Quatorzièmes Journées Francophones de Programmation par Contraintes

JFPC 2018

Amiens, 13-15 juin 2018

JFPC 2018

Actes des Quatorzièmes Journées Francophones
de Programmation par Contraintes

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

Organisation

MIS, Université de Picardie Jules Verne

Président des journées

Chu-Min Li, MIS, Université de Picardie Jules Verne

Président du comité de programme

Cyril Terrioux, LIS, Aix-Marseille Université

Comité de programme

Président :

Cyril Terrioux	LIS, Aix-Marseille Université

Membres :

David Allouche	MIAT, INRA Toulouse
Gilles Audemard	CRIL, Université d'Artois
Vincent Barichard	LERIA, Université d'Angers
Quentin Cappart	École Polytechnique de Montréal
Clément Carbonnel	Université d'Oxford
Remi Coletta	LIRMM, Université de Montpellier
Thi-Bich-Hanh Dao	LIFO, Université d'Orléans
Sophie Demassey	CMA, MINES ParisTech
Gilles Dequen	MIS, Université de Picardie Jules Verne
Steven Gay	Google France
Djamal Habet	LIS, Aix-Marseille Université
Emmanuel Hebrard	LAAS, Université de Toulouse
Marie-José Huguet	LAAS, Université de Toulouse
Arnaud Lallouet	Huawei Technologies Ltd
Frederic Lardeux	LERIA, Université d'Angers
Nadjib Lazaar	LIRMM, Université de Montpellier
Olivier Lhomme	IBM France
Chu-Min Li	MIS, Université de Picardie Jules Verne
Xavier Lorca	LS2N, IMT Atlantique
Samir Loudni	GREYC, Université de Caen Basse-Normandie
Margaux Nattaf	Mines de Saint Etienne
Samba Ndojh Ndiaye	LIRIS, Université Claude Bernard Lyon 1
Alexandre Niveau	GREYC, Université de Caen Basse-Normandie
Anastasia Paparrizou	CRIL, Université d'Artois
Lionel Paris	LIS, Aix-Marseille Université
Marie Pelleau	I3S, Université de Nice-Sophia Antipolis
Guillaume Perez	Cornell University
Thierry Petit	LS2N, IMT Atlantique
Eric Piette	LAMSADE, Université Paris-Dauphine
Cédric Piette	CRIL, Université d'Artois
Jean-Charles Regin	I3S, Université de Nice-Sophia Antipolis
Pierre Schaus	ICTEAM, Université catholique de Louvain
Mohamed Siala	Insight, University College Cork
Laurent Simon	Labri, Bordeaux Institute of Technology
Christine Solnon	LIRIS, INSA Lyon
Elise Vareilles	IMT Mines Albi, France + ETS Montréal, Canada
Julien Vion	LAMIH, Université de Valenciennes et du Hainaut-Cambrésis
Mohamed Wahbi	Insight, University College Cork

Rapporteurs additionnels : Philippe Refalo, Hajer Saada.

Comité d'organisation

Président :

Chu-Min Li	MIS, Université de Picardie Jules Verne

Membres :

Gilles Dequen	MIS, Université de Picardie Jules Verne
Laure Devendeville	MIS, Université de Picardie Jules Verne
David Durand	MIS, Université de Picardie Jules Verne
Sorina Ionica	MIS, Université de Picardie Jules Verne
Céline Joiron	MIS, Université de Picardie Jules Verne
Gilles Kassel	MIS, Université de Picardie Jules Verne
Clément Lecat	MIS, Université de Picardie Jules Verne
Yu Li	MIS, Université de Picardie Jules Verne
Corinne Lucet	MIS, Université de Picardie Jules Verne

Préface

Comme tous les ans, les Journées Francophones de Programmation par Contraintes (JFPC) sont l'occasion pour notre communauté de se réunir et d'échanger dans un cadre à la fois scientifique et convivial. Cette année, la co-localisation des Journées d'Intelligence Artificielle Fondamentale (JIAF) nous offre en plus l'opportunité d'interagir avec la communauté francophone travaillant sur l'Intelligence Artificielle Fondamentale et de renforcer les liens existants entre nos deux communautés.

Pour cette quatorzième édition, seulement 19 papiers ont été soumis. Il s'agit là du plus faible nombre de papiers soumis depuis la création des JFPC. J'ose espérer que cette baisse n'est que temporaire et que le nombre de soumissions repartira à la hausse dès la prochaine édition. Hormis le nombre de soumissions, cette édition s'inscrit dans la continuité. La diversité des travaux soumis est toujours aussi remarquable tant du point de vue des thématiques abordées (SAT, ASP, logique modale, CSP, ...) que de la nature des travaux et des domaines d'application. Les JFPC étant avant tout un lieu d'échange, le taux de sélection est souvent très élevé. Pour autant, cela n'enlève rien à la qualité des papiers acceptés, bien au contraire. Cette année ne fait pas exception avec, par exemple, un taux proche de 95 %. Comme les années précédentes, les papiers acceptés sont, pour partie, des traductions ou des résumés d'articles déjà publiés, notamment dans des conférences internationales comme CP, SAT, UAI ou ICTAI. Les autres présentent des travaux nouveaux sous la forme d'articles courts ou longs.

Lors de cette édition, comme lors des deux précédentes, un prix viendra récompenser le meilleur article étudiant. Une nouvelle fois, il s'agira de mettre en avant la qualité du travail mené par les doctorants de notre communauté. J'en profite d'ailleurs pour remercier les organisateurs de CP, qui aura lieu cette année à Lille du 27 au 31 août, pour leur généreuse contribution à ce prix.

Je profite de ces quelques mots pour remercier chaleureusement Stefan Woltran (DBAI, TU Wien) et Christophe Lecoutre (CRIL, Université d'Artois) d'avoir accepté de participer à ces JFPC. Stefan Woltran donnera un exposé invité sur la résolution des problèmes (Q)SAT et #SAT par le biais de la programmation dynamique et de la notion de décomposition arborescente. Quant à Christophe Lecoutre, il nous proposera un tutoriel sur l'enseignement de la programmation par contraintes à l'aide de l'API de modélisation MCSP3. Enfin, nous aurons également l'opportunité de suivre un exposé donné par Torsten Schaub dans le cadre des JIAF.

Cette année, le comité de programme comptait 38 membres, représentant 26 institutions issues de 6 pays. Dans un premier temps, les membres de ce comité se sont investis dans le travail de relecture en vue d'écrire des rapports les plus constructifs possibles. Puis, ils ont été amenés à pré-sélectionner les trois papiers finalistes pour le prix du meilleur article étudiant. Enfin, ils ont désigné deux papiers parmi les papiers candidats, pour une double présentation aux JFPC et à la Conférence Nationale en Intelligence Artificielle (CNIA). En effet, cette conférence, qui se déroulera début juillet à Nancy, a notamment pour objectif que les diverses tendances de l'intelligence artificielle y soient représentées. À ce titre, elle a sollicité les JFPC afin que quelques papiers issus de cette quatorzième édition soient présentés également lors de CNIA. Je tiens à remercier ici les membres du comité de programme pour l'ensemble du travail accompli et pour sa qualité.

Je terminerai en remerciant l'Association Française de Programmation par Contraintes (AFPC) qui œuvre, année après année, pour le développement et la promotion de la PPC et apporte tout son soutien aux différentes initiatives et manifestations la concernant comme, par exemple, les JFPC. Enfin, je souhaite mettre en lumière tout le travail réalisé depuis plusieurs mois par Chu-Min Li et tout le comité d'organisation pour que ces journées soient une réussite. Un grand merci à eux.

Cyril Terrioux
Président du comité de programme des JFPC 2018

Table des matières

Solving (Q)SAT problems via Tree Decomposition and Dynamic Programming

Stefan Woltran
DBAI, TU Wien
woltran@dbai.tuwien.ac.at

Abstract

Parameterized algorithms tackle computationally hard problems such that solving certain instances can be done efficiently due to inherent structural features. One prominent parameter to capture structure is treewidth, which is defined in terms of tree decompositions. The actual solving is then performed by dynamic programming (DP) along a traversal of a tree decomposition of the instance at hand. However, the concrete implementation of such algorithms is often tedious and requires certain engineering efforts.

In this talk, we first recall how the SAT-problem can be solved with this particular method. We extend these ideas towards quantified Boolean formulas (QBFs) and present a system that solves satisfiability for QBFs via DP using BDDs as an internal data-structure. We sketch the necessary tuning efforts in order to make this system a serious competitor for state-of-the-art QBF solvers. As a second example, we present a recent implementation for the model-counting problem (#SAT) that works on the GPU. Finally, we discuss a new theoretical result on projected model-counting.

Utilisation de l'opérateur OWA pour le clustering conceptuel équitable

Noureddine Aribi[1] **Abdelkader Ouali**[2] **Yahia Lebbah**[1] **Samir Loudni**[2]

[1] Lab. LITIO, University of Oran 1, 31000 Oran, Algeria.

[2] Normandie Univ, UNICAEN, ENSICAEN, CNRS, GREYC, 14000 Caen, France.

{aribi.noureddine,lebbah.yahia}@univ-oran1.dz {abdelkader.ouali,samir.loudni}@unicaen.fr

Résumé

Nous proposons une approche de clustering conceptuel qui exploite la notion *d'équité* en optimisation multi-agents. L'idée consiste à s'assurer que les clusters obtenus soient équilibrés, i.e. approximativement de même taille. Dans notre approche, chaque agent représente un concept et l'utilité d'un agent correspondant à une mesure spécifique (par exemple la fréquence du concept). Le problème consiste alors à trouver la meilleure satisfaction cumulative tout en mettant l'accent sur un compromis équitable entre tous les agents individuels. Pour déterminer la solution optimale de meilleur compromis, nous avons exploité une formulation équitable de l'opérateur des moyennes pondérées ordonnées (en anglais, Ordered Weighted Averages - OWA), avec un jeu de poids particulièrement adapté à la recherche de solutions équitables. Les expériences effectuées sur les jeux de données de l'UCI et sur des instances issues d'une application réelle (ERP) montrent que notre approche trouve efficacement des cleusterings de bonne qualité.

Abstract

We propose an equitable conceptual clustering approach based on multi-agent optimization. In the context of conceptual clustering, each cluster is represented by an agent having its own satisfaction and the problem consists in finding the best cumulative satisfaction while emphasizing a fair compromise between all individual agents. The fairness goal is achieved using an equitable formulation of the Ordered Weighted Averages (OWA) operator. Experiments performed on UCI datasets and on instances coming from real application ERP show that our approach efficiently finds clusterings of consistently high quality.

1 Introduction

Structurer les données dans le processus de découverte des connaissances est une tâche fondamentale qui permet de mieux comprendre les données et d'identifier des groupements d'objets (appelés clusters) en fonction d'une mesure de similarité prédéfinie. En pratique, les utilisateurs aimeraient souvent effectuer d'autres actions, telle que l'interprétation sémantique de chaque cluster. Les méthodes telles que le clustering conceptuel répondent à cette question en tentant de trouver des descriptions des clusters au moyen de concepts formels.

De nombreuses approches ont été proposées pour le clustering conceptuel. Les approches traditionnelles [12, 8] combinent la formation des clusters et des descriptions. D'autres techniques [19, 18] ont plutôt choisi de découpler la recherche des descriptions - avant ou après l'étape du clustering. Plus récemment, des approches de programmation par contraintes (PPC) [4] et de programmation linéaire en nombres entiers (PLNE) [16] ont été proposées pour résoudre le problème du clustering conceptuel optimal dans un cadre déclaratif. Elles combinent deux techniques exactes : dans un premier temps, un outil de fouille de données dédié (i.e. `LCM` [21]) est utilisé pour calculer l'ensemble de tous les concepts formels et, dans un deuxième temps, la PLNE ou la PPC est utilisée pour sélectionner les meilleurs k clusters (i.e. les concepts) qui optimisent un critère donné. La plupart des mesures d'optimisation utilisées dans ces approches conduisent à des clusters déséquilibrés. S'assurer que les clusters obtenus soient (approximativement) équilibrés, permet de rendre les résultats du clustering plus utiles et exploitables [2, 23].

Cet article introduit le concept *d'équité* et de *solutions équitablement efficaces* pour le problème du clustering conceptuel dans un contexte multi-agents, où chaque agent représente un concept et possède sa propre utilité liée à une mesure spécifique (e.g. la fré-

quence). Ici, l'équité correspond à l'idée de favoriser des solutions qui partagent équitablement la satisfaction entre les agents [9]. L'équité a été entièrement étudiée par la communauté d'optimisation multicritères [10], et formalisée à travers les trois propriétés : **(i) La symétrie** qui signifie que tous les agents ont la même importance. Par exemple, les deux vecteurs d'utilités $(5, 3, 0)$ et $(0, 3, 5)$ sont équivalents. **(ii) La Pareto-monotonie** exprime qu'une solution $(x_1, x_2, ..., x_n)$ est meilleure que la solution $(y_1, y_2, ..., y_n)$, si et seulement si, $x_i \geq y_i$ pour tout i, avec au moins une inégalité stricte. **(iii) Le Principe de transfert** formalise une distribution équitable des utilités [20]. L'intuition est que tout transfert entre deux utilités inéquitables x_i et x_j, qui préserve la moyenne des utilités, améliorerait l'utilité globale.

Une manière courante de traiter le concept de solutions équitablement efficaces consiste à définir des fonctions d'agrégation qui remplissent les propriétés ci-dessus. Ceci définit une famille d'agrégations équitables qui sont *Schur-convexe* [11]. Dans la littérature, il existe plusieurs fonctions pour agréger les utilités des agents au moyen de *fonction d'utilité collective* (CUF). Les agrégations les plus utilisées sont `maxMin`, `maxSum` et `minDev`. La fonction `maxSum` combine linéairement des utilités, tandis que la fonction `minDev` minimise l'écart entre le meilleur et le pire des utilités. Le principe de transfert n'est pas assuré par `maxMin` et `minDev`, sur toutes les utilités, conduisant ainsi à *l'effet de noyade* [7]. La fonction `maxSum` est entièrement compensatoire et ne capture donc pas l'idée d'équité.

Dans ce papier, nous proposons une approche efficace pour le clustering conceptuel équitable, qui fait appel à deux techniques exactes : (1) extraction des motifs fermés (en utilisant l'algorithme `LCM` [21]) ; (2) sélection des meilleurs clusters à l'aide de la Programmation Linéaire en Nombres Entiers (PLNE) qui exploite une fonction d'agrégation équitable basée sur `OWA`, remplissant les trois propriétés d'équité cités ci-dessus. Notre opérateur `OWA` utilise des poids spécifiques proposés par Golden et Perny [9] qui assurent la propriété de Schur-convexité. Les expériences réalisées sur des ensembles de données UCI et sur un ensemble d'instances provenant d'une application réelle ERP montrent que notre approche basée sur `OWA` trouve efficacement des clusterings de hautes qualités, par rapport aux approches concurrentes.

La section 2 introduit les concepts utilisés dans ce document. La section 3 décrit nos modèles PLNE pour le clustering conceptuel équitable. Nous discutons les travaux connexes dans la section 4 avant de montrer les performances de notre approche dans la section 5. La section 6 conclut et esquisse les orientations de nos recherches futures.

2 Préliminaires

2.1 Concepts formels et clustering conceptuel

Concepts formels. Soit $\mathcal{D}$ un ensemble de m transactions (numérotées de 1 à m), $\mathcal{I}$ un ensemble de n items (numérotés de 1 à n), et $R \subseteq T \times \mathcal{I}$ une relation binaire qui lie les transactions aux items : $(t, i) \in R$ si la transaction t contient l'item i : $i \in t$. Un itemset (ou *motif*) est un sous-ensemble non nul de $\mathcal{I}$. Par exemple, la table 1a contient une base transactionnelle $\mathcal{D}$ avec $m = 11$ transactions $t_1, \ldots, t_{11}$ décrit par $n = 8$ items. L'*extent* d'un ensemble $I \subseteq \mathcal{I}$ d'items est l'ensemble des transactions contenant tous les items de I, i.e. $ext(I) = \{t \in \mathcal{D} | \ \forall i \in I, (t, i) \in R\}$. L'*intent* d'un sous-ensemble $T \subseteq \mathcal{D}$ est l'ensemble des items contenus dans toutes les transactions dans T, c'est-à-dire, $int(T) = \{i \in \mathcal{I} | \ \forall t \in T, (t, i) \in R\}$. Ces deux opérateurs induisent une connexion de Galois entre $2^{\mathcal{D}}$ et $2^{\mathcal{I}}$, i.e. $T \subseteq ext(I) \Leftrightarrow I \subseteq int(T)$. Une paire telle que $(I = int(T), T = ext(I))$ s'appelle **concept formel**. Cette définition définit une propriété de **fermeture** sur l'ensemble de données $\mathcal{D}$, $\texttt{closed}(I) \Leftrightarrow I = int(ext(I))$. Un ensemble d'items I pour lequel $\texttt{closed}(I) = vrai$ s'appelle *motif fermé*. En utilisant $ext(I)$, on peut définir la *fréquence* d'un concept : $\texttt{freq}(I) = |ext(I)|$, sa *diversité* : $\texttt{divers}(I) = \sum_{t \in ext(I)} |\{i \in \mathcal{I} \mid \ (i \notin I) \wedge (i \in t)\}|$, et sa *taille* : $\texttt{size}(I) = |\{i \mid i \in I\}|$. On note $\mathcal{C}$ l'ensemble de tous les concepts formels.

Clustering Conceptuel. Le clustering consiste à partitionner un ensemble de transactions en groupes relativement homogènes. Le clustering conceptuel vise à fournir une description distincte de chaque groupe - le concept caractérisant les transactions qu'il contient. Ce problème peut être formulé comme suit : "trouver un ensemble de k clusters, chacun décrit par un motif fermé $P_1, P_2, ..., P_k$, couvrant toutes les transactions sans aucun chevauchement entre les clusters". Par exemple, la table 1c illustre trois classifications possibles pour $k = 3$. Une fonction d'évaluation f qui optimise un critère donné peut être utilisée pour exprimer la qualité du clustering. Différents critères d'optimisation peuvent être considérés : maximiser la *somme des fréquences* des concepts sélectionnés ; minimiser la *somme des diversités* des concepts sélectionnés. Par exemple, pour la base transactionnelle $\mathcal{D}$ et $k = 3$, minimiser $f(P_1, ..., P_k) = \sum_{1 \leq i \leq k} \texttt{divers}(P_i)$ fournit un clustering s_1, avec une valeur optimale égale à 18 (voir la table 1c). La solution $s_1 = (1, 1, 9)$ engendre un gros cluster (de taille 9) couvrant la plupart des transactions, et deux clusters qui couvrent une seule transaction. Toutefois, cette solution est moins intéressante que celle où tous les clusters sont de taille comparable.

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

(a) Données transactionnelles $\mathcal{T}$.

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

(b) $(a_{t,c})$ matrice associée à l'ensemble de données $\mathcal{D}$.

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

(c) 3 clustering conceptuels pour $k = 3$.

TABLE 1 – Exemple de motivation.

La façon la plus courante pour obtenir des clusters plus équilibrés est de considérer des fonctions d'optimisation dédiées :

- *maximiser la fréquence minimale* (`maxMin`). On cherche des solutions dans lesquelles la fréquence minimale des concepts est la plus grande possible.

- *minimiser l'écart entre les fréquences des clusters* (`minDev`). On impose une petite différence entre les fréquences des clusters : `Min max(freq`$(P_1), \ldots)$ - `min(freq`$(P_1), \ldots)$.

Cependant, ces deux fonctions souffrent de l'*effet de noyade* [7]. Sur le `maxMin` (resp. `minDev`), le principe de transfert n'est assuré que sur l'utilité `min` (resp. `max`), et donc les utilités intermédiaires ne sont pas nécessairement équitables. Pour assurer l'équité, on considère, dans la section suivante, un opérateur sophistiqué qui raisonne sur l'ensemble des utilités.

2.2 Optimisation multi-agent équitable

Soit $N = \{1, .., n\}$ un ensemble de n agents. Une solution d'un problème d'optimisation multi-agent est caractérisée par un vecteur d'utilités $x = (x_1, ..., x_n) \in \mathbb{R}^n_+$, où x_i représente l'utilité (degré de satisfaction) du i^{me} agent. Les vecteurs d'utilités sont généralement comparés en utilisant la relation de dominance de Pareto (P-dominance). La P-dominance faible $\succsim_P$ entre deux vecteurs d'utilités x, x' est définie par : $x \succsim_P x' \Leftrightarrow [\forall i \in N, x_i \geq x'_i]$, alors que la P-dominance stricte $\succ_P$ entre x et x' est donnée par : $x \succ_P x' \Leftrightarrow [x \succsim_P x' \ \wedge not(x' \succsim_P x)]$. Une solution x^* est Pareto-optimale (a.k.a *efficace*) si et seulement s'il n'y a pas de solution x qui domine x^*. L'ensemble des solutions Pareto-optimales forme le front de Pareto $P = \{x \mid \nexists x', x' \succ_P x\}$. La P-dominance peut être formulée comme suit : $max\ \{(x_1, ..., x_n) : x \in Q\}$, où Q est l'ensemble des solutions réalisables. La P-dominance peut conduire à un grand ensemble de solutions incomparables. De plus, la P-dominance est insensible aux solutions extrémales (*outliers*). Pour favoriser les vecteurs d'utilités *équilibrés*, il est nécessaire de raffiner la relation de P-dominance. L'intuition principale derrière le concept d'équité consiste à choisir des solutions qui partagent équitablement la satisfaction entre les agents [20]. Formellement, une relation de dominance équitable $\succsim_\parallel$ doit remplir trois propriétés principales [11, 9] : **(i) Symétrie**. On considère un vecteur d'utilité $x \in \mathbb{R}^n_+$. Pour toute permutation σ sur N, on a $(x_{\sigma(1)}, ..., x_{\sigma(n)}) \sim (x_1, ..., x_n)$. Cela signifie que tous les agents ont la même importance. Par exemple, le vecteur d'utilité $(5, 3, 0)$ est considéré comme équivalent au vecteur d'utilité $(0, 3, 5)$. **(ii) P-Monotonie.** Pour tous $x, y \in \mathbb{R}^n_+$, $x \succsim_P y \Rightarrow x \succsim_\parallel y$ et $x \succ_P y \Rightarrow x \succ_\parallel y$. **(iii) Principe de transfert.** (a.k.a transfert *Pigou-Dalton* dans la théorie du choix social) Soit $x \in \mathbb{R}^n_+$ et $x_i > x_j$ pour certains $i, j \in N$. Soit e^z un vecteur tel que $\forall i \neq z, e^z_i = 0$ et $e^z_z = 1$. Pour tout ϵ où $0 < \epsilon \leq \frac{x_i - x_j}{2}$, on obtient $x - \epsilon e^i + \epsilon e^j \succsim_\parallel x$. Toute légère amélioration de x_j au détriment (réduction) de x_i, qui préserve la *moyenne des utilités*, produirait une meilleure distribution des utilités parmi les agents et améliorerait par conséquent l'utilité globale de la solution. Par exemple, si l'on considère deux vecteurs d'utilités $x = (\mathbf{11}, 10, \mathbf{7}, 10)$ et $y = (\mathbf{9}, 10, \mathbf{9}, 10)$, alors le principe de transfert implique que $y = (\mathbf{9}, 10, \mathbf{9}, 10)$, car il y a un transfert de taille $\epsilon = 2$ (i.e. $\frac{x_1 - x_3}{2}$), qui permet d'avoir y à partir de x. La combinaison de la P-monotonie et du principe de transfert conduit à ce que l'on appelle *dominance de Lorenz généralisée* définie dans [5] (pour plus de détails, voir [9, 11]).

2.3 Fonctions d'agrégation équitables

Une manière d'évaluer la qualité d'un vecteur d'utilité consiste à agréger les utilités individuelles avec une *fonction d'utilité collective* [13] $G : \mathbb{R}_+^n \to \mathbb{R}_+$, ce qui améliore le bien-être global avec $max\{G(x) : x \in Q\}$. La fonction G peut être une combinaison linéaire d'utilités individuelles (i.e $G(x) \stackrel{\text{def}}{=} sum(x)$), ce qui ne convient pas au contexte d'équité. Une autre façon de construire G est basée sur la fonction `min` (i.e. $G(x) \stackrel{\text{def}}{=} min(x)$), mais elle est sensible à l'*effet de noyade* [7]. D'autres raffinements de la fonction `min` existent (e.g. le `min` augmenté, `lexmin` [3]), mais ne résolvent pas vraiment le problème, car tous sont sensibles à l'*effet de noyade*. Afin de garantir des agrégations équitables, G devrait être conforme aux trois propriétés d'équité. La manière la plus connue est d'utiliser une fonction Schur-convexe ψ, qui préserve les trois propriétés d'équité : $x \succsim_{\parallel} y \Leftrightarrow \psi(x) \geq \psi(y)$. Précisément, quand une fonction d'agrégation G est Schur-convexe [11], alors il s'agit d'une agrégation équitable [10]. Ainsi, les fonctions Schur-convexes jouent un rôle clé dans les agrégations équitables (pour plus de détails, voir [11, 10]). Dans la section suivante, nous introduirons une fonction d'agrégation qui assure l'équité.

2.4 Moyennes pondérées ordonnées (OWA)

Cette section se concentre sur les moyennes pondérées ordonnées (en anglais, Ordered Weighted Averages – OWA) [22] définies comme suit :

$$G^w(x) = \sum_{i=1}^{n} w_i x_{\sigma(i)} \qquad (1)$$

où $w = (w_1, \ldots, w_n) \in [0,1]^n$ et $x_{\sigma(1)} \leq x_{\sigma(2)} \leq \cdots \leq x_{\sigma(n)}$. OWA fournit une famille de compromis entre les opérateurs `sum` et `min`. Le premier peut être obtenu par $G^{(1/n,1/n,\ldots,1/n)}$ et le second par $G^{(1,0,\ldots,0)}$. Golden et Perny [9] proposent des coefficients pour la fonction d'agrégation OWA pour qu'elle soit Schur-convexe :

Théorème 1 *[9] Soient les coefficients suivants de l'agrégation OWA : $W(x) = \sum_{k=1}^{n} sin(\frac{(n+1-k)\pi}{2n+1}) x_{\sigma(k)}$. W est une fonction Schur-convexe.*

Le théorème 1 est fondamental, puisque les fonctions Schur-convexes assurent l'équité [10, 9].

3 Modèles PLNE

Cette section décrit les différents modèles PLNE pour trouver un clustering conceptuel équitable. Notre approche suit l'approche en deux étapes de [16] : (1) un outil dédié à l'extraction de motifs fermés (i.e. `LCM` [21]) est utilisé pour calculer l'ensemble $\mathcal{C}$ de tous les motifs fermés ; (2) La PLNE est utilisée pour sélectionner un sous-ensemble de $\mathcal{C}$ qui est une partition de l'ensemble $\mathcal{D}$ de transactions et qui optimise un critère donné. Pour favoriser les clusterings équitables, nous améliorons la deuxième étape avec des contraintes supplémentaires assurant une agrégation `OWA` équitable.

