Pour la méthode PD, les couches contraintes sont plus rapides à calculer, car seulement les assignations de contraintes cohérentes sont considérées. Donc le temps d'exécution de cette méthode diminue lorsque les contraintes sont plus grandes. Au contraire, la méthode PE augmente avec la taille des contraintes, car elle n'utilise pas la structure en couche du graphe pour la résolution. Une augmentation de la taille des contraintes induit donc une augmentation du nombre de contraintes, ce qui tend à augmenter le temps d'exécution. Le temps d'exécution de S&E est stable par rapport à la taille des contraintes.

Annotation de Partie du Discours L'objectif de cette application est d'attribuer à chaque mot d'une phrase, ou d'un texte, une étiquette de partie du discours (PDD). Le paquet NLTK Python [2] fournit des jeux de données de phrases avec des PDD annotés. Nous expérimentons sur trois ensembles de données avec les 12 balises PDD universelles. Pour chaque jeu de données, les contraintes de cohérence sont composées des couches associées à la même PDD, mais pour éviter d'avoir un problème trop contraint, seuls 20% de chaque composant sont conservés. Afin d'évaluer l'impact du nombre de contraintes, le problème est résolu avec seulement un sous-ensemble d'entre elles. La Table 2 montre le temps d'exécution des méthodes en fonction du nombre de contraintes de cohérence. On constate que la méthode PE est la seule à résoudre le problème pour un nombre plus élevé de contraintes. Les méthodes S&E et PD prennent trop de temps, avec une limite de mémoire atteinte pour la PD lorsque le nombre de contraintes augmente.

5 Conclusion et Travail Futur

Dans de nombreuses applications résolues à l'aide de modèles de Markov cachés, des contraintes de cohérence entre les séquences peuvent être trouvées, mais ne sont pas utilisées dans l'algorithme de décodage classique. Dans ce travail, nous avons formalisé ce problème comme étant la recherche du plus long chemin dans un graphe acyclique dirigé en couches avec des contraintes de cohérence sur les couches du graphe. Nous avons montré qu'une approche de Séparation et Évaluation est la meilleure pour résoudre un problème avec un petit nombre de grande contrainte, tandis que la Programmation Entière doit être préférée quand le nombre de contrainte est plus grand.
Dans ce travail, nous nous sommes concentrés sur les contraintes de cohérence, en imposant que le même nœud soit sélectionné entre différentes couches. Cependant, dans certaines applications, il peut être acceptable d'avoir des ensembles de nœuds qui peuvent apparaître ensemble dans les couches d'une contrainte de cohérence (par exemple, un arrêt de non-activité et un arrêt de repos, dans le problème d'assignation d'état de camions). L'impact d'autre contraintes logiques sur la méthode S&E pourrait aussi être étudié.

Références

[1] Arnaud Adam, Olivier Finance, and Isabelle Thomas. Monitoring trucks to reveal belgian geographical structures and dynamics : From gps traces to spatial interactions. *Journal of Transport Geography*, 2021.

[2] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python : analyzing text with the natural language toolkit*. 2009.

[3] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2022.

[4] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 1989.

[5] Erik LL Sonnhammer, Gunnar Von Heijne, Anders Krogh, et al. A hidden markov model for predicting transmembrane helices in protein sequences. In *Ismb*, 1998.

[6] Mehdi Taghavi, Elnaz Irannezhad, and Carlo G Prato. Identifying truck stops from a large stream of gps data via a hidden markov chain model. In *ITSC*, 2019.

[7] Andrew Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*, 1967.