3.1 Modèles PLNE pour l'opérateur `OWA`

Cette section présente notre première formulation PLNE, appelée modèle `OWA` de base, pour le clustering conceptuel équitable utilisant un opérateur `OWA` équitable. Ensuite, nous montrons comment ce modèle de base peut être amélioré en post-traitant les contraintes `OWA`. Soit $\mathcal{D}$ une base transactionnelle avec m transactions définies sur un ensemble de n items $\mathcal{I}$. Soit $\mathcal{C}$ l'ensemble de p motifs fermés (par rapport à la mesure de fréquence) représentant les clusters candidats. Soit $a_{t,c}$ une matrice binaire $m \times p$ où $(a_{t,c} = 1)$ ssi $c \subseteq t$, i.e., la transaction t appartient à l'extension du motif fermé c. La matrice $(a_{t,c})$ associée à l'ensemble de données $\mathcal{D}$ de la table 1a est décrite avec la table 1b. Soit v la liste des utilités des motifs fermés (e.g., fréquence, diversité, etc.). Pour chaque motif fermé ($c \in \mathcal{C}$), une variable binaire x_c est associée t.q. $(x_c = 1)$ ssi le cluster c est sélectionné.

(a) Modèle PLNE de base pour `OWA`. La figure 1a donne le modèle PLNE pour un clustering conceptuel équitable. Il utilise deux types de contraintes : des contraintes de clustering et des contraintes `OWA` modélisant l'opération de tri requise par l'opérateur `OWA` :

- Contraintes de clustering conceptuel. Les contraintes (C1) exigent que le sous-ensemble de motifs fermés sélectionnés soit une partition de $\mathcal{D}$. Les contraintes (C2) imposent une borne inférieure k_{min} et/ou une borne supérieure k_{max} sur le nombre de motifs fermés sélectionnés.

- Contraintes `OWA`. La fonction objectif et les contraintes (O1) et (O2) implémentent une formulation linéaire connue [15] de l'opérateur `OWA` pour le clustering conceptuel, où les coefficients ω sont fixés par le théorème 1. Comme expliqué dans la section 2.4, `OWA` est une somme pondérée sur des utilités triées. C'est pourquoi nous avons introduit r, qui représente la version triée du vecteur d'utilités v. M est une constante suffisamment grande. Soit z une matrice booléenne de taille $|\mathcal{C}|^2$ utilisée pour formuler les contraintes de tri (O1) et (O2). Ces contraintes imposent que le vecteur d'utilités $v \,.\, x$ des motifs fermés soit trié par ordre croissant correspondant aux coefficients ω de `OWA`. Ces contraintes de tri sont expliquées

en détail dans [15]. Il s'ensuit que la k^{me} plus petite valeur d'utilité r_k aura le k^{me} poids le plus élevé ω_k. La fonction objectif maximise la somme pondérée en utilisant les poids ω de `OWA` donnés par le théorème 1.

(b) Modèle PLNE amélioré pour `OWA`. Afin de trouver un clustering conceptuel équitable, nous proposons un modèle optimisé (voir la fig. 1b) comme suit : (1) Les contraintes de tri (O1) et (O2) sont utilisées lorsque les valeurs d'utilité sont données en compréhension. Comme les valeurs d'utilité des concepts formels sont connues à l'avance, le tri peut être effectué immédiatement après la recherche de motifs fermés. Nous notons $v^{\uparrow}$ la version triée de v par ordre croissant. (2) Nous associons les poids ω de `OWA` aux valeurs d'utilité triées, de sorte que toutes les utilités égales auront le même poids. Pour nos expériences, nous avons utilisé le modèle `OWA` amélioré. Nos résultats préliminaires ont montré que le modèle `OWA` amélioré surclasse clairement le modèle `OWA` de base en termes de temps CPU. Ceci est dû au fait que n^2 contraintes supplémentaires et n^2 variables supplémentaires sont utilisées pour coder les contraintes de tri du modèle `OWA`. Ceci constitue une forte limitation en termes de taille des données qui pourraient être considérées.

Proposition 1 *Les deux modèles PLNE basique et amélioré sont équivalents.*

Preuve 1 *Les deux modèles* `OWA` *utilisent les poids ω du théorème 1, ce qui assure une agrégation équitable.* `OWA` *amélioré est une optimisation du modèle de base : (1) Il utilise un tri à priori des utilités (pas besoin de contraintes de tri) ; (2) Le même poids est attribué aux utilités égales (même niveau de satisfaction), ce qui préserve directement la conformité avec le théorème 1. Ainsi, les deux modèles* `OWA` *sont équivalents.* □

(c) Stabilité numérique du modèle PLNE. L'ensemble des motifs fermés est souvent énorme, ce qui conduit à un très grand vecteur ω dans le modèle `OWA` de base, et affecte la stabilité numérique du solveur. Le modèle `OWA` optimisé s'attaque à ce problème, en attribuant le même poids aux utilités égales. Cela permet de résoudre des instances du monde réel dans nos expérimentations exposées dans la section 5.

3.2 Autres modèles PLNE

Comme décrit dans la section 2.1, une agrégation linéaire d'utilités individuelles $max\{sum(x) : x \in Q\}$ n'assure pas l'équité. Cela suggère de recourir à des opérateurs d'agrégation non linéaires, en particulier le `maxMin` et le `minDev`. L'agrégation `maxMin` $max\{min(x) : x \in Q\}$ s'attaque à l'équité en améliorant la pire des utilités. Cette fonction peut être linéarisée en maximisant une variable $z \geq 0$, i.e. une borne inférieure du vecteur d'utilité $v \,.\, x$ (voir la figure 2a, inégalité C3), où v est le critère à optimiser (e.g. la fréquence). Le modèle PLNE est donnée dans la figure 2a.

Une autre façon d'assurer l'équité consiste à minimiser l'écart maximal entre la meilleure et la plus mauvaise utilité : `Min` $\{\texttt{max}(x) - \texttt{min}(x) : x \in Q\}$. Le modèle `minDev` peut être linéarisé en introduisant $2 \times n$ contraintes et deux variables de décision $z_{max} \geq 0$ et $z_{min} \geq 0$ pour maintenir les valeurs `max` et `min` du vecteur d'utilités $v \,.\, x$ (voir la figure 2b, inégalités C4-C5). Le modèle PLNE résultant est donné par la figure 2b.

4 Travaux connexes

Approches heuristiques. Plusieurs méthodes ont exploré l'idée de séparer la classification de la recherche des descriptions conceptuelles. Pensa *et al.* [18] commencent par extraire les motifs fermés, puis effectuent le clustering k-Means sur ces motifs. Perkowitz et Etzioni [19] inversent les deux phases : la phase *cluster-mining* utilise d'abord une technique de clustering pour former des clusters. À partir des clusters résultants, les descriptions sont apprises par une technique d'apprentissage des règles. Toutes ces techniques sont de nature heuristique et sont fortement influencées par les conditions d'initialisation, nécessitant plusieurs redémarrages, ce qui augmente les coûts de calcul.

Approches déclaratives. Récemment, [16, 17] ont développé des cadres déclaratifs avec la PLNE, qui trouvent des clusterings conceptuels optimaux, où les clusters correspondent à des concepts. Plus tard, Chabert *et al.* ont introduit deux nouveaux modèles PPC pour le clustering conceptuel optimal. Le premier modèle (noté `FullCP2`) peut être vu comme une amélioration de [6]. Le second modèle (noté `HybridCP`) suit l'approche en deux étapes de [16] : la première étape est exactement la même ; la deuxième étape utilise la PPC pour sélectionner les concepts formels. Notre travail est différent dans le sens qu'il garantit que la clustering conceptuel trouvé est optimal et *equitable*.

Clustering basé sur la distance vise à trouver des clusters homogènes uniquement sur la base d'une mesure de dissimilarité entre objets. Différents cadres déclaratifs ont été développés, qui s'appuient sur la PPC [6] ou la PLNE [1, 14]. Il existe quelques approches pour obtenir des clusters équilibrés. La plus importante est l'approche proposée par [2]. Elle comporte trois étapes : (1) échantillonnage ; (2) partitionnement de l'ensemble échantillonné et (3) satisfaction des contraintes d'équilibre imposées sur les clusters. Notre adoption des motifs fermés réduit la redondance

$$\texttt{Max} \sum_{c=1}^{|\mathcal{C}|} \omega_c \,.\, r_c$$

$$\text{s.t.} \begin{cases} \text{Clustering.} \begin{cases} \text{(C1)} \quad \sum_{c=1}^{|\mathcal{C}|} a_{t,c} \,.\, x_c = 1, & \forall t \in \mathcal{D} \\ \text{(C2)} \quad k_{min} \leq \sum_{c=1}^{|\mathcal{C}|} x_c \leq k_{max} \end{cases} \\ \text{Tri } \texttt{OWA}. \begin{cases} \text{(O1)} \quad r_c - (v_i \,.\, x_i) \leq M z_{c,i}, & \forall\, i, c = 1, ..., |\mathcal{C}| \\ \text{(O2)} \quad \sum_{i=1}^{|\mathcal{C}|} z_{c,i} \leq c - 1, & \forall\, c = 1, ..., |\mathcal{C}| \end{cases} \\ x_c \in \{0,1\},\ r_c \in \mathbb{R}_+, \quad \forall\, c = 1, ..., |\mathcal{C}| \\ z_{c,i} \in \{0,1\}, \quad \forall\, i, c = 1, ..., |\mathcal{C}| \end{cases}$$

(a) Modèle `OWA` de base.

$$\texttt{Max} \sum_{c=1}^{|\mathcal{C}|} \omega_c \,.\, (v_c^{\uparrow} \,.\, x_c^{\uparrow})$$

$$\text{s.t.} \begin{cases} \text{(C1)}, \quad \text{(C2)} \\ x_c \in \{0,1\}, \\ \quad \forall\, c = 1, ..., |\mathcal{C}| \end{cases}$$

(b) Modèle `OWA` amélioré.

FIGURE 1 – Modèles PLNE pour le clustering conceptuel équitable basé sur l'opérateur `OWA`.

$$\texttt{Max}\ z$$

$$\text{s.t.} \begin{cases} \text{(C1)}, \quad \text{(C2)} \\ \textbf{(C3)} \quad z \leq v_c \,.\, x_c, & \forall\, c = 1, ..., |\mathcal{C}| \\ x_c \in \{0,1\}, & \forall\, c = 1, ..., |\mathcal{C}| \\ z \geq 0 \end{cases}$$

(a) Modèle PLNE `maxMin`.

$$\texttt{Max}\ z_{max} - z_{min}$$

$$\text{s.t.} \begin{cases} \text{(C1)}, \quad \text{(C2)} \\ \textbf{(C4)} \quad z_{max} \geq v_c \,.\, x_c, & \forall\, c = 1, ..., |\mathcal{C}| \\ \textbf{(C5)} \quad z_{min} \leq v_c \,.\, x_c, & \forall\, c = 1, ..., |\mathcal{C}| \\ x_c \in \{0,1\}, & \forall\, c = 1, ..., |\mathcal{C}| \\ z_{max} \geq 0,\ z_{min} \geq 0 \end{cases}$$

(b) Modèle PLNE `minDev`.

FIGURE 2 – Modèles PLNE pour le clustering conceptuel.

par rapport à d'autres façons de sélectionner les clusters candidats. De plus, `OWA` donne des garanties plus fortes sur les clusterings obtenus en termes d'équité.

5 Expérimentations

L'évaluation expérimentale a pour objectif de répondre aux questions suivantes : (1) passage à l'échelle des modèles PLNE sur les jeux de données considérés ; (2) qualité des clusters résultants et leur description ; (3) comparaison (en temps CPU) avec les modèles PPC introduits par Chabert *et al.* [4].

Protocole expérimental. Toutes les expérimentations ont été menées au centre de calcul Cerist [1], où chaque nœud a deux CPUs Xeon E5-2650 avec 16 cores à 2.00GHz et 64 Go de RAM. Nous avons utilisé `LCM` pour extraire tous les motifs fermés et `CPLEX` v.12.6.1 pour résoudre les différents modèles PLNE. Un *Timeout* de 24 heures a été fixé.

Jeux de données. Nous avons utilisé des jeux de données connus provenant du dépôt de l'UCI. Nous avons également considéré les mêmes instances (appelées ERP-i, avec $i \in [1, 7]$) utilisées dans [4] et provenant d'un cas d'application réel [2], qui vise à extraire des concepts de configuration à partir d'un progiciel de gestion intégré (ERP). Le tableau 2 décrit les caractéristiques de tous les jeux de données.

Pour évaluer la qualité d'un clustering, nous avons évalué la cohérence des transactions couvertes par le clustering, qui est mesurée par la similarité intra-cluster *(ICS)* et la dissimilarité inter-clusters *(ICD)*, les deux devraient être aussi grandes que possible. La mesure de similarité entre deux transactions t et t' est définie par : $s : \mathcal{D} \times \mathcal{D} \mapsto [0,1]$, $s(t,t') = \frac{|t \cap t'|}{|t \cup t'|}$, $ICS(P_1, ..., P_k) = \frac{1}{2} \sum_{1 \leq i \leq k} (\sum_{t,t' \in P_i} s(t,t'))$ et $ICD(P_1, ..., P_k) = \sum_{1 \leq i < j \leq k} (\sum_{t \in P_i, t' \in P_j} (1 - s(t,t')))$

Pour évaluer l'équilibre des clusters par rapport à la fréquence, nous avons utilisé trois mesures : (1) Le ratio entre la fréquence du plus petit concept et la fréquence moyenne (i.e. Min/Avg). Pour les m transactions couvertes par les k clusters, Avg est calculée par (m/k) ; (2) L'écart-type sur la fréquence (i.e. $StdDev$) ; (3) La déviation entre la plus petite et la plus grande description des concepts sélectionnés (i.e. $devSize$). Nous notons que ces trois mesures ne considèrent pas l'efficacité des solutions.

(a) Analyse qualitative des clusterings. La Fig. 3a compare qualitativement, avec la mesure

1. `http://www.rx-racim.cerist.dz/?page_id=26`.
2. Ces jeux de données sont disponibles sur `http://liris.cnrs.fr/csolnon/ERP.html`.

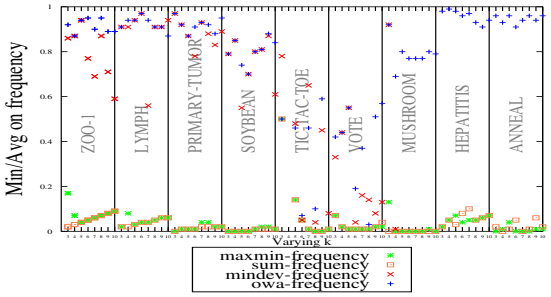
Jeu de données	$\#\mathcal{D}$	$\#\mathcal{I}$	Densité(%)	$\#\mathcal{C}$
Soybean	630	50	32	31,759
Primary-tumor	336	31	48	87,230
Lymph	148	68	40	154,220
Vote	435	48	33	227,031
tic-tac-toe	958	27	33	42,711
Mushroom	8124	119	18	221,524
Zoo-1	101	36	44	4,567
Hepatitis	137	68	50	3,788,341
Anneal	812	93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

(a) Jeux de données UCI.

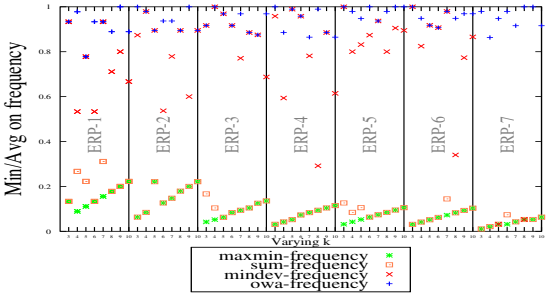
Dataset	$\#\mathcal{D}$	$\#\mathcal{I}$	Density(%)	$\#\mathcal{C}$
ERP-1	50	27	48	1,580
ERP-2	47	47	58	8,1337
ERP-3	75	36	51	10,835
ERP-4	84	42	45	14,305
ERP-5	94	53	51	63,633
ERP-6	95	61	48	71,918
ERP-7	160	66	45	728,537

(b) Jeux de données ERP.

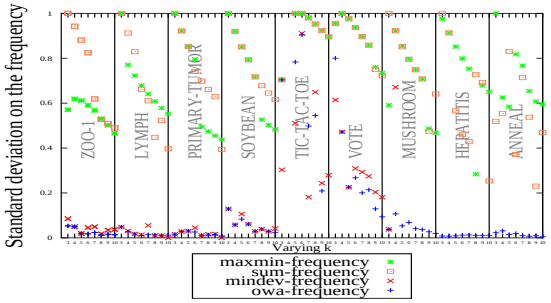
TABLE 2 – Description des jeux de données. Chaque ligne donne le nombre de transactions ($\#\mathcal{D}$), le nombre d'items ($\#\mathcal{I}$), la densité et le nombre de motifs fermés extraits ($\#\mathcal{C}$).



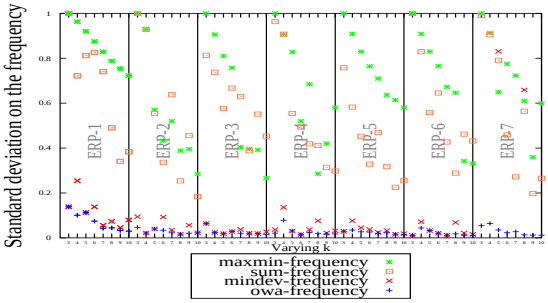
(a) Évaluation (Min/Avg) sur les jeux de données UCI.



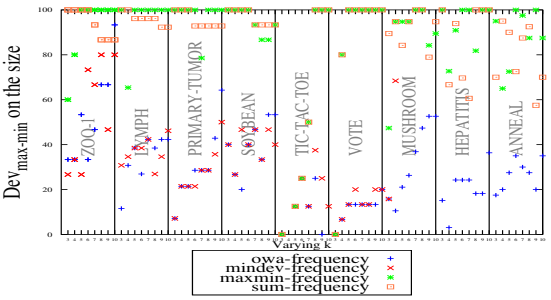
(b) Évaluation (Min/Avg) sur les instances ERP.



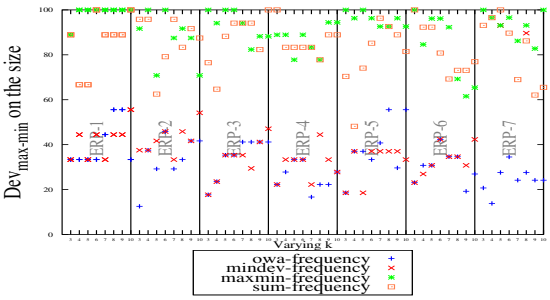
(c) Évaluation *StdDev* sur les jeux de données UCI.



(d) Évaluation *StdDev* sur les jeux de données ERP.



(e) Évaluation *devSize* sur les jeux de données UCI.



(f) Évaluation *devSize* sur les jeux de données ERP.

FIGURE 3 – Qualité de l'équilibre des clusters résultants des différents modèles PLNE.

$\mathcal{D}$	k	OWA		minDev		maxMin		maxSum	
		ICS	ICD	ICS	ICD	ICS	ICD	ICS	ICD
Soybean	3	0.447	**0.784**	0.447	**0.784**	**1.000**	0.026	**1.000**	0.026
	4	0.331	**0.865**	0.331	**0.865**	**1.000**	0.026	**1.000**	0.026
	5	0.259	0.895	0.284	**0.905**	**1.000**	0.026	**1.000**	0.026
	6	0.231	**0.940**	0.231	**0.940**	**1.000**	0.026	**1.000**	0.026
	7	0.195	**0.964**	0.195	**0.964**	**0.959**	0.108	**0.959**	0.108
	8	0.186	**0.987**	0.186	**0.987**	0.671	0.474	**0.959**	0.108
	9	0.166	**1.000**	0.166	**1.000**	0.671	0.474	**0.959**	0.108
	10	0.136	**0.999**	0.142	**0.999**	0.670	0.474	**0.959**	0.108

(a) Maximisation de la fréquence.

$\mathcal{D}$	k	OWA		minDev		maxMin		maxSum	
		ICS	ICD	ICS	ICD	ICS	ICD	ICS	ICD
soybean	3	0.447	**0.776**	0.447	**0.776**	**1.000**	0.026	0.447	0.776
	4	0.334	0.839	0.338	**0.854**	**1.000**	0.026	0.406	0.831
	5	0.296	**0.900**	0.301	**0.900**	**1.000**	0.026	0.389	0.843
	6	0.257	0.929	0.265	**0.934**	**1.000**	0.026	0.398	0.851
	7	0.240	**0.956**	0.240	**0.956**	**0.959**	0.106	0.330	0.909
	8	0.220	0.971	0.198	**0.978**	**0.959**	0.106	0.323	0.918
	9	0.183	**0.991**	0.184	0.989	0.959	0.106	0.216	**0.975**
	10	0.170	0.999	0.157	**1.000**	0.959	0.106	0.213	**0.980**

(b) Minimisation de la diversité.

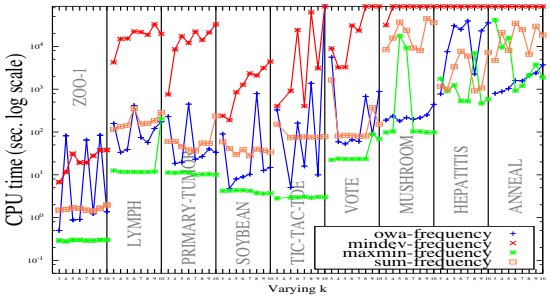
TABLE 3 – Comparaison de la qualité des clusterings résultants en termes de l'ICS et de l'ICD.

Min/Avg, les clusterings résultants des différents modèles PLNE pour différentes valeurs de k sur les jeux de données de l'UCI. `maxMin` et `maxSum` produisent des clusterings avec un déséquilibre fort par rapport à `OWA` et `minDev` (`maxMin` et `maxSum` atteignent toujours des valeurs Min/Avg plus petites). Il est intéressant de noter que `OWA` et `minDev` ont presque la même performance sur les jeux de données avec un nombre de motifs fermés compris entre 10^3 et 10^5. Cependant, pour les trois jeux de données les plus difficiles − Mushroom, Hepatitis et Anneal − la disparité entre les deux modèles PLNE devient plus prononcée : `OWA` obtient toujours des clusterings les plus équilibrés (valeurs Min/Avg proches de 1). Sur ces jeux de données, `minDev` ne parvient pas à trouver une solution même pour des petites valeurs de k. Le même comportement est observé sur les jeux de données ERP (voir Fig. 3b). Sur ERP-7, `minDev` n'a pas été en mesure de trouver une solution. Cela s'explique en partie par le nombre de motifs fermés (10^6), qui génère un nombre important de contraintes par rapport à d'autres jeux ERP (de 10^3 à 10^5). En considérant la mesure *stdDev* (voir les figures 3c et 3d), `OWA` et `minDev` atteignent des valeurs plus petites de *StdDev* sur tous les jeux de données, mais `OWA` est légèrement mieux que `minDev`. Quand on examine la taille des descriptions (voir les figures 3e et 3f), nous pouvons voir que `maxMin` et `maxSum` conduisent à des valeurs *devSize* plus grandes. Ceci est indicatif d'un (ou de quelques) motif(s) ayant une grande fréquence et une petite taille, ou inversement, de motifs ayant une grande taille et une petite fréquence. Ces résultats sont conformes à nos conclusions précédentes. Cependant, pour `minDev` et `OWA`, les solutions optimales trouvées par les deux modèles PLNE tendent à offrir de meilleurs compromis entre les deux critères. Enfin, la Tab. 3 compare [3] les quatre modèles selon l'*ICS* et l'*ICD*. Nous pouvons voir que `minDev` et `OWA` minimisent l'*ICS* pour obtenir des valeurs plus élevées de l'*ICD*. Ce comportement traduit des clusters plus équilibrés : l'*ICS* est nécessairement limitée par le nombre de transactions dans un cluster mais l'*ICD* augmente s'il y a plus de transactions dans d'autres clusters à comparer. `maxMin` et `maxSum` montrent un comportement opposé, produisant un (ou quelques) grands clusters dominants, et de nombreux clusters plus petits.

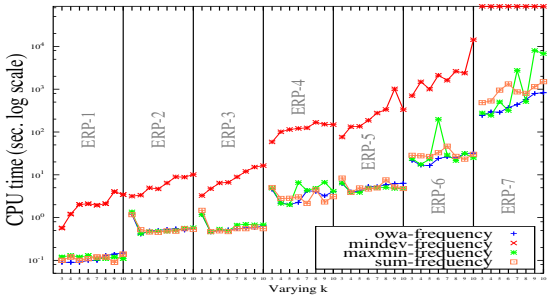
(b) Passage à l'échelle. Les figures 4a et 4b comparent les temps CPU pour calculer les clusterings optimaux pour différentes valeurs de k et sur les jeux de données de l'UCI et de l'ERP. Les clusterings maximisent la somme des fréquences des motifs sélectionnés. Les temps CPU comprennent le temps passé par `LCM` pour extraire tous les motifs fermés. Sur les jeux de données de l'UCI, les performances de `minDev` sont très inférieures par rapport aux autres modèles PLNE. Bien que les résultats qualitatifs de `minDev` soient satisfaisants, ce modèle reste entravé par de longues durées de résolution : il dépasse le *Timeout* sur 32 instances (parmi 72), notamment sur les trois jeux de données les plus difficiles − Mushroom, Hepatitis and Anneal − (voire la Fig. 4a). Cela provient probablement du fait que des contraintes supplémentaires ($2 \times n$) sont utilisées pour capturer la déviation minimale. Cependant, `OWA` donne des résultats assez compétitifs, tout en obtenant des clusterings équitables et optimaux (voir l'analyse qualitative). Il est capable de résoudre tous les jeux et vient en deuxième position. Globalement, `maxMin` obtient les meilleures performances. Cependant, comme noté ci-dessus, les clustering optimaux trouvés sont loin d'être équilibrés ; ils correspondent à des solutions extrêmes (les pires cas). Ceci pourrait être expliqué en partie par l'approche locale de `maxMin` qui est moins contraignante en obtenant ainsi de bonnes performances. Le même comportement est observé pour `minDev` sur les jeux de données ERP. Enfin, les trois modèles PLNE − `OWA`, `maxMin` et `maxSum` − ont des performances très similaires sur les jeux de données de l'ERP. Nous concluons que le modèle `OWA` offre un bon compromis entre la qualité de la solution et le temps de calcul.

(c) Modèles PLNE vs. Modèles PPC. Les figures 5a et 5b comparent les performances de `maxMin`

3. Voir `https://loudni.users.greyc.fr/CClustering.html` pour d'autres résultats.



(a) Instances de l'UCI : maximisation de la fréquence.



(b) Instances de l'ERP : maximisation de la fréquence.

Instance	OWA avec k non fixé $k \in [3, 10]$		OWA avec k fixé		OWA avec k non fixé $k \in [3, \lvert\mathcal{D}\rvert - 1]$	
	meilleur k	Temps (s.) (2)	meilleur k	Temps (s.) (2)	meilleur k	Temps (s.) (2)
Soybean	10	27.09	10	**14.82**	501	15.76
Primary-tumor	10	26.81	10	33.34	215	**14.52**
Lymph	10	77.97	10	173.00	147	**20.61**
Vote	10	89.8	10	879.22	342	**42.3**
tic-tac-toe	9	2,104.07	9	**9.95**	956	11.07
Mushroom	10	**377.21**	10	442.34	8,123	982.95
Zoo-1	10	5.47	10	1.37	59	**0.8**
Hepatitis	10	8,462.45	10	35,498.2	136	**607.51**
Anneal	10	3,674.89	10	3,666.82	459	**1,453.04**

(c) Maximisation de la fréquence.

Instance	OWA avec k non fixé $k \in [3, 10]$		OWA avec k fixé		OWA avec k non fixé $k \in [3, \lvert\mathcal{D}\rvert - 1]$	
	meilleur k	Temps (s.) (2)	meilleur k	Temps (s.) (2)	meilleur k	Temps (s.) (2)
Soybean	10	13.7	10	165.42	501	**9.61**
Primary-tumor	10	46.19	10	210.01	215	**18.5**
Lymph	10	123.84	10	569.63	145	**22.05**
Vote	10	146.72	10	786.84	342	**45.7**
tic-tac-toe	9	37,882.31	9	293.82	956	**7.21**
Mushroom	10	**274.62**	10	667.99	8,123	1,086.13
Zoo-1	10	0.89	10	1.82	59	**0.8**
Hepatitis	10	37,915.3	8	6,275.23	136	**630.91**
Anneal	10	6,839.68	10	25,760.25	459	**2,311.01**

(d) Minimisation de la diversité.

FIGURE 4 – Analyse des temps CPU.

par la PLNE avec les deux modèles PPC (`FullCP2` et `HybridCP`) qui maximisent la fréquence minimale d'un cluster sur les jeux de données de l'UCI et de l'ERP. Les temps CPU d'`HybridCP` incluent ceux de l'étape de prétraitement. Le modèle PLNE `maxMin` est plus performant que `FullCP2` et `HybridCP` sur tous les jeux de données. Aucun des deux modèles PPC ne passe à l'échelle : ils ne parviennent pas à trouver une solution dans la limite de temps pour ($k \geq 4$), sauf pour 4 jeux de données. De plus, le modèle PLNE bat clairement les deux modèles PPC. Enfin, notez que `FullCP2` est légèrement mieux que `HybridCP`.

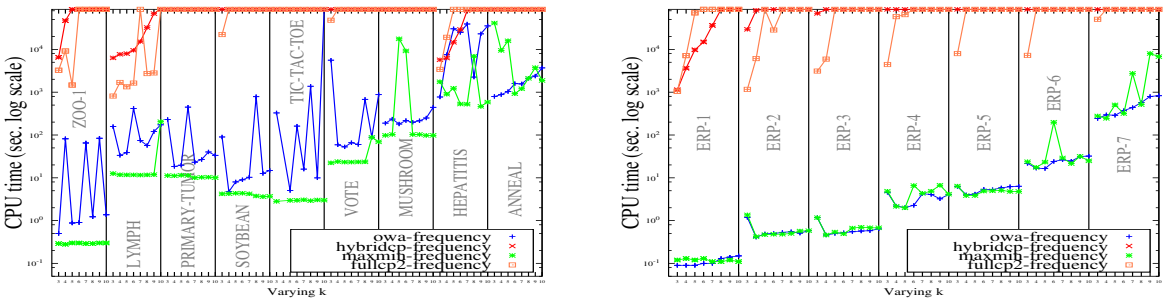
(d) Modèle `OWA` avec k non fixé. Nous évaluons dans cette expérimentation la capacité du modèle `OWA` à trouver la solution optimale lorsque k n'est pas fixé. Nous avons sélectionné deux paramètres : $k \in [3, 10]$ (`OWA`-1) et $k \in [3, |\mathcal{D}| - 1]$ (`OWA`-2). Les Figs. 4c et 4d comparent les temps CPU quand k n'est pas fixé (Colonnes 3 et 7), et quand k est fixé (Col. 5) sur les jeux de données de l'UCI. La Col. 4 rapporte les meilleures valeurs trouvées pour k ($3 \leq k \leq 10$) qui optimisent les deux mesures. Pour tous les jeux de données sauf deux, `OWA`-1 et `OWA`-2 sont les approches les plus performantes. `OWA`-1 est capable de résoudre des instances 5 (resp. 7) plus rapidement en maximisant la fréquence (ou la diversité). Notons que `OWA`-1 et `OWA` (avec k fixé) sont similaires sur la meilleure valeur de k. En se comparant avec `OWA`-1, `OWA`-2 passe bien à l'échelle, en particulier sur les deux jeux de données les plus difficiles – Anneal et Hepatitis – (*speed-up* jusqu'à 60, 09). En effet, des valeurs plus élevées de k permettent de trouver un clustering plus rapidement qu'avec des valeurs plus petites de k : il y a ($|\mathcal{D}| - 1$) clusters pour 3 jeux de données, alors que pour le reste des jeux de données k est plutôt élevé.

6 Conclusion

Nous avons proposé une approche efficace pour la clustering conceptuel équitable. Cette approche utilise l'extraction de motifs fermés pour découvrir des candidats pour les descriptions. L'approche PLNE implémente une fonction d'agrégation équitable basée sur `OWA` pour sélectionner les meilleurs clusters à fréquences équilibrées. Contrairement aux opérateurs `maxMin` et `minDev`, notre approche offre un bon compromis entre la qualité de la solution et le temps de calcul. Nous prévoyons d'étendre notre approche au cas multicritère, où les utilités ne sont pas comparables.

Références

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



(a) Instances de l'UCI : maximisation de la fréquence. (b) Instances de l'ERP : maximisation de la fréquence.

FIGURE 5 – Comparaison des temps CPU du modèle PLNE de `maxMin` avec les deux modèles PPC.

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

[3] S. Bouveret and M. Lemaître. Computing leximin-optimal solutions in constraint networks. *Artif. Intell.*, 173(2) :343–364, 2009.

[4] M. Chabert and C. Solnon. Constraint programming for multi-criteria conceptual clustering. In *CP 2017*, volume 10416 of *LNCS*, pages 460–476. Springer, 2017.

[5] K. M. Chong. An induction theorem for rearrangements. *CJM*, 28 :154–160, 1976.

[6] T-B-H Dao, K-C Duong, and C. Vrain. Constrained clustering by constraint programming. *Artif. Intell.*, 244 :70–94, 2017.

[7] D. Dubois and P. Fortemps. Computing improved optimal solutions to max-min flexible constraint satisfaction problems. *EJOR*, 118 :95–126, 1999.

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

[9] B. Golden and P. Perny. Infinite order lorenz dominance for fair multiagent optimization. In *AAMAS*, pages 383–390, 2010.

[10] Michael M. Kostreva, Wlodzimierz Ogryczak, and Adam Wierzbicki. Equitable aggregations and multiple criteria analysis. *EJOR*, 158(2) :362–377, 2004.

[11] W. Marshall and I. Olkin. *Inequalities : Theory of Majorization and its Applications*. Academic Press, London, 1979.

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

[13] Hervi Moulin. *Axioms of Cooperative Decision Making*. Number 9780521360555 in Cambridge Books. Cambridge University Press, 1989.

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

[15] Włodzimierz Ogryczak and Tomasz Śliwiński. On solving linear programs with the ordered weighted averaging objective. *EJOR*, 148(1) :80 – 91, 2003.

[16] A. Ouali, S. Loudni, Y. Lebbah, P. Boizumault, A. Zimmermann, and L. Loukil. Efficiently finding conceptual clustering models with integer linear programming. In *IJCAI 2016*, pages 647–654, 2016.

[17] A. Ouali, A. Zimmermann, S. Loudni, Y. Lebbah, B. Crémilleux, P. Boizumault, and L. Loukil. Integer linear programming for pattern set mining ; with an application to tiling. In *PAKDD 2017*, pages 286–299, 2017.

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

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

[20] A.K. Sen and J.E. Foster. *On economic inequality*. Clarendon Press, Oxford, 1997.

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

[22] Ronald R. Yager. On ordered weighted averaging aggregation operators in multicriteria decisionmaking. *IEEE Trans. Systems, Man, and Cybernetics*, 18(1) :183–190, 1988.

[23] Y. Yang and B. Padmanabhan. Segmenting customer transactions using a pattern-based clustering approach. In *ICDM*, pages 411–418, 2003.

Une version distribuée du solveur Syrup

Gilles Audemard Jean-Marie Lagniez Nicolas Szczepanski Sébastien Tabary
Université D'Artois. CRIL/CNRS UMR 8188, Lens

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

Résumé

Un solveur SAT parallèle « portfolio » doit partager des clauses afin d'être efficace. Dans un environnement à mémoire distribué, un tel partage engendre des coûts de communication élevés. Dans ce papier, nous proposons une nouvelle version du solveur de l'état de l'art Syrup maintenant capable d'être exécuté sur une architecture à mémoire distribuée. Nous analysons et comparons différents modèles de programmation des communications. Nous montrons qu'il est possible de partager beaucoup de clauses sans pénaliser les solveurs. Les expérimentations utilisant jusqu'à 256 cœurs de calcul sur les instances de la compétition SAT 2016 montrent que notre solveur est plus efficace que les autres approches. Cela ouvre de nombreuses perceptives afin d'améliorer les solveurs parallèles ayant besoin de partager beaucoup de données.

1 Introduction

La programmation d'un solveur SAT parallèle partageant des données sur le réseau n'est pas si simple. Notamment, ces données sont principalement des clauses apprises par les solveurs et doivent donc être échangées à n'importe quel moment durant la recherche. Qui plus est, il existe différentes manières de programmer les communications, quelles soient réalisées entre plusieurs machines ou entre plusieurs *threads* sur une même machine. Plusieurs techniques de programmations parallèles sont alors regroupées en ce qui est communément appelées les modèles de programmations parallèles. Pour réaliser ces contributions, nous utilisons comme base le solveur *multi-thread* Syrup. Ces travaux ont donnés lieu à la publication internationale [2].

Dans ce papier, nous réalisons d'abord une étude expérimentale du solveur *multi-thread* Syrup afin d'étudier son extensibilité. Ensuite, nous exposons ce qu'implique l'utilisation d'architectures distribuées dans le cadre de SAT et présentons nos objectifs. Puis, nous évaluons deux modèles de programmations distribuées existants et déjà utilisés dans des solveurs SAT. Le premier est nommé le modèle de programmation pur par passage de message et est utilisé par AmPharoS et TopoSAT. Le deuxième est appelé le modèle partiellement hybride et est utilisé par HordeSAT. Nous montrons que ces schémas ont un impact direct sur les performances des solveurs. De ce fait, nous introduisons un modèle complètement hybride à partir du solveur Syrup qui n'a, à notre connaissance, jamais été appliqué à SAT.

2 Étude expérimentale de Syrup

Un point clé dans l'efficacité de Syrup provient de la manière de sélectionner les clauses à partager ainsi que l'utilisation d'une structure de donnée dédiée à la gestion de ce partage. Ces deux caractéristiques se révèlent nécessaires afin d'éviter une surcharge des unités de calcul. Même si Syrup est extensible jusqu'à 32 cœurs de calcul, il n'est pas initialement conçu pour fonctionner avec un nombre très élevé de cœurs.

#cœurs	sat	unsat	Total
1	15	11	26
2	25	15	40
4	29	23	52
8	31	25	56
16	36	27	63
32	42	28	70

Table 1 – Résultats de Syrup sur 1 à 32 cœurs de calculs.

Afin d'évaluer les performances et l'extensibilité du solveur *multi-thread* Syrup, nous l'exécutons sur la totalité des 100 instances du *parallel track* de la SAT Race 2015 en utilisant 1, 2, 4, 8, 16 et 32 cœurs. Dans ces expérimentations, nous fixons le temps réel à 1800 secondes pour toutes les instances. Un ordinateur de

32 cœurs avec 256GB de mémoire RAM a été utilisé, c'est un *quad-processor Intel XEON X7550*.

Comme le montre la figure 1 et la table 1, plus nous augmentons le nombre de cœurs, plus nous avons d'instances résolues et celles-ci sont résolues de plus en plus rapidement. La version séquentielle de Syrup, Glucose, résout 26 instances (15 satisfaisables et 11 insatisfaisables). Avec 8 cœurs, Syrup résout 56 instances (31 satisfaisables et 25 insatisfaisables) et avec 32 cœurs, il résout 70 instances (42 satisfaisables et 28 insatisfaisables). Le nombre d'instances insatisfaisables résolues augmente moins rapidement que le nombre d'instances satisfaisables car il y a beaucoup moins d'instances insatisfaisables (33 *Versus* 53). La table 1 récapitule le nombres d'instances résolues suivant le nombre de cœurs utilisés.

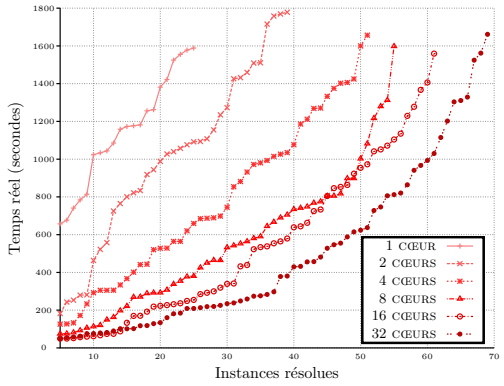


Figure 1 – Résultats du solveur parallèle Syrup sur 1 à 32 cœurs de calculs sur les 100 instances du *parallel track* de la SAT Race 2015.

La prochaine étape est alors d'utiliser plus de cœurs de calcul tout en essayant d'avoir une bonne extensibilité. Afin de dépasser cette limite, une autre manière d'obtenir plus de cœurs consiste à considérer plusieurs ordinateurs comme ceux disponibles via les *clusters* de calcul ou plus récemment via le *cloud computing*. Ce papier expose alors une version de Syrup distribuée appelée D-Syrup.

3 L'architecture distribuée

Deux méthodes peuvent être employées en considérant le partage d'informations dans une architecture distribuée : celle centralisée et celle décentralisée. La première manière dite centralisée échange les informations en les envoyant à un processus maître qui les renvoi aux autres unités de calcul. En revanche, la seconde dite décentralisée envoie directement les informations aux autres unités.

Dans AmPharoS [1], nous avons expérimentalement démontré qu'une manière centralisée en charge de collecter et partager les informations entraîne une congestion rapide du réseau. Plus précisément, l'échange des clauses apprises n'apporte alors plus aucun gain sur la plupart des problèmes. Par conséquent, partager d'une manière centralisée les clauses n'est potentiellement pas la bonne solution car l'objectif d'une architecture distribuée dans un solveur SAT parallèle est d'utiliser un nombre considérable d'ordinateurs.

Afin de pallier le problème du goulot d'étranglement induit par l'architecture centralisée, nous utilisons une architecture décentralisée. En d'autres mots, dans celle-ci et contrairement à l'architecture centralisée, chacun des processus communique directement les informations avec tous les autres, sans passer par un processus maître. Cependant, notons qu'un bouchon dans les communications peut encore survenir quand le nombre de clauses à partager est trop élevé. De plus, les communications réalisées doivent être prudemment réalisées afin d'éviter les interblocages. Notons bien ici que nous parlons des *deadlocks* engendrés par l'échange de messages via le réseau, pas ceux induit par la mémoire partagée et leurs sections critiques associées (*mutex*). Un *deadlock* apparaît quand un processus demande une ressource qui est déjà occupée par d'autres processus. Par exemple, considérons le cas ou chaque processus est implémenté tel que qu'il réalise respectivement une opération `send` bloquante puis une opération `receive` elle aussi bloquante. Le terme « bloquante » signifie alors ici que l'opération `send` (resp. `receive`) est bloquée tant que le message n'a pas été envoyé (resp. reçu) sur le réseau. Dans une telle situation, si deux processus envoient chacun un message à l'autre, alors chaque processus doit attendre que le message soit reçu par l'autre pour continuer. Par conséquent, ils sont tous les deux bloqués dans leurs opérations `send`. Plus précisément, une opération `send` n'est pas complétée tant que l'opération `receive` correspondante n'est pas exécutée.

Ce problème est bien connu et plusieurs solutions existent afin d'éviter et/ou détecter ces *deadlocks* [8, 4]. Au niveau des solveurs SAT parallèles, la plupart des approches proposées dans la littérature [6, 5, 1] utilisent des communications non-bloquantes et/ou collectives pour éviter ce problème. Généralement, la solution retenue consiste à attendre que tous les $\text{BUFFERS}_{network}$ contenant les messages à envoyer peuvent être consultés et modifiés sans risque de *deadlocks*. Cela consiste à attendre que les messages aient été envoyés (les messages n'ont pas été nécessairement reçus). Cette solution est réalisée via ce que nous appelons un cycle de communications. C'est une boucle effectuant plusieurs tâches qui est répétée tant que la solution n'a pas été trouvée :

— Exécution de toutes les communications des

données d'un BUFFER$_{network}$ (`MPI_SEND` et `MPI_RECEIVE`) ;

— Attente des terminaisons des communications (`MPI_WAIT_ALL`) ;

— Attente de 5 secondes. Durant ce temps, un *thread* solveur remplit le BUFFER$_{network}$ de clauses.

Un autre point important concerne la manière de réaliser les communications entre les solveurs. Deux solutions sont généralement considérées :

— Un *thread* alterne périodiquement étapes de recherches et de communications (AmPharoS) ;

— Deux *threads* distincts réalisent séparément ces deux tâches (HordeSAT [5] et TopoSAT [6]).

Nous évaluons empiriquement une version distribuée du solveur *multi-thread* Syrup en utilisant des *threads* chargés de faire les communications.

4 Le modèle de programmation pur par passage de messages

Ce modèle de programmation respecte le protocole de communication présenté dans la section précédente. En plus de cela, il se définit via certaines contraintes :

— Il utilise plusieurs processus par ordinateur ;

— Chaque solveur SAT séquentiel forme un processus : il est capable d'effectuer la recherche aussi bien que les communications grâce à l'aide d'un ou de deux *threads* (solveur et communicateur).

— Les données sont répliquées pour chaque solveur à la fois sur le réseau et dans la mémoire partagée.

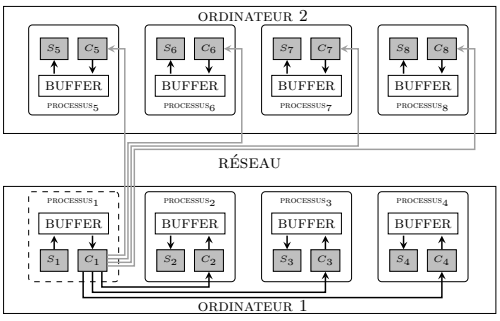


Figure 2 – Le modèle de programmation pur par passage de messages : l'accent est réalisé sur les données (clauses apprises) envoyées par le solveur S_1 aux autres solveurs sur une configuration possédant deux ordinateur.

La figure 2 illustre ce modèle de programmation avec deux *threads* par processus solveur. Le processus solveur PROCESSUS$_1$ représenté par un carré en pointillé, regroupe un *thread* de recherche S_1 (un solveur) et son propre *thread* communicateur C_1. Dans ce papier, les données (les clauses apprises) son partagées entre ces deux *threads* en utilisant la mémoire partagée via un tampon de données nommée BUFFER comme dans la figure 2.

Sur cette figure, nous nous focalisons sur les données envoyées par le *thread* de recherche S_1 à la totalité des autres solveurs représentés par les autres *threads* de recherche S_n via les processus PROCESSUS$_n$ avec $n \neq 1$. Pour cela, les clauses apprises sont d'abord transférées du *thread* S_1 à son *thread* communicateur C_1 via un BUFFER utilisant la mémoire partagée (bibliothèque *multi-thread*). Ensuite, chaque *thread* communicateur C_n tel que $n \neq 1$ reçoit les clauses apprises à partir de C_1 uniquement par passage de messages (via une bibliothèque réseau) même quand deux *threads* sont sur la même machine. Pour finir, chaque *thread* communicateur C_n envoie les clauses apprises reçues à leur *thread* de recherche respectif S_n (associé au PROCESSUS$_n$) tel que $n \neq 1$.

Plus précisément, sur un même ordinateur, cette manière de fonctionner oblige la copie de données entre les solveurs sans utiliser l'espace d'adressage global habituellement induit par la mémoire partagée (flèche noire). Au lieu de cela, seul le passage de messages est utilisé entre deux processus, peu importe si ils sont sur la même machine ou pas. De plus, quand les données doivent être envoyées aux solveurs situés sur une autre machine, ce modèle exige d'envoyer un message à chaque solveur séquentiel (et donc à chaque processus) plutôt qu'un seul par machine (flèche grise). À titre d'exemple, dans la figure 2, lorsque le communicateur C_1 envoie une clause à la deuxième machine, celle-ci est envoyée 4 fois plutôt qu'une seule fois. De ce fait, dans notre exemple, le réseau est 4 fois plus encombré. Par conséquent, la même donnée est répliquée et envoyée sur le réseau autant de fois que le nombre de solveurs inclus dans la machine les réceptionnant, et cela conduit généralement à la congestion du réseau.

Le modèle de programmation pur par passage de messages est la solution retenue par les auteurs de TopoSAT afin de réaliser leurs communications. Ils utilisent des cycles de communications de 5 secondes. Un point clé de l'architecture de TopoSAT est l'utilisation de communications point à point et non bloquantes afin de construire différentes topologies d'échange de messages (*2-dimensional grid*, *medium-coupled*, ...). Ces topologies visent alors à réduire la congestion du réseau induit par ce modèle de programmation. Toutefois, notons que nos objectifs ne sont pas de comparer les topologies. De ce fait, nous implémentons pour chaque modèle présenté, la topologie où chaque processus communique avec tous les autres, autrement dit, celle sans aucune restriction.

Ce modèle est aussi la solution que nous avons em-

ployé dans le solveur « diviser pour mieux régner » AmPharoS. Contrairement à ce qui vient d'être présenté dans cette section, AmPharoS a la particularité de n'employer qu'un seul *thread* par processus. Ce *thread* alors similaire à un processus qui effectue à la fois les phases de recherche et de communication. Cette manière de faire ne retire aucun des défauts de ce modèle de programmation. Nous montrons alors expérimentalement une baisse des performances anormales lors de l'échange des clauses apprises et ces travaux ont alors pour objectif de régler ce problème.

Observations Empiriques

Afin de comparer différents modèles de programmation, nous avons développé un modèle pur par passage de messages basé sur le solveur séquentiel glucose. Cette implémentation fournit directement une version distribuée de Syrup en utilisant les *threads* communicateurs présentés dans cette section.

Premièrement, nous avons étudié l'impact de la taille des buffer transférant les clauses entre un *thread* de recherche S_n et un *thread* communicateur C_n, ainsi que la fréquence des échanges (via les cycles de communications). Afin d'examiner les performances globales du solveur pendant cette étude, nous exposons le nombre de propagations unitaires par seconde. Pour réaliser cette expérimentation, nous exécutons 256 processus sur 32 ordinateurs composés chacun de 8 cœurs durant 100 secondes (temps réel). Nous avons sélectionné 20 instances provenant de la compétition SAT Race 2015 (*parallel track*) et non résolues par le solveur Syrup en 300 secondes. Ces résultats ont été obtenus dans la section 2 avec 32 cœurs de calcul. Il représente un ensemble de familles d'instances variées et difficilement résolvables par Syrup. Par conséquent, ces 20 instances peuvent être vues comme un échantillon représentant diverses instances difficiles. Afin d'évaluer l'impact de la taille du buffer et les différents temps de cycles de communications sur l'efficacité du solveur, trois tailles ont été sélectionnées pour les buffer (20MB, 100MB, et 200MB) et trois différents cycles de communications ont été considérés (tous les 0.5, 5, et 10 secondes).

Le tableau 2 reporte pour chaque combinaison de taille du buffer et de temps de cycle de communications, la moyenne du nombre de clauses reçues noté « Reçues » dans la table, la moyenne du nombre de clauses réellement retenues par les solveurs noté « Retenues » (ce sont les clauses importées à travers le réseau et qui ont traversées la mémoire partagée (le buffer) et sont alors *1-watched* : certaines peuvent être supprimées suivant la taille du buffer), la moyenne du nombre de clauses retenues qui entrent en conflit (ces clauses passent de *1-watched* à *2-watched*) notée « Ret. (conflits) », la moyenne du nombre de clauses retenues utilisées durant le processus de la propagation unitaire notée « Ret. (utiles) », la moyenne du nombre de propagation unitaire par seconde notée « Propagation » et, pour finir, la moyenne du nombre de conflits par seconde notée « Conflits ». Ces mesures sont effectuées sur toutes les instances pendant leur temps réel d'exécution de 100 secondes sur 32 machines pour un total de 256 cœurs de calcul (32 *bi-processors Intel XEON X7550*). Afin de ne pas ralentir le solveur, elles sont récupérées uniquement toutes les 5 secondes pendant la recherche. Pour chaque mesure, une moyenne globale est calculée à partir des données apportées par les 20 instances.

	20MB			**100MB**			**200MB**		
Com. Cycle	**0.5s**	**5s**	**10s**	**0.5s**	**5s**	**10s**	**0.5s**	**5s**	**10s**
Reçues	8,270	30,989	38,820	9,920	28,369	39,890	11,007	28,296	34,346
Retenues	82%	25%	0.8%	93%	51%	34%	93%	71%	62%
Ret. (conflits)	43	55	26	54	51	16	72	33	12
Ret. (utiles)	494	618	306	553	575	173	783	452	192
Propagations	100,379	489,289	525,738	91,103	479,145	540,286	69,912	469,021	443,696
Conflits	216	823	1101	254	734	1069	275	721	891

Table 2 – Modèle de programmation pur par passage de messages. Impact de la taille des buffer et de la fréquence des échanges (cycles de communications) sur les performances globales des solveurs. Tous ces résultats correspondent à la moyenne obtenue par tous les processus par seconde.

Premièrement, nous pouvons observer que quelque soit la taille des buffer, les pires résultats sont obtenus pour les cycles de communication de 0.5 secondes. En effet, les nombres de propagations et de conflits par seconde sont considérablement affectés : 5 fois moins qu'en utilisant des cycles de 5 ou 10 secondes. Cela est causé par les opérations bloquantes exécutées durant un cycle de communications (`MPI_WAIT_ALL` et opérations utilisant les *mutexs*). Car plus les cycles sont courts, plus ses opérations sont réalisées fréquemment. De ce fait, dès que nous augmentons le temps des cycles de communications à 5 secondes, le nombre d'appels à ces opérations bloquantes est divisé par 10. Par conséquent, les performances globales du solveur en terme de propagations et de conflits sont meilleures.

Deuxièmement, nous pouvons aussi remarquer que le nombre de clauses retenues diminue quand le temps d'un cycle de communications augmente. En effet, dans ce cas, plus de clauses doivent être partagées en un cycle. Par conséquent, le buffer de la mémoire partagée a plus de chance d'être plein. Dans ce derniers cas, les clauses ne sont simplement pas partagées et sont alors supprimées du buffer. Cela est un potentiel défaut, car certaines de ces clauses non partagées peuvent être utiles au solveur.

À présent, nous nous focalisons sur un temps des cycles de communication de 5 secondes. Nous obser-

vons que plus nous augmentons la taille des BUFFER, plus le pourcentage de clauses retenues est élevé. Effectivement, la même quantité de données est partagée en un cycle, mais moins de clauses sont supprimées. Cela peut être vu comme un bon point car augmenter la taille du BUFFER permet donc d'apporter plus de clauses aux solveurs S_n. Néanmoins, le nombre de clauses retenues étant utiles et entrant en conflit, aussi bien que le nombre de propagations et de conflits diminuent légèrement quand la taille du BUFFER augmente. Cela est expliquer par une surcharge des solveurs à cause de la gestion de la base de clauses apprises. Plus précisément, un nombre trop élevé de clauses apprises ralentit les solveurs séquentiels car il y a de plus en plus de littéraux à examiner durant la propagation unitaire.

Afin d'obtenir les meilleures performances, nous avons fixé ces deux paramètres :

— Le temps d'un cycle de communications à 5 secondes ;

— Taille du BUFFER de la mémoire partagée à 20MB.

5 Le modèle de programmation partiellement hybride

Dans cette section, nous proposons un modèle de programmation partiellement hybride comme une alternative afin de pallier les inconvénients du modèle de programmation pur par passage de messages. Ce modèle de programmation se définie par certaines caractéristiques :

— Il intègre un seul processus par ordinateur ;

— Plusieurs solveurs SAT séquentiels forment un processus : il s'agit d'un regroupement de plusieurs *threads* de recherche associés à un seul *thread* communicateur ;

— Deux librairies indépendantes sont utilisées pour communiquer : une gère la mémoire partagée tandis qu'une autre gère les communications sur le réseau par passage de messages ;

— Le *thread* communicateur regroupe les données qui doivent être envoyées sur le réseau et distribue celles qu'il reçoit.

Le *thread* communicateur est utilisé comme une interface afin de partager les données entre des solveurs localisés sur des ordinateurs différents tandis que la mémoire partagée permet d'échanger les données entre les solveurs sur le même ordinateur. La figure 3 décrit ce schéma. Chaque ordinateur utilise un seul processus qui regroupe tous les solveurs SAT séquentiels d'une machine et un *thread* communicateur.

Cette figure décrit la transmission de clauses du solveur S_1 aux autres solveurs S_n tel que $n \neq 1$. Pour cela, le solveur S_1 envoie d'abord ses clauses apprises aux *threads* solveurs S_2, S_3, S_4 puis à son *thread* communicateur C_1 en utilisant le tampon localisé dans la mémoire partagée (BUFFER). Remarquons que les BUFFER du modèle de programmation pur par passage de messages et de ce modèle sont les mêmes aux niveaux de leurs implémentations, seul leur utilisation change. Par la suite, le communicateur C_1 envoie les clauses à travers le réseau au communicateur C_2. Pour finir, une fois que le communicateur C_2 a récupéré les clauses, il les distribue aux solveurs S_5, S_6, S_7 et S_8.

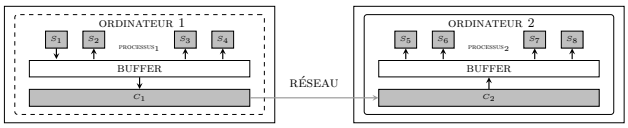


Figure 3 – Le modèle de programmation partiellement hybride : une focalisation est réalisée sur les données (clauses apprises) envoyées par le solveur S_1 aux autres solveurs avec deux ordinateurs.

De toute évidence, le principal avantage de cette approche est la mutualisation des données échangées à la fois sur le réseau et via la mémoire partagée. L'hybridation permet d'utiliser proprement les deux sortes d'échange de données en fonction de la localisation des *threads* (sur la même machine ou sur des machines différentes). Ainsi, ce modèle corrige le défaut le plus important du modèle de programmation pur par passage de messages. À titre d'exemple, une clause doit être dupliquée 4 fois en mémoire puis envoyée 4 fois sur le réseau afin d'atteindre tous les processus d'une autre machine dans la figure 2 représentant une possible configuration du modèle de programmation pur par passage de messages. Au lieu de cela, le modèle partiellement hybride représenté par la figure 3 met en mémoire une seule fois la clause et l'envoie aussi qu'une seule fois sur le réseau pour atteindre une autre machine. Par conséquent, les données échangées à travers le réseau sont considérablement réduites.

Le solveur HORDESAT est basé sur ce modèle de programmation. Sa principale caractéristique est la gestion des messages envoyés via le *thread* communicateur. Il utilise des cycles de communication de 5 secondes. Sa particularité est d'avoir une communication collective des données d'une taille fixe toutes les 5 secondes. Plus précisément, les tampons $\text{BUFFER}_{network}$ de chaque *thread* communicateur possède la même taille fixée à 1500 entiers. Autrement dit, quand HORDESAT est exécuté sur o machines, sur une machine donnée, 1500 entiers de données son envoyés aux autre machines et cette machine reçoit $1500 \times (o-1)$ entiers à chaque cycle de communications.

La taille fixe du BUFFER$_{network}$ d'HordeSAT induit quelques problèmes. Si le nombre de clauses ne dépasse pas les 1500 entiers au moment de l'envoi du message sur le réseau, un rembourrage de 0 est ajouté au BUFFER$_{network}$ afin d'obtenir la taille désirée. À l'inverse, lorsque la limite de taille est atteinte avant l'envoi, les clauses ne sont simplement pas ajoutées au BUFFER$_{network}$.

Afin d'éviter ces problèmes, les auteurs d'HordeSAT [5] proposent d'adapter les heuristiques des solveurs SAT séquentiels afin qu'ils produisent plus ou moins de clauses en fonction de la demande du BUFFER$_{network}$. Ainsi, pour un cycle de communications, si le nombre de clauses n'est pas suffisant afin de remplir totalement le BUFFER$_{network}$, alors une fonction est appelée afin d'encourager heuristiquement les solveurs SAT séquentiels à produire plus de clauses pour le prochain cycle de communications. Inversement, quand le BUFFER$_{network}$ a atteint sa limite et ne peut plus accueillir de clauses, les solveurs SAT séquentiels sont amenés à produire moins de clauses apprises pour le prochain cycle. Notons aussi qu'avant de mettre les clauses dans le BUFFER$_{network}$, celles-ci sont triées d'une manière heuristique (suivant leurs tailles et leurs valeurs LBD [3]) afin de transmettre les « meilleures » en priorité.

Nous avons développé notre propre solveur partiellement hybride, en utilisant les *threads* communicateurs présentés au début de cette section. Ce solveur est basé sur le solveur parallèle *multi-thread* Syrup (version 4.1). Rappelons que dans ce modèle de programmation, nous devons ajuster deux tampons différents : celui utilisé pour partager les clauses entre les différents *threads* de recherche sur une même machine et utilisé par un appel du solveur Syrup, et le tampon nécessaire à l'envoi et la réception des clauses sur le réseau. Il y a un BUFFER$_{network}$ par machine, c'est-à-dire, un seul par *thread* communicateur. Nous proposons d'effectuer une communication collective qui échange des données de tailles différentes. En d'autres mots, nous ne limitons pas la taille des BUFFER$_{network}$. De ce fait, lors d'un cycle, une machine peut alors envoyer directement un très grand nombre de clauses tandis qu'une autre peut en envoyer un très petit nombre. Notre objectif est donc opposé à HordeSAT, puisque nous avons choisi de ne pas modifier la manière dont les clauses sont produites par les solveurs SAT séquentiels. De plus, cela évite d'envoyer inutilement des données composées de zéro sur le réseau et de ne pas écarter quelques clauses qui auraient dû être partagées. En pratique, cela est permis par les bibliothèques MPI ou la programmation par *socket*. Plus précisément, HordeSAT fait appel à la fonction `MPI_AllGATHER` tandis que notre solveur utilise la fonction `MPI_AllGATHERV`.

	20MB			100MB			200MB		
Com. Cycle	**0.5s**	**5s**	**10s**	**0.5s**	**5s**	**10s**	**0.5s**	**5s**	**10s**
Reçues	28,202	38,448	41,826	26,980	36,505	36,370	26,836	34,643	35,427
Retenues	84%	68%	43%	86%	85%	81%	86%	86%	86%
Ret. (conflits)	342	318	209	325	280	190	326	258	176
Ret. (utiles)	3,841	3,256	2,578	3,429	2,688	2,008	3,647	2,603	1,853
Propagations	504,540	636,850	658,177	528,529	577,653	603,685	511,709	559,477	595,084
Conflits	976	1364	1283	928	1253	1248	929	1184	1203

Table 3 – Modèle de programmation partiellement hybride. Impact de la taille des BUFFER et de la fréquence des échanges (cycles de communications) sur les performances globales des solveurs. Tous ces résultats correspondent à la moyenne obtenue par tous les processus par seconde.

Observations Empiriques

Comme pour la section précédente 4 (le modèle de programmation pur par passage de messages), nous étudions également l'impact de la taille du BUFFER et la fréquence des échanges (temps des cycles de communication) sur les performances globales des solveurs pour ce modèle avec les mêmes instances. Les résultats sont présentés dans le tableau 3. Ils montrent la même tendance que pour le modèle de programmation pur par passage de messages (tableau 2). Un point important est que le modèle partiellement hybride fournit des taux de propagations et de conflits par seconde plus élevés que le modèle pur par passage de messages. De plus, comme pour le modèle de programmation pur par passage de messages, un BUFFER fixé à 20MB et des cycles de communication de 5 secondes semblent être le meilleur compromis.

6 Le modèle de programmation complètement hybride

Comme peut le suggérer les résultats du modèle de programmation partiellement hybride (tableau 3), il semble bénéfique de partager les clauses à travers le réseau aussi vite que possible. En effet, nous observons que la quantité de clauses retenues (lignes Ret. (conflit) et Ret. (utiles)) est, quelque soit la taille du BUFFER choisie, plus importante quand la fréquence des cycles de communication augmente, c'est-à-dire, quand le temps des cycles de communication diminue. Néanmoins, quand les cycles de communication sont très court (0,5 secondes), nous pouvons remarquer que le modèle de programmation partiellement hybride souffre d'un problème lié à ses sections critiques (*mutexs* et `MPI_WAIT_ALL`). En effet, ce problème réduit considérablement l'efficacité du solveur. Plus précisément, avec un BUFFER de 20MB, nous avons 504,540 propagations (resp. 976 conflits) par seconde pour des cycles de 0.5 secondes alors que nous avons 636,850

propagations (resp. 1364 conflits) par seconde avec des cycles de communication de 5 secondes. Il est important de remarquer que ces chiffres sont respectivement les bornes minimums et maximums des résultats obtenus quelque soit la taille du BUFFER (lignes propagations et conflits du tableau 3). Par conséquent, cela démontre que les sections critiques ont un rôle très importants dans l'efficacité d'un solveur SAT parallèle. Par section critique, nous parlons des deux moyens habituellement employés afin d'éviter les problèmes de *deadlock*, à savoir, l'utilisation des *mutexs* avec la mémoire partagée et l'utilisation d'opérations bloquantes comme `MPI_WAIT_ALL` avec le passage de messages. Dans cette section, nous abordons alors ce problème en introduisant un algorithme qui possède deux objectifs orthogonaux :

— Réduire le nombre d'accès aux sections critiques ;

— Envoyer les clauses aussi vite que possible.

Le but est donc d'envoyer les clauses directement sur le réseau, c'est-à-dire, sans attendre un cycle de communications. Pour cela, nous modifions les *threads* de recherche afin qu'il envoient eux-mêmes les clauses sur le réseau. Ainsi, les *threads* de recherche font une partie du travail des *threads* communicateurs : l'envoi des clauses sur le réseau. Par conséquent, le *thread* communicateur d'un processus (de chaque ordinateur) devient un *thread* receveur puisque son seul travail est à présent de recevoir les clauses. Contrairement au modèle de programmation partiellement hybride, ce modèle de programmation permet à plusieurs *threads* de communiquer des données sur le réseau en même temps. Nous appelons donc une telle approche le modèle de programmation complètement hybride. Il est donc définit par les contraintes suivantes :

— Il intègre un seul processus par ordinateur ;

— Plusieurs solveurs SAT séquentiels forment un processus : il s'agit d'un regroupement de plusieurs *threads* ayant les fonctionnalités de recherche et d'envoi des clauses sur le réseau associés à un seul *thread* les recevant alors nommé receveur ;

— Deux librairies indépendantes sont utilisées pour communiquer : une gère la mémoire partagée tandis qu'une autre gère les communications réseau par passage de messages ;

— Les *threads* receveurs distribuent les données qu'ils reçoivent à leurs *threads* de recherche.

La figure 4 représente le modèle de programmation complètement hybride. Chaque ordinateur possède un seul processus qui intègre tous les *threads* de recherche et un *thread* receveur. Dans cette figure, le solveur S_1 envoie une clause aux solveurs S_2, S_3 et S_4 via la mémoire partagée (BUFFER) et au deuxième ordinateur

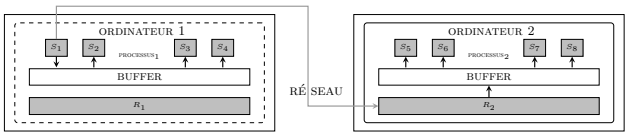


Figure 4 – Modèle de programmation complètement hybride : Une focalisation est réalisée sur les données (clauses apprises) envoyées par le solveur S_1 aux autres solveurs avec deux ordinateurs.

via le réseau. Par la suite, le *thread* R_2 reçoit cette clause et l'envoie au solveurs S_5, S_6, S_7 et S_8 via la mémoire partagée.

Notons bien que dans notre méthode, il n'y a pas de cycle de communications. À la place, les clauses sont envoyées une par une par les *threads* de recherche via une communication bloquante (`MPI_SEND`). Les *threads* receveurs récupèrent constamment les clauses via une boucle exécutant l'opération bloquante `MPI_RECEIVE` tant que la satisfaisabilité ou l'insastifaisabilité de l'instance n'a pas été trouvée. Lorsque qu'une clause est reçue via un `MPI_RECEIVE`, celle-ci est directement placée dans le (BUFFER) en prenant soin de bloquer puis débloquer un *mutex*.

Théoriquement, un *thread* de recherche est bloqué lors de l'envoi d'une clause tant que cette dernière n'est pas reçue par les autres ordinateurs. Toutefois, en pratique, un tampon est utilisé afin de permettre aux solveurs d'envoyer des clauses sans attendre la fin de la communication. Ainsi, l'opération `MPI_SEND` est bloquée uniquement lorsque ce tampon est plein. Cela permet aux solveurs de se concentrer sur la recherche et pas sur l'envoi des clauses.

Concernant le *thread* receveur de chaque ordinateur, il reçoit les clauses les unes après les autres en utilisant l'opération bloquante `MPI_RECEIVE`. Par conséquent, ce *thread* effectue une attente active tant que le message n'a pas été reçu. L'attente active peut alors diminuer la rapidité des solveurs séquentiels associés. En effet, dans cette situation, le *thread* receveur lit dans le fichier descripteur du *socket* réseau tant qu'un message n'a pas été reçu. Le nombre d'opérations alors effectuées par le processeur peut alors ralentir les autres *threads*, notamment, ceux dédiés à la recherche. Heureusement, nous avons observé que cette situation n'arrive quasiment jamais. En effet, le nombre de clauses à partager sur le réseau est gigantesque car elles proviennent de tous les *threads* de recherche et que chacun d'entre eux peut produire un nombre de clauses apprises exponentiel par rapport à la taille de la formule initiale. Par conséquent, les *threads* receveurs ne font que recevoir des clauses et réalisent que très peu d'attente active. De plus, notons que malgré son défaut d'efficacité, une attente active

permet de recevoir les clauses plus rapidement qu'une communication ne la réalisant pas.

Pour finir la description de ce modèle, notons aussi que nous devons limiter d'une manière heuristique le nombre de clauses à envoyer. Pour cela, l'heuristique utilisée est celle par défaut dans SYRUP.

Cette dernière apporte plus de précision sur les problèmes de *deadlocks*, qui peuvent subvenir lors de l'utilisation de la mémoire partagée ou du passage de messages.

Comme il n'y a aucun cycle de communications, nous devons utiliser une autre manière d'éviter les *deadlocks* (interblocages). Pour cela, nous utilisons la notion de *thread safety*. Celle-ci assure que les *threads* exécutant des opérations `MPI_RECEIVE` et/ou `MPI_SEND` n'entraînent jamais de *deadlocks* quels que soient leurs ordres d'exécutions [4]. Cela est garanti par un algorithme distribué d'exclusion mutuelle [11, 9, 10, 14] qui s'assure que seulement un *thread* peut exécuter une section critique à un temps donné. Le surcoût engendré par la notion de *thread safety* a été largement étudié dans [13]. Néanmoins, la plupart des implémentations de la norme MPI ne possèdent pas d'algorithmes assez sophistiqués. En effet, à partir d'un certain nombre de cœurs de calcul, le *thread safety* n'est plus assuré par certaines implémentations. Par exemple, avec l'implémentation OpenMPI, nous notons l'apparition de *deadlocks* par passage de messages à partir de 32 cœurs de calcul. Pourtant, nous avons trouvé une implémentation nommée MPICH qui fonctionne bien et supporte le *thread safety* avec un grand nombre de cœurs de calcul. Des recherches sont encore d'actualité afin d'améliorer le *thread safety*. Par exemple, dans [7], les auteurs détaillent un nouveau mécanisme conceptuel pour MPI afin de faire face à ce problème. À notre connaissance, aucun solveur SAT parallèle distribué n'est basé sur ce modèle de programmation.

Observations Empiriques

Toujours en utilisant le même protocole que pour les autres modèles de programmation (sections 4 et section 5), nous évaluons l'impact du modèle de programmation complètement hybride sur les performances globales du solveur. Le tableau 4 expose les résultats. Rappelons que comme les clauses sont envoyées dès que possible, il n'y a aucun temps de cycle de communications dans ce tableau pour le modèle complètement hybride. Par conséquent, nous pouvons juste examiner l'impact des différentes tailles du BUFFER. De plus, afin de comparer les différents modèles, nous avons choisi d'afficher à coté des résultats du modèle complètement hybride les deux meilleures configurations des modèles partiellement hybrides et purs par passage de messages. Pour ces deux derniers modèles, la meilleure configuration était celle avec un temps des cycles de communication de 5 secondes et une taille du BUFFER fixée à 20MB (tableaux 2 et 3).

Modèle de programmation	**Complètement hybride**			**Partiellement hybride**	**Pur passage de messages**
Taille du BUFFER	**20MB**	**100MB**	**200MB**	**20MB**	**20MB**
Reçues	35,776	32,840	33,343	38,448	30,989
Retenues	83%	86%	86%	68%	25%
Ret. (conflicts)	477	445	473	318	55
Ret. (useful)	5,845	5,899	6,184	3,256	618
Propagations	626,658	629,126	614,832	636,850	489,289
Conflits	1344	1246	1268	1364	823

Table 4 – Modèle de programmation complètement hybride. Impact de la taille des BUFFER sur les performances globales des solveurs. Tous ces résultats correspondent à la moyenne obtenue par tous les processus par seconde sur 20 instances variées de la SAT race 2015. Le temps des cycles de communication est de 5 secondes pour le modèle partiellement hybride et le modèle pur par passage de messages tandis qu'il n'y en a pas pour le modèle complètement hybride car ce dernier envoie les clauses dès que possible. Une focalisation est faite sur une taille du BUFFER fixée à 20MB.

Ici encore, le meilleur compromis est d'avoir un BUFFER à 20 MB pour le modèle de programmation complètement hybride. De plus, nous observons des taux de propagations et de conflits par seconde comparables au modèle de programmation partiellement hybride. Plus intéressant, le nombre de clauses retenues utiles et en conflit (lignes 5 et 6) est beaucoup plus élevé pour le modèle complètement hybride. En d'autres termes, l'échange des clauses de ce modèle est plus efficace car celles-ci sont plus utiles à la recherche que pour les autres modèles.

7 Expérimentations

Dans la première partie de cette section, nous comparons les trois différents modèles de programmation sur les 100 instances de la SAT Race de 2015 (*parallel track*) afin d'ajuster notre solveur et de montrer quel est le meilleur modèle de programmation. Ensuite, nous comparons le meilleur modèle de programmation trouvé avec les solveurs SAT *portfolio* distribués TOPOSAT et HORDESAT sur les 300 instances fournies par la compétition SAT de 2016.

7.1 Comparaison des différents modèles de programmation

D'abord, nous comparons les trois modèles de programmation : le pur par passage de messages, le partiellement hybride et le complètement hybride sur les 100 instances de la compétition SAT-*race* 2015 du *track* parallèle. Comme pour les sections précédentes 6, 5 et 4, nous utilisons 32 ordinateurs contenant chacun 8 cœurs de calcul (32 bi-processors Intel XEON X7550

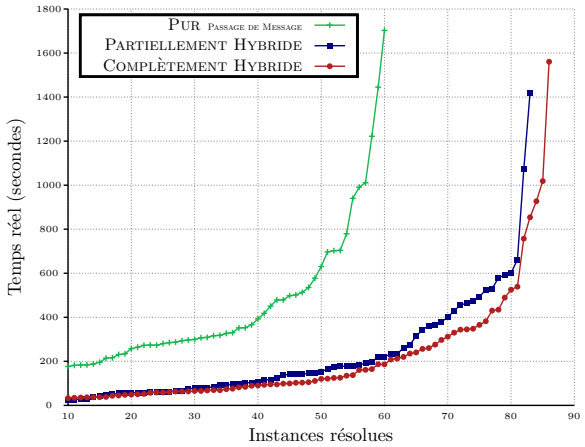


Figure 5 – Résultats expérimentaux des trois différents modèles de programmation sur les 100 instances du *parallel track* de la SAT-*race* 2015.

Modèle de programmation	sat	unsat	Total
Pur par passage de messages	25	26	61
Partiellement hybride	51	33	84
Complètement hybride	54	33	87

Table 5 – Résultats des trois différents modèles de programmation en nombre d'instances résolues.

à 2 GHz avec un gigabyte Ethernet controller). Ces machines nous apportent donc un total de 256 cœurs de calcul. Le temps réel maximum accordé à la résolution est de 1,800 secondes pour chaque instance. Nous utilisons la meilleure configuration trouvée dans les sections précédentes (6, 5 et 4). Ces configurations ont alors des temps de cycle de communications de 5 secondes pour le modèle pur par passage de messages et le modèle partiellement hybride et une taille du BUFFER à 20MB pour tous les modèles. Les résultats sont reportés dans la figure 5 et le tableau 5.

Nous pouvons observer que ces résultats concordent avec ceux des sections précédentes et nos analyses sur leurs performances. Notamment, le modèle pur par passage de messages est le moins bon des modèles en terme de performance. Tous ces résultats valident nos études expérimentales préliminaires réalisées dans les sections précédentes. Cela valide également que plus nous avons de clauses retenues, plus les performances du solveur sont élevées. Par conséquent, il apparaît évident que pour augmenter le nombre de clauses partagées utiles, une bonne pratique est de les partager dès que possible. Nous appelons D-Syrup notre solveur distribué et basé sur le modèle complètement hybride.

7.2 Évaluation de D-Syrup

Dans cette section, nous comparons la version complètement hybride de Syrup, nommé D-Syrup, contre le solveur partiellement hybride HordeSAT qui est composé de solveurs Plingeling et contre le modèle pur par passage de messages TopoSAT qui est composé de solveurs Glucose (version 3). Tout ces solveurs utilisent 256 *threads*, c'est-à-dire, 32 ordinateurs possédant 8 cœurs chacun. Le matériel utilisé est le même que la section précédente (32 bi-processors Intel XEON X7550 à 2 GHz avec un gigabyte Ethernet controller). Afin d'être plus juste et montrer que notre solveur est efficace sur d'autres types d'instances que celles sur lesquelles nous l'avons développé, nous changeons les instances à résoudre. Au lieu de prendre celles de la section précédente (SAT race 2015), cette fois, nous utilisons les 300 instances provenant de la SAT compétition de 2016 (*application track*). Étant donné le nombre important d'instances, nous utilisons un temps limite à la résolution de 900 secondes. De plus, afin d'apporter une évaluation complète de D-Syrup, nous avons aussi ajouté les résultats du solveur séquentiel Glucose (1 cœur), du solveur parallèle *multi-thread* Syrup (8 cœurs) et d'une version de notre solveur distribué D-Syrup avec 128 cœurs (16 machines de 8 cœurs). Cela a pour but de voir si D-Syrup maintient une bonne extensibilité sur cet ensemble d'instances.

Solveur	#Cœurs	sat	unsat	Total
D-Syrup	256	75	109	184
HordeSAT	256	70	84	154
TopoSAT	256	69	58	127
D-Syrup	128	73	104	177
Syrup	8	56	75	131
Glucose	1	49	57	106

Table 6 – Résultats de Glucose (1 cœur), Syrup (8 cœurs), D-Syrup (128 et 256 cœurs), HordeSAT (256 cœurs) et TopoSAT (256 cœurs) sur les 300 instances de la compétition SAT 2016.

Les résultats sont reportés sur la figure 6 et le tableau 6. Clairement, sur cet ensemble d'instances, D-Syrup surpasse HordeSAT et TopoSAT. HordeSAT semble pénalisé par des cycles de communication trop courts (1 seconde) entraînant trop d'accès dans les sections critiques. De plus, le $\text{BUFFER}_{network}$ d'HordeSAT d'une taille fixe limite le partage (section 5). Concernant TopoSAT, il est clairement inefficace car il est basé sur le modèle pur par passage de messages. Ces résultats montrent aussi que la stratégie paresseuse à la base de Syrup maintient son efficacité quel que soit le nombre de cœurs utilisés.

Quand nous comparons D-Syrup avec 128 et 256

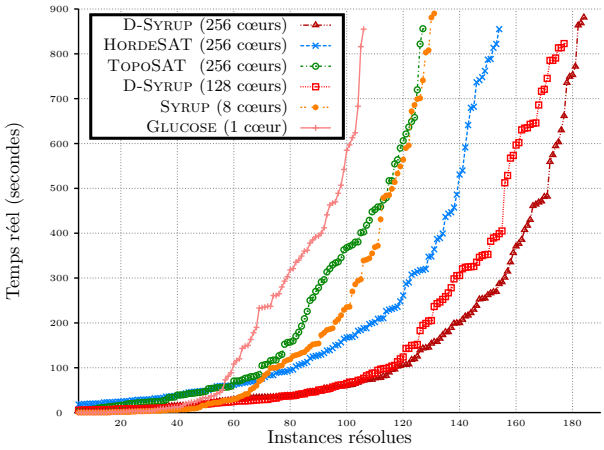


Figure 6 – Résultats de Glucose (1 cœur), Syrup (8 cœurs), D-Syrup (128 et 256 cœurs), HordeSAT (256 cœurs) et TopoSAT (256 cœurs) sur les 300 instances de la compétition sat 2016 (*cactus plot*).

cœurs de calcul, dans la figure 6, nous observons que notre solveur a une bonne extensibilité en nombre d'instances résolues. En effet, quand nous comparons Syrup sur 8 cœurs à D-Syrup, nous pouvons noter que D-Syrup résout 53 instances de plus, en d'autres termes, c'est environ 18% d'instances résolues en plus.

8 Conclusion

Nos expérimentations montre que partager les clauses aussi vite que possible entraîne un impact positif sur l'efficacité des solveurs. Le nombre d'instances résolues augmente toujours (la version D-Syrup de 256 cœurs résout 7 instances de plus que celle de 128 cœurs). Ces travaux de thèse sont plus détaillés dans [12].

Références

[1] Gilles Audemard, Jean-Marie Lagniez, Nicolas Szczepanski, and Sébastien Tabary. An adaptive parallel SAT solver. In *Principles and Practice of Constraint Programming - 22nd International Conference, CP'16, Proceedings*, pages 30–48, 2016.

[2] Gilles Audemard, Jean-Marie Lagniez, Nicolas Szczepanski, and Sébastien Tabary. A distributed version of syrup. In *Proceedings of International Conference on Theory and Applications of Satisfiability Testing (SAT'17)*, Lecture Notes in Computer Science, pages 215–232, 2017.

[3] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern sat solver. In *Twenty-first International Joint Conference on Artificial Intelligence(IJCAI'09)*, pages 399–404, jul 2009.

[4] Pavan Balaji, Darius Buntinas, David Goodell, William Gropp, and Rajeev Thakur. Fine-grained multithreading support for hybrid threaded MPI programming. *International Journal of High Performance Computing Applications IJHPCA*, 24(1) :49–57, 2010.

[5] Tomas Balyo, Peter Sanders, and Carsten Sinz. Hordesat : A massively parallel portfolio SAT solver. In *Proceedings of the Eighteenth International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 9340 of *Lecture Notes in Computer Science*, pages 156–172. Springer, 2015.

[6] Thorsten Ehlers, Dirk Nowotka, and Philipp Sieweck. Communication in massively-parallel SAT solving. In *ICTAI*, pages 709–716. IEEE Computer Society, 2014.

[7] Ryan E. Grant, Anthony Skjellum, and Purushotham V.Bangalore. *Lightweight Threading with MPI Using Persistent Communications Semantics*. United States. National Nuclear Security Administration, 2015.

[8] Ajay D. Kshemkalyani and Mukesh Singhal. Efficient detection and resolution of generalized distributed deadlocks. *IEEE Transactions on Software Engineering TSE*, 20(1) :43–54, 1994.

[9] Sandeep Lodha and Ajay D. Kshemkalyani. A fair distributed mutual exclusion algorithm. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 11(6) :537–549, 2000.

[10] Shojiro Nishio, Kin F. Li, and Eric G. Manning. A resilient mutual exclusion algorithm for computer networks. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 1990.

[11] Mukesh Singhal. A taxonomy of distributed mutual exclusion. *Journal of Parallel and Distributed Computing*, 18(1) :94–101, 1993.

[12] Nicolas Szczepanski. *SAT en parallele*. PhD thesis, Universite d'Artois Jean Perrin, France, 2017.

[13] Rajeev Thakur and William Gropp. Test suite for evaluating performance of MPI implementations that support mpi_thread_multiple. In *Parallel Virtual Machine PVM/MPI*, volume 4757, pages 46–55. Springer, 2007.

[14] Weigang Wu, Jiebin Zhang, Aoxue Luo, and Jiannong Cao. Distributed mutual exclusion algorithms for intersection traffic control. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 26(1) :65–74, 2015.

Identification de paramètres dynamiques de réseaux de gènes

Jonathan Behaegel[1*] **Jean-Paul Comet**[1] **Marie Pelleau**[1]

[1] Université Côte d'Azur, CNRS, I3S, France

behaegel@i3s.unice.fr comet@unice.fr marie.pelleau@i3s.unice.fr

Résumé

L'étude des réseaux de gènes nous permet de mieux comprendre certains processus biologiques tels que l'adaptation de l'organisme à une perturbation de l'environnement. Dans le cadre d'une modélisation discrète des réseaux de gènes, il a été montré que la logique de Hoare pouvait aider à l'identification des paramètres du modèle afin que ce dernier exhibe les traces biologiques observées. Dans cet article nous présentons une modélisation hybride des réseaux de gènes mettant l'accent sur le temps passé dans chacun des états et nous introduisons une extension de la logique de Hoare dans ce cas hybride. Le calcul de la plus faible précondition associé à cette logique de Hoare modifiée permet de déterminer les contraintes minimales sur les paramètres dynamiques d'un réseau de gènes à partir d'une trace biologique observée. Ces contraintes forment un CSP continu que nous résolvons à l'aide du solveur continu AbSolute. Les premiers résultats expérimentaux montrent que les solutions exhibées sont en accord avec les spécifications du triplet de Hoare provenant de l'expertise biologique.

Abstract

The study of gene networks allows us to better understand some biological processes such as the adaptation of the organism to a disturbance of the environment. In a discrete modelling framework of gene networks, it has been shown that the Hoare logic can help the modeller to identify the parameters of the model so that the latter exhibits the observed biological traces. In this paper we present a hybrid modelling of gene networks which pays particular attention to the time spent in each state and we introduce an extension of the Hoare logic in this hybrid case. The weakest precondition calculus associated with this modified Hoare logic makes it possible to determine the minimal constraints on the dynamic parameters of a gene network from an observed biological trace. These constraints form a continuous CSP that we solve using the AbSolute continuous solver. The first experimental results show that the obtained solutions are in agreement with the specification of the Hoare triple coming from biological expertise.

1 Introduction

La modélisation des réseaux de régulation génétique a pour but l'étude et la compréhension des mécanismes moléculaires permettant à l'organisme d'assurer les fonctions essentielles allant du métabolisme à l'adaptation à une perturbation de l'environnement. Les réseaux de gènes sont décrits par des régulations qui peuvent être classées en deux types : les activations et les inhibitions. La combinaison de ces régulations permettent de nombreux comportements et on peut montrer que la complexité de ces systèmes provient des rétrocontrôles positifs et négatifs couramment observés qui mènent à la multistationnarité et à l'homéostasie (capacité à maintenir un équilibre). L'étude de la dynamique de ces systèmes ouvre de nouvelles perspectives avec des applications en pharmacologie, en médecine, ou en toxicologie... Différents cadres de modélisation (qualitative, continue, stochastique, hybride) sont possibles mais quel que soit le cadre choisi, un point crucial du processus de modélisation réside dans la détermination des paramètres qui gouvernent la dynamique du modèle et leur identification reste l'étape limitante.

Le cadre de modélisation discret de René Thomas [17] ne cherche à représenter que l'enchaînement d'événements qualitatifs et les méthodes formelles classiques (model checking, programmation par contraintes, logique de Hoare) ont été utilisées avec succès pour automatiser l'identification des paramètres gouvernant la dynamique [4, 7, 3]. Cependant, certains phénomènes biologiques font intervenir une

*Papier doctorant : Jonathan Behaegel[1] est auteur principal.

composante temporelle jouant un rôle primordial : le cycle circadien en est un bel exemple puisqu'il permet à l'organisme de s'adapter à l'alternance jour/nuit. Pour modéliser ces phénomènes, la prise en compte du temps impose *a minima* un cadre de modélisation hybride qui ajoute à l'approche de René Thomas une mesure du temps passé dans chacun des états. Notre objectif est alors de chercher à automatiser l'identification des paramètres d'un modèle hybride pour que ce dernier soit en accord avec les observations expérimentales (variations de la concentration des protéines d'un système biologique au cours de la journée, données très difficiles à obtenir pour un grand nombre de protéines). Ces observations sont représentées par une trace biologique qui exprime un enchaînement d'événements qualitatifs séparés par les durées mesurées. Pour cela, nous avons modifié la logique de Hoare ainsi que son calcul de plus faible précondition pour construire un système de contraintes résolue par le solveur AbSolute [14].

La programmation par contraintes a déjà été utilisée pour résoudre des problèmes liés aux réseaux biologiques. Dans le cadre discret, elle a été utilisée pour la recherche des paramètres cinétiques [7], ou la recherche de bifurcations [10]. Elle a aussi été utilisée pour l'optimisation de modes de flux dans des réseaux métaboliques [13]. Dans le cadre continu, l'approche par contraintes a permis de vérifier des propriétés temporelles du modèle [5], ou identifier les paramètres dynamiques d'un système décrit par équations différentielles dans le cadre de la biologie synthétique [15]. Notre but est d'aborder pour des modèles hybrides le problème d'identification des paramètres dynamiques alors que généralement ce sont les problèmes de vérification [1, 12] ou d'approximation de modèle [11] qui sont abordés.

L'article présente en section 2 le cadre de modélisation hybride utilisé pour décrire la dynamique des réseaux de gènes. La section 3 décrit la logique de Hoare ainsi que son calcul de plus faible précondition adapté à ce cadre de modélisation hybride. Le solveur AbSolute est esquissé en section 4 en mettant l'accent sur la réécriture symbolique et la notion de solution sûre. Nous présentons en section 5 nos résultats expérimentaux d'identification de paramètres dynamiques grâce à AbSolute et nous finissons par discuter l'approche proposée.

2 Modélisation d'un réseau de gènes

De nombreux cadres de modélisation dédiés aux réseaux génétiques peuvent être utilisés pour représenter la dynamique du réseau de gènes. Un des premiers cadres de modélisation qui a été totalement formalisé est celui introduit par René Thomas [17, 4] mais le temps séparant deux événements qualitatifs a été totalement abstrait. Ici, nous présentons un cadre de modélisation hybride inspiré de l'approche discrète de Thomas [2].

2.1 Représentation d'un réseau de gènes

Les réseaux de gènes sont représentés par des graphes étiquetés, voir figure 1. Les deux types de sommets permettent de décrire d'une part les entités (abstractions de gènes et protéines associés, représentées par des cercles) et d'autre part les régulations entre ces entités (représentées par des rectangles) et appelées multiplexes. L'ensemble des entités est noté V. Les arcs pointant sur un multiplexe (arcs en pointillés) spécifient les entités prenant part à la régulation et l'ensemble des prédécesseurs d'une entité $v \in V$ est noté $R^-(v)$. Les arcs sortant d'un multiplexe (arcs pleins) spécifient les entités qui sont *positivement* régulées par le multiplexe. Comme les régulations ne sont pas toujours actives, on associe à chaque multiplexe une formule décrivant les conditions sous lesquelles la régulation associée au multiplexe a lieu.

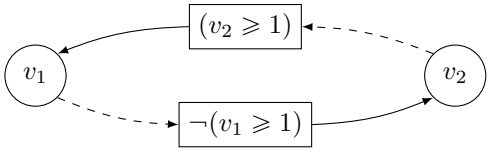


FIGURE 1 – Graphe d'interaction représentant une boucle de rétroaction négative simple.

La figure 1 présente un graphe d'interaction avec 2 entités v_1 et v_2 : v_2 active v_1 et v_1 inhibe v_2 (l'inhibition est représentée par la négation de la formule). Le tout forme une boucle de rétroaction négative puisque chacune des 2 entités a une action négative indirecte sur elle-même.

2.2 Dynamique d'un réseau de gènes

Le cadre de modélisation hybride [2] repose sur la discrétisation des concentrations des entités biologiques, tout en incluant le temps passé entre deux événements successifs. Supposons que l'entité v agisse sur d'autres entités à différents seuils et que le nombre de seuils différents soit b_v. Alors l'ordonnancement de ces seuils mène à la définition des niveaux qualitatifs de l'entité v : le niveau 0 représente l'intervalle des concentrations inférieures au premier seuil, le niveau b_v représente l'intervalle des concentrations supérieures au plus grand seuil et le niveau k représente l'intervalle des concentrations comprises entre le k-ème et le

$(k{+}1)$-ème seuil ($k \in [\![0, b_v[\![$). Un état qualitatif du système est la donnée d'un niveau qualitatif pour chacune des entités.

A chaque état qualitatif est associé un "espace temporel" qui peut être vu comme l'hypercube $[0,1]^n$ où n est le nombre d'entités, voir les carrés de la figure 2. L'état du système est caractérisé non seulement par l'état qualitatif, mais aussi par la position précise au sein de l'espace temporel. Les états du système sont ainsi appelés états hybrides et sont définis par un couple de vecteurs $h = (\eta, \pi)$ où η, appelé état discret, est le vecteur des niveaux qualitatifs des entités (la composante v du vecteur est un entier positif inférieur ou égal à b_v) et $\pi \in [0,1]^n$, appelé partie fractionnaire, est le vecteur des positions exactes à l'intérieur de l'espace temporel associé à cet état discret, voir figure 2. Par convention, η_v (resp. π_v) dénote le niveau qualitatif (resp. la partie fractionnaire) de l'entité v. Deux états discrets seront dits voisins s'ils ne diffèrent que d'une seule composante et si cette composante ne diffère que de un.

L'évolution au sein de chaque état qualitatif, représentée par des *transitions continues*, est gouvernée par des célérités à identifier, et les transitions d'un état à un autre appelées *transitions discrètes* sont possibles sous certaines conditions.

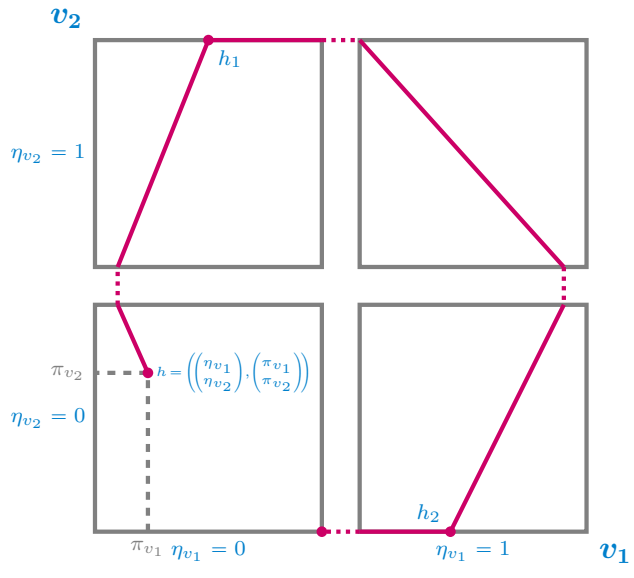


FIGURE 2 – Une dynamique possible associée au graphe d'interaction de la figure 1.

La figure 2 montre l'évolution d'un modèle possible pour le graphe d'interaction de la figure 1 en partant du point h à l'intérieur de l'état discret $(\eta_{v_1}, \eta_{v_2}) = (0,0)$. Tant qu'on reste dans cet état discret, la concentration de v_2 augmente. Lorsque v_2 ne peut plus augmenter, on change d'état discret $(\eta_{v_1}, \eta_{v_2}) = (0,1)$. Dans ce nouvel état, v_1 et v_2 augmentent jusqu'à saturation de v_2. v_1 continue alors d'augmenter ce qui permet d'entrer dans l'état discret $(1,1)$... Finalement la trace de la figure 2 s'arrête dans l'état discret $(0,0)$ où $\pi_{v_1} = 1$ et $\pi_{v_2} = 0$.

Un multiplexe est actif dans un état hybride h si sa formule associée est évaluée à vrai après substitution des entités par leurs niveaux qualitatifs. Les multiplexes actifs seront appelés *ressources* et l'ensemble des ressources de v à l'état h est noté $\rho(h,v)$. Par exemple, dans la figure 2 on a $\rho(h, v_1) = \varnothing$ car la formule du seul multiplexe régulant v_1 est évaluée à faux dans cet état, et $\rho(h, v_2) = \{v_1\}$ car la formule du seul multiplexe régulant v_2 est évaluée à vrai dans cet état. Notons que les ressources d'une entité à l'état hybride h ne dépendent que des niveaux qualitatifs des entités : les ressources ne varient pas à l'intérieur d'un "espace temporel".

Les différentes entités n'évoluent évidemment pas à la même vitesse dans chacun des "espaces temporels". De même, la vitesse d'une entité dépend de ses ressources et de son niveau qualitatif. Ainsi, la dynamique des entités est contrôlée par les célérités $C_{v,\omega,k} \in \mathbb{R}$ indexées par l'entité v, par ω l'ensemble des ressources de v dans l'état discret courant, et par le niveau qualitatif k de l'entité v. Par exemple, dans l'état discret $(0,0)$ de la figure 2, la célérité contrôlant l'entité v_1 est $C_{v_1,\varnothing,0}$ et la célérité contrôlant l'entité v_2 est $C_{v_2,\{v_1\},0}$. Une célérité nulle indique que l'entité a atteint un état d'équilibre. Pour atteindre cet état d'équilibre, il faut de plus que les célérités de cette entité, soumises aux mêmes régulations, se dirigent vers cet état d'équilibre :

$$\forall v \in V,\ \forall \omega \subset R^-(v),\ \forall n \in [\![0, b_v]\!],\quad C_{v,\omega,n} = 0 \Rightarrow \begin{cases} \forall i \in [\![n+1, b_v]\!] & C_{v,\omega,i} < 0 \\ \forall i \in [\![0, n-1]\!] & C_{v,\omega,i} > 0 \end{cases} \tag{1}$$

Dans tous les cas, les célérités d'une même entité avec les mêmes ressources dans des états discrets voisins se dirigent dans la même direction et donc ont le même signe :

$$\forall v \in V,\ \forall \omega \subset R^-(v),\ \forall k \in [\![0, b_v - 1]\!],\quad C_{v,\omega,k} \times C_{v,\omega,k+1} \geqslant 0 \tag{2}$$

Dans un état discret, si la célérité de v est positive (resp. négative), on dit que v fait face à un mur externe lorsque le niveau qualitatif maximal (resp. minimal) est atteint $\eta_v = b_v$ (resp. $\eta_v = 0$). Ainsi, v ne peut pas continuer à augmenter (resp. diminuer). Dans les autres cas, si $\eta_v < b_v$ (resp. $\eta_v > 0$), l'entité v fait face à un mur interne lorsque les célérités de v dans l'état courant et dans l'état voisin vers lequel v tendrait, sont de signes contraires.

Enfin, les transitions entre deux états discrets sont possibles lorsque les deux conditions suivantes sont respectées : une entité v doit être sur le bord de l'état discret ($\pi_v = 0$ ou 1) et cette entité doit pouvoir changer de niveau qualitatif (ne pas faire face à un mur). La définition 1 introduit la notion d'entité glissante :

Définition 1 *Soit un réseau de gènes, $v \in V$ une entité et $h = (\eta, \pi)$ un état hybride.*

1. *v fait face à un mur externe en h si :*
   $\big((C_{v,\rho(h,v),\eta_v} < 0) \wedge (\eta_v = 0)\big) \vee$
   $\big((C_{v,\rho(h,v),\eta_v} > 0) \wedge (\eta_v = b_v)\big)$.
2. *Soit $h' = (\eta', \pi')$ un autre état hybride tel que $\eta'_v = \eta_v + \mathrm{sgn}(C_{v,\rho(h,v),\eta_v})$ et $\eta'_u = \eta_u$ pour tout $u \in V, u \neq v$. v fait face à un mur interne en h si* $\mathrm{sgn}(C_{v,\rho(h,v),\eta_v}) \times \mathrm{sgn}(C_{v,\rho(h',v),\eta'_v}) = -1$, *avec* sgn *la fonction signe usuelle, $\rho(h, v)$ les ressources de v dans h.*

On note $\mathsf{sv}(h)$ *l'ensemble des* entités glissantes, *c.-à-d. qui font face à un mur interne ou externe en h.*

Dans la figure 2, l'entité v_2 fait face à un mur externe, elle est donc glissante dans les états discrets $(0, 1)$ et $(1, 0)$.

Pour chaque entité, le temps pour atteindre un bord est appelé délai de contact, qui dépend de la célérité de l'entité et de la distance à parcourir dans l'état hybride h (définition 2). Notons que lorsque la célérité d'une entité est nulle, le délai de contact est infini.

Définition 2 (Délai de contact) *Soit un réseau de gènes, v une entité de V et $h = (\eta, \pi)$ un état hybride. Nous notons $\delta_h(v)$ le* délai de contact *de v en h pour atteindre le bord de l'état discret. δ_h est la fonction de V dans $\mathbb{R}^+ \cup +\infty$ définie par :*

- *Si $C_{v,\rho(h,v),\eta_v} = 0$ alors $\delta_h(v) = +\infty$;*
- *Si $C_{v,\rho(h,v),\eta_v} > 0$ (*resp. *< 0) alors $\delta_h(v) = \frac{1-\pi_v}{C_{v,\rho(h,v),\eta_v}}$ (*resp. *$\frac{-\pi_v}{C_{v,\rho(h,v),\eta_v}}$).*

Nous définissons également l'ensemble *first* des entités atteignant leur bord en premier à partir de l'état hybride h (c.-à-d. les entités ayant un délai de contact minimum dans l'état hybride courant), en excluant les entités *glissantes*. Ainsi, l'ensemble *first* représente l'ensemble des entités qui vont pouvoir changer de niveau qualitatif et mener à des transitions discrètes.

La définition complète de la dynamique d'un réseau de gènes (transitions continues et transitions discrètes) est donnée dans [2]. La partie suivante introduit un moyen de construire les contraintes sur les paramètres dynamiques d'un réseau de gènes.

3 Logique de Hoare

La logique de Hoare a été développée pour prouver la correction des programmes impératifs [9]. Elle s'appuie sur la notion de triplet de Hoare de la forme $\{Pre\}\ p\ \{Post\}$ où Pre et $Post$ représentent des conditions sur l'état du système et p est un programme impératif. D'un point de vue intuitif, un triplet sera dit correct si l'exécution du programme p, à partir d'un état où la précondition Pre est satisfaite, est possible et mène à un état où la postcondition $Post$ est satisfaite. Les règles d'inférence associées à l'affectation, l'instruction conditionnelle, la composition de programme et le *while* permettent de prouver efficacement qu'un programme, s'il termine, est correct.

Dans cet article, nous modifions la logique de Hoare pour pouvoir synthétiser les contraintes nécessaires sur les paramètres dynamiques du modèle assurant que le modèle exhibe une trace biologique observée. Cette approche a été développée dans le cadre de modélisation discrète [3] et nous présentons ici l'extension au cadre hybride présentée dans [2].

3.1 Logique de Hoare modifiée

Dans le cadre de la modélisation des réseaux de gènes, le programme impératif p est remplacé par une trace biologique observée écrite de manière formelle dans le langage de chemins, et les conditions Pre et $Post$ expriment des propriétés sur les états hybrides de départ et d'arrivée.

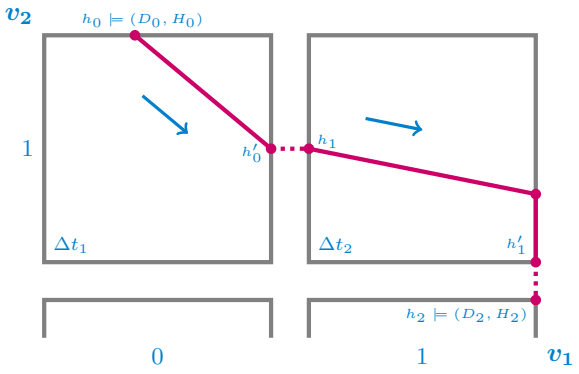


FIGURE 3 – Trace biologique avec des transitions continues et discrètes. Les pré et postconditions sont (D_0, H_0) et (D_2, H_2). Δt_1 et Δt_2 représentent les temps passés dans les états qualitatifs et les vecteurs représentent les vecteurs célérités.

Les pré et postconditions sont exprimés dans un langage de propriété : une propriété est un couple de la forme (D, H), où la condition discrète D décrit exclusivement l'état discret η de chaque entité biologique, et

la condition hybride H porte sur les parties fractionnaires π de ces mêmes entités. Si aucune connaissance sur la condition D (resp. H) n'est disponible, cette dernière est représentée par la tautologie True. La figure 3 montre un chemin dont le point de départ h_0 est défini par la précondition $(\eta(v_1) = 0 \wedge \eta(v_2) = 1, \pi_{v_1} = 0.4 \wedge \pi_{v_2} = 1)$.

Le langage de chemins permet de décrire les traces biologiques : il comporte le chemin vide ϵ, les chemins élémentaires qui sont de la forme $(\Delta t, a, dpa)$ et les chemins non élémentaires composés d'une séquence finie de chemins élémentaires. Un chemin élémentaire est défini par un triplet $(\Delta t, a, dpa)$:

- L'atome de chemin discret *dpa* (discrete path atom), de la forme $v+$ (resp. $v-$), exprime la prochaine transition vers un état discret voisin : le niveau qualitatif de v est incrémenté (resp. décrémenté).
- L'élément a permet de spécifier des propriétés relatives à la dynamique au sein de l'état discret courant. Le modélisateur peut caractériser les comportements à l'aide du langage d'assertion suivant : $a :== \top \mid C_v \,\square\, c \mid (\mathsf{no})\mathsf{slide}(v) \mid (\mathsf{no})\mathsf{slide}^+(v) \mid (\mathsf{no})\mathsf{slide}^-(v) \mid \neg a \mid a \wedge a \mid a \vee a$ où $\square$ appartient à l'ensemble des symboles de comparaison. Le terme $\mathsf{slide}^+(v)$ (resp. $\mathsf{slide}^-(v)$) indique que v va glisser le long de son bord supérieur (resp. inférieur), tandis que $\mathsf{noslide}^+(v)$ (resp. $\mathsf{noslide}^-(v)$) empêche ce glissement. Le terme $\mathsf{slide}(v)$ (resp. $\mathsf{noslide}(v)$) désigne la disjonction (resp. conjonction) de $\mathsf{slide}^+(v)$ et $\mathsf{slide}^-(v)$ (resp. $\mathsf{noslide}^+(v)$ et $\mathsf{noslide}^-(v)$). Notons que dans un chemin élémentaire, une même entité ne peut apparaître à la fois dans le *dpa* et dans un terme slide. Pour exprimer que la trace de la figure 3 possède un glissement de v_1 dans l'état discret où $\eta_{v_1} = 1$ et $\eta_{v_2} = 1$, on peut écrire $\mathsf{slide}^+(v_1)$. Ce langage d'assertion reprend le langage défini en [2] en le complétant par les atomes $\mathsf{noslide}^{\pm}(v)$ qui permettent de mieux décrire les données expérimentales.
- Enfin, Δt indique le temps exact passé dans l'état discret courant.

Ainsi, le triplet de Hoare représentant la trace de la figure 3 s'écrit :

$$\begin{Bmatrix} D_0 \\ H_0 \end{Bmatrix} \begin{pmatrix} \Delta t_1 \\ \top \\ v_1+ \end{pmatrix}; \begin{pmatrix} \Delta t_2 \\ \mathsf{slide}^+(v_1) \\ v_2- \end{pmatrix} \begin{Bmatrix} D_2 \\ H_2 \end{Bmatrix} \tag{3}$$

On dira qu'une transition continue $h \longrightarrow h'$ satisfait le couple d'assertion $(\Delta t, a)$ si la transition existe, si cette transition dure Δt et si elle respecte l'assertion a ($C_v \,\square\, c$ est satisfait si $C_{v,\rho(h,v),\eta_v} \,\square\, c$, $\mathsf{slide}^+(v_1)$ est satisfait si π_{v_1} atteint 1 et v_1 fait face à un mur, *etc.*)

Chaque instruction élémentaire du chemin p correspond à une transition continue d'une durée Δt définie dans l'état discret courant, suivi d'une transition discrète permettant à l'entité du *dpa* d'incrémenter ou de décrémenter son niveau qualitatif. La figure 3 montre qu'à partir de l'état hybride h_0 satisfaisant la précondition de l'équation 3, il existe une transition continue $h_0 \longrightarrow h'_0$ d'une durée Δt_1 menant à une transition discrète $h'_0 \longrightarrow h_1$. Le processus est répété pour l'instruction élémentaire suivante où un glissement en v_1 ($\mathsf{slide}^+(v_1)$) a lieu, jusqu'à atteindre l'état hybride h_2 satisfaisant la postcondition (D_2, H_2).

3.2 Plus faible précondition

Edsger Dijkstra a introduit une sémantique de transformation de prédicats [9] pour les langages de programmation impérative : elle associe à chaque instruction de ce langage une transformation de prédicats. Pour chaque instruction élémentaire, la plus faible précondition est une fonction qui associe à chaque postcondition *Post* une précondition *Pre*. En itérant le processus, on peut construire la plus faible précondition d'un programme impératif complet. Nous définissons de manière similaire la plus faible précondition pour chacun de nos chemins élémentaires :

Définition 3 (Plus faible précondition) *Soit p une trace biologique exprimée dans le langage de chemins et $Post = (D_f, H_f)$ une postcondition indexée par un indice final f. La* plus faible précondition *attribuée à p et $Post$ est une propriété : $\mathsf{WP}^i_f(p, Post) \equiv (D_{i,f}, H_{i,f})$, indexée par un indice initial i et par f définie récursivement de la façon suivante :*

- *Si $p = \epsilon$, alors $D_{i,f} \equiv D_f$ and $H_{i,f} \equiv H_f$;*
- *Si $p = (\Delta t, a, v\pm)$, alors :*

$$\begin{aligned} D_{i,f} \equiv & D_f[\eta_v \backslash \eta_v \pm 1], \\ H_{i,f} \equiv & H_f[\eta_v \backslash \eta_v \pm 1] \wedge \Phi^{\pm}_v(\Delta t) \wedge \mathcal{F}(\Delta t) \wedge \\ & \neg \mathcal{W}^{\pm}_v \wedge \mathcal{A}(\Delta t, a) \wedge \mathcal{J}_v; \end{aligned}$$

- *Si $p = p_1; p_2$ est une séquence de chemins :*

$$\mathsf{WP}^i_f(p_1; p_2, Post) \equiv \mathsf{WP}^i_m(p_1, \mathsf{WP}^m_f(p_2, Post))$$

indexée par un nouvel indice intermédiaire m ; avec $\Phi^{\pm}_v(\Delta t)$, $\mathcal{W}^{\pm}_v$, $\mathcal{F}(\Delta t)$, $\mathcal{A}(\Delta t, a)$ and $\mathcal{J}_v$ des sous-propriétés définies en annexe 7.1.

Intuitivement, chacune des sous-propriétés décrit une condition qui doit être satisfaite pour permettre les transitions continue et discrète associées à un chemin élémentaire :

— $\Phi_v^+(\Delta t)$ (resp. $\Phi_v^-(\Delta t)$) correspond à la contrainte permettant à l'entité v d'atteindre son bord supérieur (resp. inférieur) en un temps Δt et assurant la transition discrète ;
— $\mathcal{F}(\Delta t)$ contraint toutes les entités sauf l'entité v du *dpa* à atteindre leurs bords après v, ou à faire face à un mur ;
— $\neg\mathcal{W}_v^{\pm}$ interdit la présence d'un mur pour l'entité v, l'autorisant ainsi à changer de niveau qualitatif ;
— $\mathcal{A}(\Delta t, a)$ traduit les formules de l'assertion a en contraintes spécifiques à l'état courant ;
— $\mathcal{J}_v$ fait la jonction entre deux états successifs.

Ces 5 sous-propriétés sont décrites avec précision en annexe 7.1.

Sur la figure 3, dans l'état discret $(1, 1)$, v_1 atteint en premier un mur et on a un glissement jusqu'à ce que v_2 atteigne son bord inférieur. La plus faible précondition associée au chemin p et à la postcondition *post* :

$$p = \begin{pmatrix} \Delta t_2 \\ a \\ v_2- \end{pmatrix} \quad post = \begin{Bmatrix} D_2 \\ H_2 \end{Bmatrix} \quad \text{avec } a \equiv \mathsf{slide}^+(v_1)$$

est donnée par

$$\begin{aligned} D_1 \equiv & D_2[\eta_{v_2} \backslash \eta_{v_2} - 1], \\ H_1 \equiv & H_2[\eta_{v_2} \backslash \eta_{v_2} - 1] \wedge \Phi_{v_2}^-(\Delta t_2) \wedge \mathcal{F}(\Delta t_2) \wedge \\ & \neg\mathcal{W}_{v_2}^- \wedge \mathcal{A}(\Delta t_2, a) \wedge \mathcal{J}_{v_2}; \end{aligned}$$

où les sous-propriétés font intervenir les parties fractionnaires de l'état d'entrée h_1 ($\pi^1_{v_1}$ et $\pi^1_{v_2}$) ainsi que les parties fractionnaires de l'état de sortie h'_1 ($\pi^{1'}_{v_1}$ et $\pi^{1'}_{v_2}$), et s'expriment après simplification comme suit :

— v_2 atteint son bord inférieur
$$\begin{aligned} \Phi_{v_2}^-(\Delta t_2) \equiv & (\pi^{1'}_{v_2} = 0) \wedge (C_{v_2,\rho(h'_1,v_2),1} < 0) \wedge \\ & (\pi^1_{v_2} = \pi^{1'}_{v_2} - C_{v_2,\rho(h'_1,v_2),1} \times \Delta t_2) \end{aligned}$$
— v_2 n'atteint pas de mur
$$\neg\mathcal{W}_{v_2}^- \equiv C_{v_2,\rho(h_2,v_2),0} \leqslant 0$$
— v_1 atteint un mur grâce à $\mathsf{slide}^+(v_1)$
$$\begin{aligned} \mathcal{F}(\Delta t_2) \equiv & C_{v_1,\rho(h'_1,v_1),1} > 0 \\ \mathcal{A}(\Delta t_2, a) \equiv & (\pi^1_{v_1} > \pi^{1'}_{v_1} - C_{v_1,\rho(h'_1,v_1),1} \times \Delta t_2) \\ & \wedge (\pi^{1'}_{v_1} = 1) \end{aligned}$$
— Jonction entre les états h_2 et h'_1
$$\begin{aligned} \mathcal{J}_{v_2} \equiv & (\pi^2_{v_2} = 1 - \pi^{1'}_{v_2}) \wedge (\pi^2_{v_1} = \pi^{1'}_{v_1}) \wedge \\ & (C_{v_1,\rho(h'_1,v_1),1} > 0) \wedge \\ & (\pi^1_{v_1} \geqslant \pi^{1'}_{v_1} - C_{v_1,\rho(h'_1,v_1),1} \times \Delta t_2) \end{aligned}$$

L'ensemble de ces sous-propriétés permet de décrire les contraintes sur les célérités et éventuellement comparer les célérités de différentes entités entre elles. Par exemple, les assertions $\mathsf{slide}^{\pm}(v_1)$ et $\mathsf{noslide}^{\pm}(v_1)$ apportent des contraintes entre la célérité de l'entité v_1 et la célérité de l'entité v_2 du *dpa* car elles sont toutes les deux reliées au temps passé dans l'état discret courant.

La stratégie choisie pour le calcul de la plus faible précondition est la même que celle proposée par Dijkstra [9], en partant de la postcondition et en remontant les chemins élémentaires jusqu'au premier (Définition 3). Une implémentation de cette stratégie a été réalisée [1] et la preuve de correction de la logique de Hoare adaptée au cadre de modélisation hybride est donnée dans [2].

3.3 Exemple de contraintes obtenues

En se basant sur la figure 1, on peut construire les contraintes sur les célérités de la boucle négative à partir d'une trace biologique expérimentale. Pour cela, on considère le chemin et la postcondition suivants :

$$\begin{pmatrix} 5.0 \\ \mathsf{nsl}(v_1) \\ v_2+ \end{pmatrix}; \begin{pmatrix} 7.0 \\ \mathsf{slide}^+(v_2) \\ v_1+ \end{pmatrix}; \begin{pmatrix} 8.0 \\ \mathsf{nsl}(v_1) \\ v_2- \end{pmatrix}; \begin{pmatrix} 4.0 \\ \mathsf{slide}^-(v_2) \\ v_1- \end{pmatrix} \begin{Bmatrix} D_4 \\ H_4 \end{Bmatrix}$$

avec $D_4 \equiv (\eta_{v_1} = 0) \wedge (\eta_{v_2} = 0)$ et $H_4 \equiv True$ et nsl une abréviation du symbole $\mathsf{noslide}$.

Nous appliquons la stratégie choisie du calcul de plus faible précondition sur le dernier chemin élémentaire $ce = \begin{pmatrix} 4.0 \\ \mathsf{slide}^-(v_2) \\ v_1- \end{pmatrix}$ en utilisant la postcondition $\begin{Bmatrix} D_4 \\ H_4 \end{Bmatrix}$ et nous calculons $\mathsf{WP}(ce, (D_4, H_4)) \equiv (D_3, H_3)$.

Le temps passé dans l'état courant est de 4.0 heures. Le *dpa* v_1- indique que l'entité v_1 atteint son bord inférieur et le franchit. L'assertion $\mathsf{slide}^-(v_2)$ quant à elle indique que l'entité v_2 atteint son bord inférieur avant que v_1 atteigne le sien.

D'après la définition 3, on peut facilement calculer les niveaux qualitatifs de chaque entité de la condition discrète à partir des niveaux qualitatifs observés en postcondition :

$$D_3 \quad \equiv \quad D_4[\eta_{v_1} \backslash \eta_{v_1} - 1] \quad \equiv \quad \eta_{v_1} = 1 \wedge \eta_{v_2} = 0$$

Les célérités à utiliser dans l'état courant sont alors facilement identifiables grâce à la connaissance des niveaux qualitatifs de chaque entité. Aucun des multiplexes n'agit sur son entité cible, les célérités sont donc $C_{v_1,\varnothing,1}$ et $C_{v_2,\varnothing,0}$.

D'après la définition de la plus faible précondition, la condition hybride H_3 est obtenue par la formule suivante :

$$\begin{aligned} H_3 \equiv & H_4[\eta_{v_1} \backslash \eta_{v_1} - 1] \wedge \Phi_{v_1}^-(4.0) \wedge \mathcal{F}(4.0) \wedge \\ & \neg\mathcal{W}_{v_1}^- \wedge \mathcal{A}(4.0, \mathsf{slide}^-(v_2)) \wedge \mathcal{J}_{v_1} \end{aligned}$$

1. `http://www.i3s.unice.fr/~comet/DOCUMENTS/hybridisation.tar.gz`

Après simplification des contraintes obtenues pour chaque sous-propriété, on obtient :

$$H_3 \equiv (C_{v_1,\varnothing,1} < 0) \wedge (C_{v_2,\varnothing,0} < 0) \wedge \neg(C_{v_1,\varnothing,0} > 0) \wedge (\pi^{3'}_{v_1} = 0) \wedge (\pi^{3'}_{v_2} = 0) \wedge (\pi^{4}_{v_1} = 1 - \pi^{3'}_{v_1}) \wedge (\pi^{4}_{v_2} = \pi^{3'}_{v_2}) \wedge (\pi^{3}_{v_1} = \pi^{3'}_{v_1} - 4.0 \times C_{v_1,\varnothing,1}) \wedge (\pi^{3}_{v_2} < \pi^{3'}_{v_2} - 4.0 \times C_{v_2,\varnothing,0})$$

On réitère ce même procédé successivement sur les trois chemins élémentaires restants pour calculer les contraintes sur toutes les célérités du système. Les contraintes données en annexe 7.2 proviennent de notre outil qui simplifie et réduit les formules.

Le système de contraintes obtenues comprend 26 variables dont 8 célérités (2 célérités pour chacun des 4 états discrets) définies sur $\mathbb{R}$ et 18 parties fractionnaires définies sur $[0, 1]$. En effet, le chemin a 4 chemins élémentaires ce qui définit 4 couples $(D_i, H_i)_{i\in[0..3]}$ et un couple final (D_4, H_4). Pour chacun des 4 couples non finaux, il y a 2 parties fractionnaires pour l'état hybride d'entrée et 2 autres pour l'état hybride de sortie, et pour le couple final, il y a seulement 2 parties fractionnaires pour l'état hybride de sortie. Les simplifications effectuées à la volée permettent de déterminer 6 parties fractionnaires de sortie qui correspondent aux 4 positions de sortie de chacun des *dpa* et aux 2 glissements imposés par la trace biologique. Au final, le CSP contient 38 contraintes dont 6 affectations, voir annexe 7.2. Sur les exemples traités, notre approche a permis de supprimer les contraintes triviales et les contraintes redondantes qui représentent environ la moitié des contraintes générées.

4 Solveur

Afin de résoudre ce problème nous utilisons AbSolute [14] un solveur de contraintes continues basé sur les domaines abstraits. Pour des questions d'efficacité, nous avons ajouté dans le solveur une phase de préprocessing durant laquelle, une réécriture symbolique des contraintes est effectuée.

4.1 Réécriture symbolique

Similairement à CPLEX, AbSolute effectue une étape de préprocessing. Cette étape peut permettre de simplifier les contraintes et réduire la taille du problème en éliminant des variables par réécriture. Durant cette étape, seules les contraintes linéaires d'égalité sont considérées. Dans la suite de cette section, le terme égalité désigne une contrainte linéaire d'égalité.

Dans un premier temps, nous identifions toutes les constantes et les conservons dans une liste. Une constante correspond à une égalité unaire. Les constantes sont remplacées dans tout le CSP par leurs valeurs, et ce processus est répété jusqu'à atteindre un point fixe.

Dans un second temps, les égalités binaires sont supprimées du problème et conservées dans une liste comme des vues de variables [16]. L'une des variables est choisie et remplacée dans tout le CSP par sa réécriture, et des contraintes sur les bornes sont ajoutées. Ce processus est lui aussi répété jusqu'à atteindre un point fixe.

Exemple 1 *Considérons le CSP avec* $V = \{v_1, v_2, v_3\}$ *les variables continues prenant leur valeurs dans* $D = \{D_1 = D_2 = D_3 = [-5, 5]\}$*, et* $C = \{c_1 : v_1 = 2, c_2 : v_1 = v_2 + v_3, c_3 : v_3 = 3 \times (v_2)^2\}$*. On peut voir que* v_1 *est une constante, on la remplace donc partout par sa valeur. La deuxième contrainte devient une vue, on peut donc remplacer* v_2 *par* $2 - v_3$*. On obtient donc le CSP suivant :* $V' = \{v_3\}$*,* $D' = \{D_3 = [-5, 5]\}$ *et* $C' = \{c_1 : v_3 = 3 \times (2 - v_3)^2, c_2 : 2 - v_3 \geqslant -5, c_3 : 2 - v_3 \leqslant 5\}$*. Sur ce petit exemple, on est passé d'un CSP à trois variables, à un CSP avec une constante, une vue et une variable.*

Reprenons l'exemple donné section 3.3, le CSP associé (annexe 7.2) contient 26 variables et 38 contraintes, après l'étape de réécriture, le CSP contient 16 constantes, 5 vues, 5 variables et 26 contraintes (dont 10 pour les bornes des vues).

Cette étape est transparente pour l'utilisateur, et si elles existent, les solutions sont données pour les variables initiales du problème.

4.2 Représentation d'une solution

Dans le continu, une solution est généralement un produit d'intervalles appelé boîte. En utilisant la notion de contracteurs [6], une solution est soit une boîte ne contenant que des solutions du CSP, soit une boîte assez petite (pour un paramètre de précision donné) pouvant contenir une solution du CSP. La figure 4 montre un exemple de solutions d'un problème, les boîtes vertes ne contiennent que des solutions et les boîtes roses peuvent en contenir.

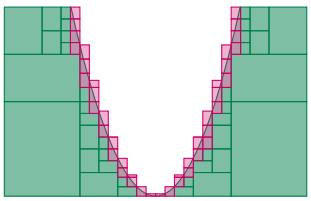


FIGURE 4 – Comparaison entre les solutions sûres et non sûres.

AbSolute peut renvoyer, si elles existent, deux types de solutions : les solutions sûres, ne contenant que des solutions du problème, et les solutions non sûres pouvant en contenir une.

Toute solution est représentée par deux listes. La première contient le domaine des variables sûres (quelque soit la valeur prise dans ce domaine, il existe une solution). Par abus de langage, on parlera de liste d'*invariants*. La seconde liste, quant à elle, contient les domaines pouvant contenir une solution. Par opposition, on parlera de liste de *changeants*.

Une solution sûre aura une liste de changeants vide et toutes les variables seront dans la liste des invariants. Une solution non sûre aura quant à elle au moins un élément dans la liste des changeants.

Lorsqu'une solution du problème est trouvée, les inconnues sont substituées par leur valeur dans les vues. Ces nouvelles affectations sont ajoutées à la liste des invariants ou des changeants suivant que l'inconnue était invariante ou changeante.

5 Résultats expérimentaux

5.1 Démarche générale

L'utilisation de la logique de Hoare pour contraindre les paramètres du modèle que l'on cherche à construire vise à l'automatisation de l'identification des paramètres et à la minimisation de l'intervention du modélisateur dans la construction du modèle. Les données d'entrée sont d'une part le réseau de gènes, et d'autre part une postcondition et une trace expérimentale exprimée dans le langage de chemins. Trois principales étapes sont à distinguer : construction des contraintes minimales portant sur les paramètres dynamiques du réseau de gènes, identification d'un jeu de paramètres satisfaisant ces contraintes, et simulation du modèle qui pourra ensuite être comparée à la trace biologique expérimentale.

La première étape consiste à utiliser le calcul de la plus faible précondition à partir de la postcondition donnée en remontant le chemin décrivant la trace biologique observée, voir sous-section 3.2. On se permet, pour les modèles à comportement cyclique, de rajouter des contraintes supplémentaires indiquant que l'état hybride initial du chemin est identique à l'état hybride final. Ces contraintes sont simplifiées et réduites à la volée.

L'étape suivante fait appel au solveur de contraintes AbSolute qui aide à l'identification d'un jeu de paramètres satisfaisant les contraintes. Une solution donnée par le solveur AbSolute correspond à la donnée d'une liste d'invariants et d'une liste de changeants. Si la liste des changeants est vide, la solution est dite sûre et un jeu de paramètres peut simplement être choisi, variable par variable. Dans la plupart des cas, plusieurs solutions sont exhibées, elles peuvent être sûres ou non.

L'utilisateur spécifie un nombre souhaité de solutions et le solveur en exhibe autant lorsque c'est possible. S'il n'y a pas de solutions sûres, on choisit une solution non sûre au hasard, une variable parmi les changeants est sélectionnée aléatoirement, on lui affecte une valeur arbitraire dans son domaine de définition, et cette variable devient constante (dans la liste des invariants). Cette contrainte est ensuite ajoutée au CSP, et le solveur est de nouveau appelé pour calculer les nouvelles solutions de ce nouveau système de contraintes. Ce procédé est répété jusqu'à épuiser la liste des changeants. Lorsqu'une solution sûre existe et que l'un de ses invariants présente un domaine de définition dont les bornes sont distinctes, sa valeur est choisie aléatoirement dans son intervalle de définition.

Enfin, la dernière étape consiste à réaliser une simulation de la dynamique du réseau de gènes.

5.2 Résultats

Pour le modèle de la figure 1, à partir du chemin et de la postcondition de la sous-section 3.3, la démarche exposée précédemment mène à un jeu de paramètres que nous avons utilisé pour simuler l'évolution du modèle, voir figure 5. De fait, une solution sûre est immédiatement donnée par AbSolute.
Toute la chaîne de traitement, allant de la construction des contraintes à la simulation est exécutée en 656 millisecondes.

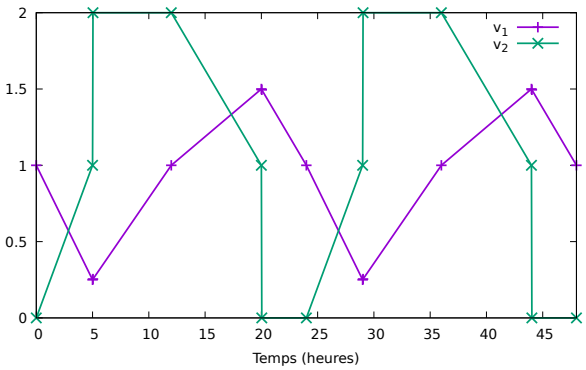


FIGURE 5 – Simulation obtenue sur 48 heures en comportement cyclique. La courbe associée à chacune des entités v_i correspond à l'évolution de la somme du niveau qualitatif discret et de la partie fractionnaire $(\eta(v_i) + \pi(v_i))$.

Les 4 chemins élémentaires du chemin de la sous-section 3.3 sont facilement identifiables dans la fi-

gure 5 : l'incrémentation du niveau qualitatif de v_2 (resp. v_1) observé en ordonnée a lieu à 5.0 heures (resp. 12.0 heures), et sa décrémentation à 20.0 heures (resp. 24.0 heures). Les phénomènes de saturation $\mathsf{slide}^+(v_2)$ et de dégradation totale $\mathsf{slide}^-(v_2)$ sont bien apparents avant le changement de niveau qualitatif v_1+ et v_1- respectivement. Enfin, l'entité v_1 ne glisse jamais, ce qui est dû à la prise en compte de nos contraintes $\mathsf{noslide}(v_1)$.

Pour la simulation de la figure 5, nous avons supposé que le système biologique présente un comportement cyclique de 24 heures et cette contrainte a été rajoutée au CSP. On peut voir dans la simulation deux cycles de période de 24 heures. Néanmoins, à chaque période, la phase est décalée de 0.53 secondes et ce décalage est dû à l'approximation des réels dans le solveur. Cependant, cette simulation est robuste puisque le temps nécessaire pour voir la phase se décaler d'une période dépasse 18 ans. Cette précision est largement suffisante pour les problèmes abordés en biologie.

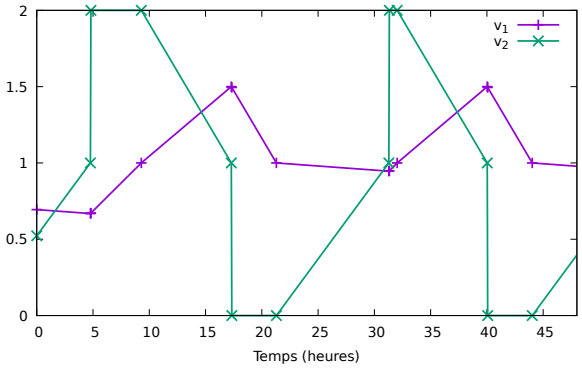


FIGURE 6 – Simulation obtenue sur 48 heures sans imposer un comportement cyclique.

Lorsqu'on n'impose pas de comportement cyclique, la démarche utilisée a du mal à identifier précisément le point initial de la simulation : les variables du CSP correspondant au début de la simulation appartiennent à la liste des changeants. Pour obtenir la figure 6, nous avons choisi au hasard des valeurs pour les parties fractionnaires du point initial, ce qui peut impacter considérablement l'évolution des entités. Néanmoins, les simulations partant d'initialisations différentes mènent après un certain temps à un comportement cyclique. En effet, il a été montré que tout système construit sur un graphe d'interaction de deux variables et dont la dynamique contient un chemin glissant sur un mur externe possède un comportement cyclique appelé *cycle limite* vers lequel toutes les traces convergent [8].

Intuitivement, la figure 7 montre que, lorsqu'on démarre de l'état hybride h_3, le bord supérieur de v_2 est atteint et franchi, puis une nouvelle transition continue

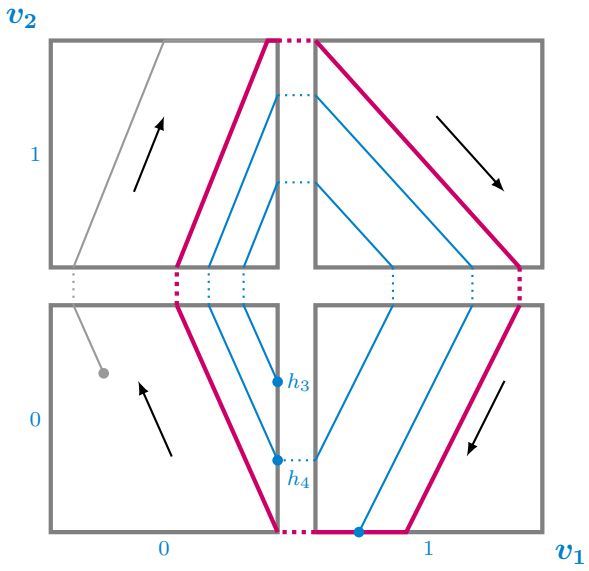


FIGURE 7 – Convergence de trajectoires vers un cycle limite possible associé au graphe de la figure 1.Le cycle limite est représenté en rose, et les traces grises et bleues montrent la convergence des traces vers le cycle limite.

est observée dans l'état discret $(0, 1)$. Cette transition continue parvient au bord supérieur de v_1 et l'alternance entre transitions continues et discrètes se poursuit jusqu'à atteindre l'état hybride h_4. La trajectoire s'est rapprochée du cycle limite en rose, et après plusieurs passages par l'état discret $(0, 0)$, le cycle limite est atteint.

6 Conclusion et Perspectives

Dans cet article, nous avons couplé le calcul de la plus faible précondition de la logique de Hoare avec le solveur de contraintes AbSolute. Cette approche nous a permis de générer des jeux de paramètres sur des modèles hybrides de réseaux de gènes menant à une dynamique exhibant la trace biologique observée. L'outil développé est très prometteur du point de vue de l'application biologique puisqu'il permet de générer sur nos premiers exemples des comportements similaires aux observations biologiques. Au cours de nos développements, nous avons été confronté à la taille des formules construites. L'explosion combinatoire nous a incité à trouver une stratégie de simplification et de réduction à la volée des contraintes du CSP. De plus, la prise en compte de relations symboliques entre les variables dans le solveur nous a permis de diminuer le nombre de variables à identifier et donc limiter les problèmes d'incertitude entre variables dépendantes dus

à la représentation des nombres réels.

Le temps d'exécution du solveur AbSolute dépend d'une part de la précision sur les intervalles des variables du CSP, et d'autre part du nombre de solutions que l'utilisateur souhaite. Pour aborder des problèmes de plus grandes tailles, notamment les systèmes biologiques complexes, il sera nécessaire de trouver le meilleur compromis entre diversité des solutions et temps de calcul pour obtenir des résultats satisfaisants. Enfin, notre objectif actuel avec retombée thérapeutique est de modéliser le couplage du cycle circadien (alternance sommeil/éveil) avec le cycle cellulaire (division des cellules) pour aborder des questions de chronothérapie du cancer (optimisation de l'heure d'administration des médicaments). Toutefois, le modèle du cycle cellulaire (5 entités biologiques abstraites) mène à un système de contraintes contenant 178 variables continues, 65 variables discrètes, 267 contraintes dont certaines disjonctions ce qui augmente considérablement la combinatoire. Le couplage de ces deux cycles mènera à un système de contraintes qui nécessitera de revoir notre stratégie de recherche de solutions. Notamment, une amélioration du système de réécriture des contraintes continues devrait permettre une résolution plus rapide.

Références

[1] R. Alur. Formal verification of hybrid systems. In *2011 Proceedings of the Ninth ACM International Conference on Embedded Software (EMSOFT)*, pages 273–278, Oct 2011.

[2] J. Behaegel, J.-P. Comet, and M. Folschette. Constraint identification using modified Hoare logic on hybrid models of gene networks. In *Proceedings of the 24th International Symposium on Temporal Representation and Reasoning (TIME)*, pages 5 :1–5 :21, October 2017.

[3] G. Bernot, J.-P. Comet, Z. Khalis, A. Richard, and O. F. Roux. A genetically modified Hoare logic. *Theoretical Computer Science*, 2018. https ://doi.org/10.1016/j.tcs.2018.02.003.

[4] G. Bernot, J.-P. Comet, A. Richard, and J. Guespin. Application of formal methods to biological regulatory networks : extending Thomas' asynchronous logical approach with temporal logic. *Journal of theoretical biology*, 229(3) :339–347, 2004.

[5] A. Bockmayr and A. Courtois. Using hybrid concurrent constraint programming to model dynamic biological systems. In *International Conference on Logic Programming*, pages 85–99. Springer, 2002.

[6] G. Chabert and L. Jaulin. Contractor programming. *Artificial Intelligence*, 173 :1079–1100, 2009.

[7] F. Corblin, E. Fanchon, and L. Trilling. Applications of a formal approach to decipher discrete genetic networks. *BMC Bioinformatics*, 11 :385, 2010.

[8] E. Cornillon. *Modèles qualitatifs de réseaux génétiques : réduction de modèles et introduction d'un temps continu.* PhD thesis, Université Côte d'Azur, 2017.

[9] E. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Commun. ACM*, 18 :453–457, August 1975.

[10] L. F. Fitime, O. F. Roux, C. Guziolowski, and L. Paulevé. Identification of bifurcation transitions in biological regulatory networks using answer-set programming. *Algorithms for Molecular Biology*, 12(1) :19 :1–19 :14, 2017.

[11] A. Girard, A. A. Julius, and G. J. Pappas. Approximate simulation relations for hybrid systems. *Discrete Event Dynamic Systems*, 18(2) :163–179, 2008.

[12] T. A. Henzinger. *The Theory of Hybrid Automata*, pages 265–292. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

[13] M. Martin, P. Dague, S. Pérès, and L. Simon. Minimality of metabolic flux modes under boolean regulation constraints. In *12th International Workshop on Constraint-Based Methods for Bioinformatics*, 2016.

[14] M. Pelleau, A. Miné, C. Truchet, and F. Benhamou. A constraint solver based on abstract domains. In *International Conference on Verification, Model Checking, and Abstract Interpretation*, pages 434–454. Springer, 2013.

[15] G. Rodrigo, J. Carrera, and A. Jaramillo. Combinatorial optimisation to design gene regulatory networks. In *4th International Workshop on Constraint-Based Methods for Bioinformatics*, 2008.

[16] C. Schulte and G. Tack. View-based propagator derivation - (extended abstract). In *20th International Conference on Principles and Practice of Constraint Programming*, pages 938–942, 2014.

[17] R. Thomas. Boolean formalization of genetic control circuits. *Journal of theoretical biology*, 42(3) :563–585, 1973.

7 Annexe

7.1 Sous-propriétés

Nous détaillons ici chacune des sous-propriétés de la plus faible précondition, voir définition 3.

7.1.1 Ressource d'une entité

Pour calculer la plus faible précondition, nous étiquetons les parties fractionnaires des états du système. Nous utilisons les étiquettes i (initial) et f (final). De plus, nous utilisons π' (resp. π) pour spécifier la partie fractionnaire de sortie (resp. d'entrée) de l'état discret courant.

Toutes les propriétés suivantes dépendent des indices i et f utilisés dans la définition 3. Pour alléger les notations, les parties fractionnaires π_v^i et $\pi_v^{i'}$ seront notées simplement π_v et $\pi_v^{'}$. Enfin, pour toute entité $v \in V$ et pour tout sous-ensemble prédécesseur de v $\omega \subset R^-(v)$, nous définissons :

$$\Phi_v^\omega \equiv \Big(\bigwedge_{m\in\omega} \varphi_m\Big) \wedge \Big(\bigwedge_{m\in R^{-1}(v)\setminus\omega} \neg\varphi_m\Big) \ .$$

En d'autres termes, Φ_v^ω est vraie dans un état h si et seulement si les ressources de v sont exactement ω.

7.1.2 Transition discrète

Pour l'entité du *dpa* $v \in V$, $\Phi_v^+(\Delta t)$ (resp. $\Phi_v^-(\Delta t)$) donne les conditions pour que v incrémente (resp. décrémente) son niveau d'expression discret après Δt unités de temps : sa célérité dans l'état courant est positive (resp. négative) et sa partie fractionnaire d'entrée dépend de sa partie fractionnaire de sortie, de Δt et de $C_{v,\rho(h,v),n}$.

$$\begin{aligned}\Phi_v^+(\Delta t) \equiv\ & (\pi_v^{'} = 1)\\ & \wedge \bigwedge_{\substack{\omega\subset R^-(v)\\ n\in[\![0,b_v]\!]}} \begin{pmatrix} \big(\Phi_v^\omega \wedge (\eta_v = n)\big) \Rightarrow (C_{v,\omega,n} > 0)\\ \wedge(\pi_v = \pi_v^{'} - C_{v,\omega,n}\cdot\Delta t)\end{pmatrix}\\ \Phi_v^-(\Delta t) \equiv\ & (\pi_v^{'} = 0)\\ & \wedge \bigwedge_{\substack{\omega\subset R^-(v)\\ n\in[\![0,b_v]\!]}} \begin{pmatrix} \big(\Phi_v^\omega \wedge (\eta_v = n)\big) \Rightarrow (C_{v,\omega,n} < 0)\\ \wedge(\pi_v = \pi_v^{'} - C_{v,\omega,n}\cdot\Delta t)\end{pmatrix}\end{aligned}$$

7.1.3 Murs internes et externes

Pour toute entité $u \in V$, $\mathcal{W}_u^+$ (resp. $\mathcal{W}_u^-$) empêche u d'incrémenter (resp. décrémenter) son niveau qualitatif. Ce mur peut être externe EW_u^+ (resp. EW_u^-) ou interne IW_u^+ (resp. IW_u^-). De plus, $\Phi_{u+}^{\omega'}$ (resp. $\Phi_{u-}^{\omega'}$), requis dans ces sous-formules, est vraie si et seulement si les ressources de u sont exactement ω' dans l'état où le niveau qualitatif de u a été incrémenté (resp. décrémenté) de 1.

$$\mathcal{W}_u^+ \equiv \mathsf{IW}_u^+ \vee \mathsf{EW}_u^+ \qquad \text{et} \qquad \mathcal{W}_u^- \equiv \mathsf{IW}_u^- \vee \mathsf{EW}_u^-$$

avec :

$$\mathsf{EW}_u^+ \equiv (\eta_u = b_u) \wedge \bigwedge_{\omega\subset R^-(u)} (\Phi_u^\omega \Rightarrow C_{u,\omega,b_u} > 0)$$

$$\mathsf{EW}_u^- \equiv (\eta_u = 0) \wedge \bigwedge_{\omega\subset R^-(u)} (\Phi_u^\omega \Rightarrow C_{u,\omega,0} < 0)$$

$$\mathsf{IW}_u^+ \equiv (\eta_u < b_u)\wedge \bigwedge_{\substack{\omega,\omega'\subset R^-(u)\\ n\in[\![0,b_u-1]\!]}} \left(\begin{pmatrix}(\eta_u = n)\wedge\\ (m = n+1)\wedge\\ \Phi_u^\omega \wedge \Phi_{u+}^{\omega'}\end{pmatrix} \Rightarrow \begin{pmatrix} C_{u,\omega,n} > 0\wedge\\ C_{u,\omega',m} < 0\end{pmatrix}\right)$$

$$\mathsf{IW}_u^- \equiv (\eta_u > 0)\wedge \bigwedge_{\substack{\omega,\omega'\subset R^-(u)\\ n\in[\![1,b_u]\!]}} \left(\begin{pmatrix}(\eta_u = n)\wedge\\ (m = n-1)\wedge\\ \Phi_u^\omega \wedge \Phi_{u-}^{\omega'}\end{pmatrix} \Rightarrow \begin{pmatrix} C_{u,\omega,n} < 0\wedge\\ C_{u,\omega',m} > 0\end{pmatrix}\right)$$

$$\Phi_{u+}^{\omega'} \equiv (\eta_u < b_u) \wedge \bigwedge_{n\in[\![0,b_u-1]\!]} \big((\eta_u = n) \Rightarrow \Phi_u^{\omega'}[\eta_u\backslash\eta_u + 1]\big)$$

$$\Phi_{u-}^{\omega'} \equiv (\eta_u > 0) \wedge \bigwedge_{n\in[\![1,b_u]\!]} \big((\eta_u = n) \Rightarrow \Phi_u^{\omega'}[\eta_u\backslash\eta_u - 1]\big)$$

7.1.4 Entités de *first*

$\mathcal{F}(\Delta t)$ contraint les entités qui ne sont pas dans *first* à atteindre leur bord après les entités de *first*, ou à faire face à un mur interne ou externe.

$$\begin{aligned}\mathcal{F}(\Delta t) \equiv \bigwedge_{u\in V\setminus first(h)} & \left(\bigwedge_{\substack{\omega\subset R^-(u)\\ n\in[\![0,b_u]\!]}} \begin{pmatrix}(\eta_u = n) \wedge \Phi_u^\omega\wedge\\ C_{u,\omega,n} > 0\wedge\\ \pi_u > \pi'_u - C_{u,\omega,n}\cdot\Delta t\end{pmatrix} \Rightarrow \mathcal{W}_u^+\right)\\ & \wedge \left(\bigwedge_{\substack{\omega\subset R^-(u)\\ n\in[\![0,b_u]\!]}} \begin{pmatrix}(\eta_u = n) \wedge \Phi_u^\omega\wedge\\ C_{u,\omega,n} < 0\wedge\\ \pi_u < \pi'_u - C_{u,\omega,n}\cdot\Delta t\end{pmatrix} \Rightarrow \mathcal{W}_u^-\right)\end{aligned}$$

7.1.5 Assertions hybrides

La sous-propriété $\mathcal{A}(\Delta t, a)$ traduit les symboles d'assertion de la transition continue en contraintes :

$$\mathcal{A}(\Delta t, a) \equiv \bigwedge_{\substack{k \in [\![1,n]\!] \\ \omega_k \in R^-(v_k) \\ n_k \in [\![0, b_{v_k}]\!]}} \left(\bigwedge_{l \in [\![1,n]\!]} \left((\eta_{v_l} = n_l) \wedge \Phi^{\omega_l}_{v_l} \right) \Rightarrow a \begin{bmatrix} C_{v_l} \backslash C_{v_l,\omega_l,n_l} \\ \mathsf{slide}(v_l) \backslash \mathcal{S}_{v_l,\omega_l,n_l}(\Delta t) \\ \mathsf{slide}^{\pm}(v_l) \backslash \mathcal{S}^{\pm}_{v_l,\omega_l,n_l}(\Delta t) \\ \mathsf{noslide}(v_l) \backslash \mathsf{no}\mathcal{S}_{v_l,\omega_l,n_l}(\Delta t) \\ \mathsf{noslide}^{\pm}(v_l) \backslash \mathsf{no}\mathcal{S}^{\pm}_{v_l,\omega_l,n_l}(\Delta t) \end{bmatrix} \right)$$

avec a l'assertion du chemin élémentaire $(\Delta t, a, v\pm)$, et, pour toute entité $u \in V$:

$$\begin{aligned}
\mathcal{S}^{+}_{u,\omega,n}(\Delta t) &\equiv (\pi'_u = 1) \wedge (\pi_u > \pi'_u - C_{u,\omega,n} \cdot \Delta t) \\
\mathcal{S}^{-}_{u,\omega,n}(\Delta t) &\equiv (\pi'_u = 0) \wedge (\pi_u < \pi'_u - C_{u,\omega,n} \cdot \Delta t) \\
\mathcal{S}_{u,\omega,n}(\Delta t) &\equiv \mathcal{S}^{+}_{u,\omega,n}(\Delta t) \vee \mathcal{S}^{-}_{u,\omega,n}(\Delta t) \\
\mathsf{no}\mathcal{S}^{+}_{u,\omega,n}(\Delta t) &\equiv (\pi'_u < 1) \wedge \\
&\qquad (C_{u,\omega,n} > 0 \Rightarrow \Delta t \leqslant \frac{\pi'_u - \pi_u}{C_{u,\omega,n}}) \\
\mathsf{no}\mathcal{S}^{-}_{u,\omega,n}(\Delta t) &\equiv (\pi'_u > 0) \wedge \\
&\qquad (C_{u,\omega,n} < 0 \Rightarrow \Delta t \leqslant \frac{\pi'_u - \pi_u}{C_{u,\omega,n}}) \\
\mathsf{no}\mathcal{S}_{u,\omega,n}(\Delta t) &\equiv \mathsf{no}\mathcal{S}^{+}_{u,\omega,n}(\Delta t) \wedge \mathsf{no}\mathcal{S}^{-}_{u,\omega,n}(\Delta t)
\end{aligned}$$

Les trois premières sous-formules indiquent que la position de sortie de l'entité u est située sur un bord. De plus, les contraintes $\pi^i_u > \pi^{i'}_u - C_{u,\omega,n} \cdot \Delta t$ et $\pi^i_u < \pi^{i'}_u - C_{u,\omega,n} \cdot \Delta t$ indiquent que la durée avant d'atteindre le bord de u est inférieure au temps passé dans l'état courant. Le signe de la célérité de l'entité glissante u est contrainte par la sous-propriété $\mathcal{F}$ et par la contrainte $\pi_u > \pi^{'}_u - C_{u,\omega,n} \cdot \Delta t$ (resp. $\pi_u < \pi^{'}_u - C_{u,\omega,n} \cdot \Delta t$) de la sous-propriété $\mathcal{S}^{+}_{u,\omega,n}(\Delta t)$ (resp. $\mathcal{S}^{-}_{u,\omega,n}(\Delta t)$) ou $\mathcal{S}_{u,\omega,n}(\Delta t)$.

Les trois dernières sous-formules indiquent que la position de sortie de l'entité u ne peut pas être sur un bord. Lorsque la célérité de l'entité u est positive (resp. négative) dans le cas de $\mathsf{no}\mathcal{S}^{+}_{u,\omega,n}(\Delta t)$ (resp. $\mathsf{no}\mathcal{S}^{-}_{u,\omega,n}(\Delta t)$), le temps passé dans l'état est inférieur ou égale au délai de contact de u ($\Delta t \leqslant \frac{\pi'_u - \pi_u}{C_{u,\omega,n}}$).

7.1.6 Jonctions

Jonctions continues Pour toute entité $v \in V$, et pour une transition continue entre deux états hybrides $h = (\eta, \pi)$ et $h' = (\eta, \pi')$, $\mathcal{CJ}_v$ établit une relation entre les parties fractionnaires et la célérité de l'entité v. Si la partie fractionnaire de sortie de v est 0 ou 1, le signe de la célérité peut être déduit et le temps nécessaire à v pour atteindre son bord est plus faible que le temps passé dans l'état discret courant. Si v n'atteint pas son bord, la position exacte de la partie fractionnaire d'entrée de v peut être déduite de la position de sortie, du temps passé dans l'état discret courant et de la célérité.

$$\mathcal{CJ}_v \equiv \begin{cases} (0 < \pi'_v < 1) \Rightarrow (\pi_v = \pi'_v - C_{v,\rho(h,v),\eta_v}.\delta^{first}_h) \\ \wedge (\pi'_v = 0) \Rightarrow C_{v,\rho(h,v),\eta_v} < 0 \wedge \\ \qquad\qquad (\pi_v \leqslant \pi'_v - C_{v,\rho(h,v),\eta_v}.\delta^{first}_h) \\ \wedge (\pi'_v = 1) \Rightarrow C_{v,\rho(h,v),\eta_v} > 0 \wedge \\ \qquad\qquad (\pi_v \geqslant \pi'_v - C_{v,\rho(h,v),\eta_v}.\delta^{first}_h) \end{cases}$$

Jonctions discrètes Pour toute entité $v \in V$, et pour une transition discrète modifiant le niveau qualitatif de v entre un état initial et un état final correspondant aux indices i et f, $\mathcal{DJ}_v$ établit une jonction entre les parties fractionnaires de ces états. Cette formule montre que la partie fractionnaire de v change de 1 à 0 (resp. 0 à 1) lors d'une incrémentation (resp. décrémentation) du niveau qualitatif de v tandis que les parties fractionnaires des autres entités restent inchangées :

$$\mathcal{DJ}_v \equiv (\pi^f_v = 1 - \pi^{i'}_v) \wedge \bigwedge_{u \in V \backslash \{v\}} (\pi^f_u = \pi^{i'}_u) \ .$$

Finalement, nous définissons :

$$\mathcal{J}_v \equiv \mathcal{DJ}_v \wedge \bigwedge_{u \in V} \mathcal{CJ}_u$$

Ces relations sont facilement interprétables sur la figure 3 : toutes les parties fractionnaires restent les mêmes entre les deux états discrets sauf la partie fractionnaire de l'entité assurant la transition.

7.2 Contraintes du système

Après simplification et réduction des contraintes construites sur l'exemple donné en section 3.3, en y ajoutant un comportement cyclique, nous obtenons 6 valeurs exactes sur des parties fractionnaires, ainsi que les 32 contraintes suivantes :

$(\pi^{3'}_{v_1} = 0.) \wedge (\pi^{3'}_{v_2} = 0.) \wedge (\pi^{2'}_{v_2} = 0.) \wedge (\pi^{1'}_{v_1} = 1.) \wedge (\pi^{1'}_{v_2} = 1.) \wedge (\pi^{0'}_{v_2} = 1.) \wedge (\pi^{4}_{v_1} = \pi^{0}_{v_1}) \wedge (\pi^{4}_{v_2} = \pi^{0}_{v_2}) \wedge (C_{v_1,\varnothing,0} \times C_{v_1,\varnothing,1} \geqslant 0.) \wedge (C_{v_1,\{m2\},0} \times C_{v_1,\{m2\},1} \geqslant 0.) \wedge (C_{v_2,\varnothing,0} \times C_{v_2,\varnothing,1} \geqslant 0.) \wedge (C_{v_2,\{m1\},0} \times C_{v_2,\{m1\},1} \geqslant 0.) \wedge (\pi^{3'}_{v_1} = 1. - \pi^{4}_{v_1}) \wedge (\pi^{3'}_{v_2} = \pi^{4}_{v_2}) \wedge (\pi^{2'}_{v_1} < 1.) \wedge (\pi^{2'}_{v_1} > 0.) \wedge (\pi^{2'}_{v_2} = 1. - \pi^{3}_{v_2}) \wedge (\pi^{2'}_{v_1} = \pi^{3}_{v_1}) \wedge (\pi^{1'}_{v_1} = 1. - \pi^{2}_{v_1}) \wedge (\pi^{1'}_{v_2} = \pi^{2}_{v_2}) \wedge (\pi^{0'}_{v_1} < 1.) \wedge (\pi^{0'}_{v_1} > 0.) \wedge (\pi^{0'}_{v_2} = 1. - \pi^{1}_{v_2}) \wedge (\pi^{0'}_{v_1} = \pi^{1}_{v_1}) \wedge (\pi^{0}_{v_2} = \pi^{0'}_{v_2} - C_{v_2,\{m1\},0} \times 5.) \wedge (C_{v_2,\{m1\},0} > 0.) \wedge (\pi^{0}_{v_1} = \pi^{0'}_{v_1} - C_{v_1,\varnothing,0} \times 5.) \wedge (C_{v_2,\{m1\},1} > 0.) \wedge (\pi^{1}_{v_1} =$

$\pi^{1'}_{v_1} - C_{v_1,\{m2\},0} \times 7.) \wedge (C_{v_1,\{m2\},0} > 0.) \wedge (\pi^{2}_{v_2} = \pi^{2'}_{v_2} - C_{v_2,\varnothing,1} \times 8.) \wedge (C_{v_2,\varnothing,1} < 0.) \wedge (\pi^{2}_{v_1} = \pi^{2'}_{v_1} - C_{v_1,\{m2\},1} \times 8.) \wedge (C_{v_2,\varnothing,0} < 0.) \wedge (\pi^{3}_{v_1} = \pi^{3'}_{v_1} - C_{v_1,\varnothing,1} \times 4.) \wedge (C_{v_1,\varnothing,1} < 0.) \wedge (\pi^{1}_{v_2} > \pi^{1'}_{v_2} - C_{v_2,\{m1\},1} \times 7.) \wedge (\pi^{3}_{v_2} < \pi^{3'}_{v_2} - C_{v_2,\varnothing,0} \times 4.)$; où m_1 représente le multiplexe actif relatif à la formule $\neg(v_1 \geqslant 1)$ et m_2 celui relié à la formule $v_2 \geqslant 1$.

Différenciation des réponses des malades aux traitements, en utilisant le Answer Set Programming et les données protéomiques

Lokmane Chebouba[1] * **Dalila Boughaci**[1] **Carito Guziolowski**[2]

[1] LRIA Laboratory, University of Science and Technology Houari Boumediene, Algiers, Algeria
[2] LS2N, UMR 6004, Ecole Centrale de Nantes, Nantes, France
lchebouba@gmail.com dalila_info@yahoo.fr carito.guziolowski@ls2n.fr

Résumé

Plusieurs approches ont été appliquées sur les données omiques et cliniques issues de patients pour la prédiction des réponses aux traitements ainsi que pour la détection des protéines et des gènes spécifiques aux maladies, et ce pour le ciblage des traitements. Ces approches utilisent en grande partie les données cliniques. Dans notre travail récemment publié [2], nous avons proposé une nouvelle méthode pour la discrimination de la réponse aux traitements des malades atteints par la leucémie myéloïde aiguë en utilisant les données protéomiques pour la construction de modèles booléens permettant la prédiction des réponses aux traitements. Pour évaluer notre méthode, nous avons utilisé les données fournies par DREAM Challenge 9 [1]. Les résultats obtenus montrent l'apport de notre approche qui a été la seule méthode, parmi les méthodes proposés pour le DREAM 9, capable de produire un modèle mécaniste (Booléen) pour différentier les differents types de patients.

Abstract

Several approaches were applied on omic and clinical data for the detection of disease-specific proteins and genes, key candidates for drug targetting. Most of the proposed appraches used mainly clinical data. This work proposes a new method for the discrimination of the treatment response of acute myeloid leukemia (AML) patients by using proteomics data for the prediction of treatment responses . To evaluate our method, we used the data provided by DREAM Challenge 9. The results obtained are encouraging and demonstrate the benefits of our approach.

*Papier doctorant : Lokmane Chebouba[1] est auteur principal.

1. Ces données sont fournies par Dr Steven Kornblau du MD Anderson Cancer Center de l'Université du Texas. Elles sont obtenues par Synapse syn2455683 dans le cadre du Challenge AML DREAM.

1 Introduction

Seul le quart des malades diagnostiqués avec la leucémie aiguë survive au-delà de 5 ans, d'où le besoin urgent d'explorer comment les modèles mathématiques peuvent contribuer à l'avancée des traitements personnalisés. Dans ce contexte, la prédiction d'un modèle mécaniste expliquant la réponse des patients aux traitements basée sur les données protéomiques peut améliorer les décisions cliniques. En 2014, DREAM 9 challenge [2] a lancé un défi pour prédire la réponse (rémission complète (CR) et la résistance primaire (PR)) de 191 patients atteints par cette leucémie à partir de leurs données protéomiques (231 protéines mesurées) et 40 données cliniques. Dans ce travail, nous étudions l'impact de l'utilisation d'un modèle mathématique construit à partir d'un réseau de signalisation, associé avec les protéines mesurées pour la prédiction des réponses des malades au traitement.

2 Méthode

Notre méthode est composée de 4 étapes principales (voir Figure 1).

2.1 Création du réseau de connaissances à priori

A partir des bases de données publiques, qui permettent de connecter les 231 protéines mesurées.

2. https ://www.synapse.org/#!Synapse :syn2455683. accès : 17 mars 2018

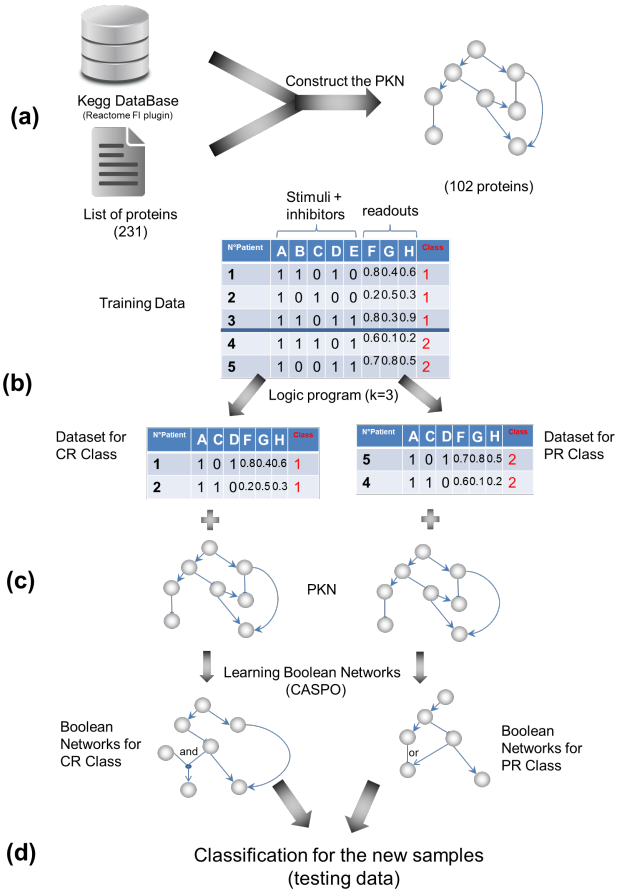


FIGURE 1 – Workflow de notre méthode.

2.2 Sélection des protéines et des patients

Nous proposons un programme logique basé sur l'Answer Set Programming (ASP) [1] pour la sélection des patients et de k protéines qui maximisent le nombre de pairs de patients pour lesquelles la valeur binaire de leurs mesures expérimentales sont égaux dans les deux classes (CR, PR).

2.3 Apprentissage

Pour cela nous utilisons l'outil CASPO [3] [5], basé aussi sur l'ASP. Cela produit deux familles de réseaux booléens (CR et PR). Notre objectif est d'apprendre les deux différentes familles des réseaux booléens ayant les mêmes noeuds de type *stimuli* et *inhibiteur*, et montrant un câblage différent pour chaque classe vers des noeuds de type *sortie* du système biologique.

3. http ://caspo.readthedocs.io/en/latest/ accès : 30 avril 2018

2.4 Classification

Nous utilisons la moyenne des erreurs au carré (MSE) entre les sorties mesurées et les sorties prédites pour chaque patient de tests, en se basant sur les deux familles de réseaux booléens. Un patient est classé dans la classe avec le MSE le plus petit.

3 Résultats

3.1 Réseau de connaissances à priori

Nous avons construit un réseau en utilisant la base de données KEGG[3]. Le réseau est composé de 102 nœuds (17 stimuli, 62 inhibiteurs et 23 sorties) connectés par 294 arcs.

3.2 Sélection des protéines et des patients

Le résultat de cette étape est un sous ensemble de k protéines présentes dans l'ensemble des stimuli et inhibiteurs. Nous avons réduit le nombre de protéines à k=10 (4 stimuli et 6 inhibiteurs).

3.3 Réseau booléen appris

Nous avons appris deux familles de réseaux booléens (CR vs PR) moyennant l'outil CASPO, en fournissant en entrée le réseau de connaissances à priori, et les deux ensembles de données réduits. Nous avons trouvé deux familles de réseaux booléens qui sont différentes et expliquent différents comportements.

4 Conclusion

Nous avons proposé une méthode pour la discrimination de la réponse aux traitements des malades atteints par la leucémie myéloïde aiguë (AML) basée sur l'ASP et les données protéomiques. Les résultats trouvés sont intéressants, la méthode donne une précision de 58.75% qui est inférieure à celle du gagnant du DREAM 9 [4] mais elle permet de détecter les meilleures protéines, pour construire des modèles mécanistes optimaux differentiant deux classes de patients. La méthode proposée est conçue en ASP pour la selection des attributs. Nous avons aussi essayé de restreindre le domaine de protéines en utilisant des méthodes de sélection d'attributs classiques (*network clustering* et PCA), mais nous avons trouvé des jeux de données avec moins de patients. L'avantage de proposer un code entièrement en ASP, et du fait que CASPO aussi soit basé en ASP, est que dans le futur nous pourrions facilement combiner les deux analyses : l'apprentisage de modèles booléens et celui des paramètres en incluant par exemple des données cliniques. Ce travail fait partie des perspectives de ce résumé.

Références

[1] Chitta Baral. *Knowledge Representation, Reasoning, and Declarative Problem Solving*. Cambridge University Press, New York, NY, USA, 2003.

[2] Lokmane Chebouba, Bertrand Miannay, Dalila Boughaci, and Carito Guziolowski. Discriminate the response of acute myeloid leukemia patients to treatment by using proteomics data and answer set programming. *BMC Bioinformatics*, 19(2) :59, Mar 2018.

[3] Minoru Kanehisa and Susumu Goto. Kegg : Kyoto encyclopedia of genes and genomes. *Nucleic Acids Research*, 28(1) :27–30, 2000.

[4] Li Liu, Yung Chang, Tao Yang, David P Noren, Byron Long, Steven Kornblau, Amina Qutub, and Jieping Ye. Evolution-informed modeling improves outcome prediction for cancers. *Evolutionary Applications*, 10(1) :68–76, 2017.

[5] Santiago Videla, Julio Saez-Rodriguez, Carito Guziolowski, and Anne Siegel. caspo : a toolbox for automated reasoning on the response of logical signaling networks families. *Bioinformatics*, 33(6) :947–950, 2017.

Bound-Impact, un sélecteur de valeur boite-noire pour l'optimisation

Jean-Guillaume Fages[1], **Charles Prud'Homme**[2]
[1]COSLING S.A.S., France
[2]TASC - IMT Atlantique, France
jg.fages@cosling.com charles.prudhomme@imt-atlantique.fr

Résumé

Cet article résume les travaux réalisés sur *Bound-Impact* [1], un sélecteur de valeur générique pour l'optimisation. En sélectionnant la valeur offrant la meilleur borne pour la variable objectif, il permet de plonger rapidement vers une première solution de bonne qualité, là où les heuristiques boites-noires actuelles ont une approche *fail-first* aveugle à la notion de coût.

1 Introduction

La simplicité de modélisation offerte par son langage déclaratif est un des atouts majeurs de la Programmation Par Contraintes (PPC). Néanmoins, les modèles les plus simples ne sont pas toujours les plus performants et il est souvent nécessaire de spécifier une heuristique de recherche dédiée pour obtenir de bonnes solutions dans des temps raisonnables. Cette tâche pouvant s'avérer complexe, de nombreuses heuristiques boites-noires ont été proposées [2, 3, 4, 5]. Ces travaux ont permis de repousser les limites de la PPC sur de nombreux problèmes. Cependant, ils reposent généralement sur des approches *fail-first* [6] qui, bien que pertinentes en satisfaction ou pour réaliser une preuve d'optimalité, ont tendance à calculer des solutions de mauvaise qualité. Or, dans la plupart des applications industrielles, il est essentiel de pouvoir calculer une bonne solution très rapidement. Ce article propose un pas supplémentaire dans la recherche d'heuristiques boites-noires efficaces, en introduisant un sélecteur de valeur générique appelé *Bound-Impact Value Selector* (BIVS), qui améliore significativement la qualité des solutions trouvées en PPC sur de nombreux problèmes, quelle que soit l'heuristique de choix de variable. En particulier, cette heuristique permet d'obtenir une première solution de bonne qualité, ce qui est très important pour les applications réelles, où il est courant de s'y arrêter [7].

2 Le sélecteur de valeur *Bound-Impact*

Definition 1 *Etant donné un problème de minimisation (maximisation), l'heuristique de choix de valeur* BIVS *sélectionne la valeur dont l'affectation, une fois propagée, offre la plus petite (grande) borne inférieure (supérieure) pour la variable objectif.*

Il s'agit donc de simuler chaque affectation possible et mesurer leur impact sur la fonction objectif, afin de descendre dans la direction la plus prometteuse. Puisque BIVS est uniquement un sélecteur de valeur, il est possible de le combiner avec les heuristiques de choix de variables existantes. Cela offre des possibilités très intéressantes d'usages.

BIVS trouve ses origines dans la méthode *Strong branching*, utilisée en optimisation linéaire en nombres entiers, et l'heuristique IBS, utilisée en programmation par contraintes. Ces deux techniques s'appliquent á l'ensemble des variables candidates au branchement et mesurent un impact, sur la fonction de coût pour la première et sur la réduction de l'espace de recherche pour la seconde. Étant une heuristique de choix de valeur, BIVS ne considère qu'une variable à la fois, ce qui en réduit le surcoût algorithmique.

BIVS introduisant une étape d'exploration en largeur, il est intéressant de le positionner vis-à-vis d'HBFS [8]. Leurs principales différences sont : 1) la composante « largeur » de BIVS est locale au domaine de la variable de décision courante, i.e. locale au noeud courant, alors qu'elle porte sur les fils gauches et droits de tout l'arbre de recherche dans le

cas d'HBFS et 2) BIVS est un sélecteur de valeur alors qu'HBFS est une stratégie de parcours de l'espace de recherche, équipée d'une heuristique de choix de variables-valeurs. BIVS et HBFS sont combinables.

Enfin, puisque la phase de simulation peut conduire à identifier plus tôt des affectations non réalisables, un dernier parallèle peut être fait avec la notion de Singleton Arc Consistency [9], restreinte à la variable de décision courante.

3 Evaluations

BIVS a été implémenté dans le solveur de contraintes Choco Solver 4.0.5 [10] et est dorénavant utilisé dans l'heuristique par défaut pour traiter les problèmes d'optimisation. Les expériences suivantes ont été réalisées sur un ordinateur Mac Pro 8-core Intel Xeon E5 @3Ghz sous l'environnement MacOS 10.12.5 et Java 1.8.0_25.

Dans cette expérience, nous considérons le TSP, un problème combinatoire classique dont la satisfaction est triviale, toute permutation étant réalisable, mais la gestion des coût rend le problème NP-difficile. Nous considérons le modèle PPC le plus répandu du TSP, reposant sur des variables de décision représentant les successeurs de chaque noeud, une contrainte CIRCUIT et des contraintes ELEMENT pour capturer le coût unitaire de chaque trajet. Nous comparons ensuite les résultats obtenus par trois heuristiques différentes :

DEFAULT choix de variable : DOMWDEG ; choix de valeur : sélection de la borne inférieure. Cela correspond à l'heuristique par défaut du solveur, et donc à ce que la plupart des utilisateurs obtiendront.

MINCOST choix de variable : ordre statique ; choix de valeur : sélection du successeur le plus proche du noeud courant (d'après la matrice de distance). Il s'agit de l'approche classique pour un utilisateur avancé, capable de spécifier sa propre heuristique de recherche.

BIVS choix de variable : ordre statique ; choix de valeur : valeur conduisant à la plus petite borne inférieur pour l'objectif (BIVS). Il s'agit de la contribution de l'article.

Les résultats obtenus pour la première solution trouvée et au bout de 30 secondes sont affichés sur la Table 1. Ces résultats confirment qu'il est essentiel d'intégrer la fonction objectif dans l'heuristique de branchement. De plus, assez étonnamment, BIVS permet d'obtenir de biens meilleurs résultats que MINCOST, malgré son surcoût algorithmique.

Afin d'étudier le comportement de BIVS sur un plus large panel de problèmes, nous considérons

Taille instance	Heuristique	1^{ere} sol. temps (s)	1^{ere} sol. coût	der. sol. coût
10	DEFAULT	0.01	748	241
10	MINCOST	0.01	380	241
10	BIVS	0.01	**310**	241
50	DEFAULT	**0.02**	3775	2043
50	MINCOST	0.03	629	352
50	BIVS	0.60	**455**	**327**
100	DEFAULT	**0.16**	7627	6008
100	MINCOST	0.17	748	497
100	BIVS	9.80	**570**	**428**

TABLE 1 – Évaluation de BIVS sur le TSP

les 403 instances MiniZinc [11, 12] issues des dernières compétitions de solveurs en optimisation. Nous comparons les résultats obtenus en suivant l'heuristique donnée dans le fichier de l'instance (FIX) avec différentes heuristiques boites-noires (ABS, IBS et DOMWDEG [1]). Enfin, nous évaluons la combinaison de DOMWDEG avec BIVS. La figure 1 indique, pour chaque heuristique, le nombre d'instances pour lesquelles le solveur a trouvé la meilleure solution. Chaque exécution disposait d'un temps limite de 15 minutes. Clairement, BIVS permet d'améliorer la capacité du solveur à trouver la meilleure solution. Il est donc pertinent d'utiliser ce sélecteur dans un cadre boite-noire.

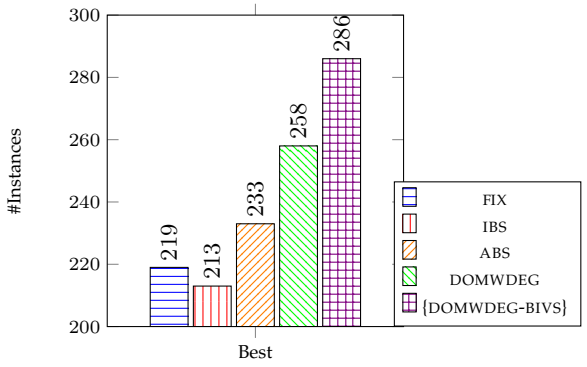


FIGURE 1 – Évaluation de BIVS sur les instances du challenge MiniZinc

4 Conclusion

Cet article introduit l'heuristique de choix de valeur *Bound-Impact* (BIVS), qui sélectionne la valeur d'une variable de décision, offrant la meilleure borne sur la variable objectif après application et propagation de son affectation. Cette heuristique, à la fois générique et performante, est très utile lorsqu'il s'agit de calculer rapidement de bonnes solutions, ce qui est le cas de la majorité des applications industrielles.

1. En utilisant InDomainMin en sélecteur de valeur.

Références

[1] J.-G. Fages and C. Prud'homme, "Making the first solution good !" in *ICTAI*, 2017.

[2] P. Refalo, "Impact-based search strategies for constraint programming," in *Principles and Practice of Constraint Programming*, 2004, pp. 557–571.

[3] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais, "Boosting systematic search by weighting constraints," in *ECAI*, 2004, pp. 146–150.

[4] L. Michel and P. Van Hentenryck, "Activity-based search for black-box constraint programming solvers," in *CPAIOR*, 2012, pp. 228–243.

[5] C. Lecoutre, L. Sais, S. Tabary, and V. Vidal, "Last conflict based reasoning," in *ECAI 2006, Including (PAIS 2006), Proceedings*, ser. Frontiers in Artificial Intelligence and Applications, vol. 141. IOS Press, 2006, pp. 133–137.

[6] R. M. Haralick and G. L. Elliott, "Increasing tree search efficiency for constraint satisfaction problems," in *IJCAI'79*. Morgan Kaufmann Publishers Inc., 1979, pp. 356–364.

[7] S. Kadioglu, M. Colena, S. Huberman, and C. Bagley, "Optimizing the cloud service experience using constraint programming," in *Principles and Practice of Constraint Programming*, ser. LNCS, G. Pesant, Ed., vol. 9255. Springer, 2015, pp. 627–637.

[8] D. Allouche, S. De Givry, G. KATSIRELOS, T. Schiex, and M. Zytnicki, "Anytime hybrid best-first search with tree decomposition for weighted CSP," in *CP 2015*, 2015, p. 17 p.

[9] C. Bessiere and R. Debruyne, "Theoretical analysis of singleton arc consistency and its extensions," *Artif. Intell.*, vol. 172, no. 1, pp. 29–41, 2008.

[10] C. Prud'homme, J.-G. Fages, and X. Lorca, *Choco Solver Documentation*, TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2016. [Online]. Available : http://www.choco-solver.org

[11] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack, "Minizinc : towards a standard cp modelling language," in *CP*. Springer-Verlag, 2007, pp. 529–543.

[12] NICTA, *MiniZinc and FlatZinc*, http ://www.g12.cs.mu.oz.au/minizinc/, 2011.

Vers un système de contraintes pour l'analyse des erreurs de précision des calculs sur les flottants*

Remy Garcia[1†] **Claude Michel**[1] **Marie Pelleau**[1] **Michel Rueher**[1]

[1] Université Côte d'Azur, CNRS, I3S, France
{prénom.nom}@i3s.unice.fr

Résumé

Les calculs sur les nombres flottants induisent des erreurs liées aux arrondis nécessaires à la clôture de l'ensemble des flottants. Ces erreurs, symptomatiques de la distance entre le calcul sur les flottants et le calcul sur les réels, sont à l'origine de nombreux problèmes, tels que la précision ou la stabilité de ces calculs. Elles font l'objet de nombreux travaux qui s'appuient sur une surestimation des erreurs effectives. Si ces approches permettent une estimation de l'erreur, estimation qui peut être affinée en découpant l'espace de recherche en sous-domaines, elles ne permettent pas, à proprement parler, de raisonner sur ces erreurs et, par exemple, de produire un jeu de valeurs d'entrée capable d'exercer une erreur donnée. Afin de pallier ce manque et d'enrichir les possibilités d'analyses de ces erreurs de précision, nous introduisons dans un solveur de contraintes sur les flottants un domaine dual à celui des valeurs, le domaine des erreurs. Ce domaine est associé à chacune des variables du problème. Nous introduisons aussi les fonctions de projections qui permettent le filtrage de ces domaines ainsi que les mécanismes nécessaires à l'analyse de ces erreurs.

Abstract

Floating-point computations induce errors linked to the rounding operations required to close the set of floating-point numbers. These errors, which are symptomatic of the distance between the computations over the floats and the computation over the real numbers, are at the origin of many problems, such as the precision or the stability of floating-point computations. They are the subject of numerous works which are based on an overestimation of actual errors. These approaches allow an estimate of the error, an estimate that can be refined by splitting the search space into subdomains, but they do not, strictly speaking, make it possible to reason about these errors and, for example, to produce input values that excercise a given error. In order to overcome this lack and to enrich the possibilities of analysis of these errors, we introduce in a solver for constraints over the floats, a domain dual to that of the values, the domain of the errors. This domain is associated with each of the variables of the problem. We also introduce the projection functions that allow the filtering of these domains as well as the mechanisms required for the analysis of these errors.

1 Introduction

Les calculs sur les nombres flottants induisent des erreurs liées aux arrondis nécessaires à la clôture de l'ensemble des flottants. Ces erreurs, symptomatiques de la distance entre le calcul sur les flottants et le calcul sur les réels, sont à l'origine de nombreux problèmes, tels que la précision ou la stabilité de ces calculs. Elles sont d'autant plus critiques que l'utilisateur de ces calculs fait le plus souvent abstraction de la nature particulière de l'arithmétique des flottants et les considère comme des calculs sur les réels. Identifier, quantifier et localiser ces erreurs est une tâche ardue qui ne peut, le plus souvent, qu'être accomplie à l'aide d'outils qui l'automatisent.

Un exemple bien connu de déviation du calcul liée aux erreurs de calcul sur les flottants est le polynôme de Rump [14] :

$$333.75b^6 + a^2(11a^2b^2 - b^6 - 121b^4 - 2) + 5.5b^8 + a/(2b)$$

avec $a = 77617$ et $b = 33096$. La valeur exacte de cette expression, calculée en utilisant la bibliothèque de calcul GMP [7], est $-54767/66192 \approx -0.827396056$. Lorsque cette expression est évaluée avec des flottants simples et un arrondi au plus près, le résultat obtenu est alors $\approx 6.3382530011411 \times 10^{29}$, soit

*Ces travaux ont été partiellement supportés par l'ANR COVERIF (ANR-15-CE25-0002).

†Papier doctorant : Remy Garcia[1] est auteur principal.

une valeur très éloignée de la valeur réelle. L'importance de la différence entre ces deux calculs, qui est de $\approx -6.3382530011411 \times 10^{29}$, souligne l'intérêt d'outils d'analyse des erreurs d'arrondi.

Les erreurs de calcul sur les flottants font l'objet de nombreux travaux qui s'appuient sur une surestimation des erreurs effectives. Ainsi l'interprète abstrait Fluctuat [6, 5] combine l'arithmétique affine et les zonotopes afin d'analyser la robustesse des programmes sur les flottants. PRECiSA [15, 13] se base sur une analyse statique du programme pour évaluer les erreurs d'arrondi. Les travaux sur l'amélioration automatique du code de Nasrine Damouche [3], ainsi que ceux d'Eva Darulova [4], s'appuient sur une évaluation de l'erreur d'arrondi afin d'estimer la distance entre l'expression sur les flottant et l'expression sur les réels. Si ces approches permettent une estimation de l'erreur, estimation qui peut être affinée en découpant l'espace de recherche en sous-domaines, elles ne permettent pas, à proprement parler, de raisonner sur ces erreurs et, par exemple, de produire un jeu de valeurs d'entrée capable d'exercer une erreur donnée.

Afin de pallier ce manque et d'enrichir les possibilités d'analyses de ces erreurs de précision, nous introduisons dans un solveur de contraintes sur les flottants [16, 10, 1, 11, 12] un domaine dual à celui des valeurs, le domaine des erreurs. Ce domaine est associé à chacune des variables du problème. Nous introduisons aussi les fonctions de projections qui permettent le filtrage de ces domaines ainsi que les mécanismes nécessaires à l'analyse de ces erreurs.

Plus précisément, nous nous positionnons sur l'analyse de la déviation des calculs en flottants par rapport aux réels. En particulier, nous ignorons volontairement la possibilité d'une erreur physique initiale sur les données d'entrées. Et pour des raisons de simplicité, nous nous restreignons aux quatre opérations arithmétiques de base. Cette simplification permet aussi un calcul exact des valeurs sur les réels [1], tant des valeurs des expressions que des erreurs de calcul. Enfin, le mode d'arrondi est supposé être le mode d'arrondi par défaut, i.e., l'arrondi au plus près pair.

2 Notation et définitions

L'ensemble des nombres flottants, un sous-ensemble fini des rationnels, a été introduit pour approximer les nombres réels sur un ordinateur. La norme IEEE pour les nombres flottants [9] définie le format des différents types de flottants ainsi que le comportement des opérations arithmétique sur ces nombres. Dans la suite, les flottants sont restreint au plus courant, i.e. les flottants binaires en simple précision sur 32 bits et les flottants binaires en double précision sur 64 bits.

Un nombre flottant binaire v est représenté par un triplet (s, e, m) où s est le signe de v, e son exposant et m sa mantisse. Quand $e > 0$, v est normalisé et sa valeur est donnée par :

$$(-1)^s \times 1.m \times 2^{e-bias}$$

où le *bias* permet de représenter les valeurs négatives de l'exposant. Par exemple, pour les flottants 32 bits, s vaut 1 bit, e vaut 8 bits, m vaut 23 bits et *bias* est égal à 127.

x^+ désigne le plus petit nombre flottant strictement plus grand que x tandis que x^- désigne le plus grand nombre flottant strictement plus petit que x. C'est-à-dire que x^+ est le successeur de x alors que x^- est son prédécesseur.

Un *ulp*, pour *unit in the last place*, est la distance qui sépare deux flottants consécutifs. Cependant, cette définition est ambiguë pour les flottants qui sont des puissances de 2, tel que 1.0 : dans ce cas, et si $x > 0$, alors $x^+ - x = 2 \times (x - x^-)$. À chaque fois que cela sera nécessaire, une formulation explicite de la distance sera utilisée.

3 Quantification de la déviation du calcul

Les calculs sur les flottants se distinguent des calculs sur les réels par l'utilisation d'arrondis. L'ensemble des nombres flottants étant un sous-ensemble fini des nombres réels, le résultat d'une opération sur les flottants n'est pas, en général, un nombre flottant. Afin de clore l'ensemble des nombres flottants pour ces opérations, ce résultat doit être arrondi au nombre flottant le plus proche selon une direction d'arrondi préalablement choisie.

La norme IEEE 754 [9] définit le comportement de l'arithmétique des flottants. Pour les quatre opérations de base, cette norme impose un arrondi *correct* : le résultat d'une opération sur les flottants doit être égal à l'arrondi du résultat de l'opération équivalente sur les réels. On a alors $z = x \odot y = round(x \cdot y)$ où z, x et y sont des nombres flottants, $\odot$, l'une des quatre opérations arithmétiques de base sur les flottants, à savoir, $\oplus$, $\ominus$, $\otimes$ et $\oslash$, $\cdot$ étant l'opération équivalente sur les réels et, *round*, la fonction d'arrondi. Cette propriété limite l'erreur introduite par une opération sur les flottants à $\pm\frac{1}{2}Ulp(z)$ pour les opérations correctement arrondies au plus proche pair, l'arrondi le plus courant.

Dès lors que le résultat d'une opération sur les flottants est arrondi, ce résultat est différent du résultat attendu sur les réels. Et chacune des opérations d'une

1. En utilisant une bibliothèque de calcul sur les rationnels et aux limitations de mémoire près.

$$Addition : z = x \oplus y \rightarrow e_z = e_x + e_y + e_{\oplus}$$
$$Soustraction : z = x \ominus y \rightarrow e_z = e_x - e_y + e_{\ominus}$$
$$Produit : z = x \otimes y \rightarrow e_z = x_{\mathbb{F}} e_y + y_{\mathbb{F}} e_x + e_x e_y + e_{\otimes}$$
$$Division : z = x \oslash y \rightarrow e_z = \frac{y_{\mathbb{F}} e_x - x_{\mathbb{F}} e_y}{y_{\mathbb{F}}(y_{\mathbb{F}} + e_y)} + e_{\oslash}$$

FIGURE 1 – Calcul de la déviation pour les opérations de base

expression complexe est susceptible d'introduire une différence entre le résultat escompté sur les réels et le résultat obtenu sur les flottants. Alors que pour une opération donnée le flottant obtenu est optimal, en termes d'arrondi, le cumul de ces approximations peut conduire à des déviations importantes comme dans le cas du polynôme de Rump.

L'origine de la déviation d'un calcul sur les flottants par rapport à son équivalent sur les réels se situant au niveau de chaque opération élémentaire, il est possible de reconstruire cette déviation à partir de la composition du comportement de chaque opération élémentaire. Considérons l'une de ces opérations, la soustraction. Si les variables d'entrée de ces opérations résultent d'un calcul, elles sont entachées d'une erreur. Par exemple, pour la variable x, la déviation sur le calcul de x, e_x est donnée par $e_x = x_{\mathbb{R}} - x_{\mathbb{F}}$ où $x_{\mathbb{R}}$ dénote le résultat attendu sur les réels et $x_{\mathbb{F}}$ dénote le résultat obtenu sur les flottants. Notez qu'à la différence d'une erreur physique, e_x est signé. Ce choix est rendu nécessaire pour capturer correctement des comportements spécifiques aux flottants comme les compensations des erreurs qui peuvent intervenir au sein d'un calcul sur les flottants.
La déviation du calcul due à la soustraction peut alors être calculée de la manière suivante : pour $z = x \ominus y$, l'erreur sur z, e_z est égale à $(x_{\mathbb{R}} - y_{\mathbb{R}}) - (x_{\mathbb{F}} \ominus y_{\mathbb{F}})$. Puisque $e_x = x_{\mathbb{R}} - x_{\mathbb{F}}$ et $e_y = y_{\mathbb{R}} - y_{\mathbb{F}}$, on a

$$e_z = ((x_{\mathbb{F}} + e_x) - (y_{\mathbb{F}} + e_y)) - (x_{\mathbb{F}} \ominus y_{\mathbb{F}})$$

La déviation entre le résultat sur les réels et le résultat sur les flottants pour une soustraction peut donc se calculer grâce à la formule suivante :

$$e_z = e_x - e_y + ((x_{\mathbb{F}} - y_{\mathbb{F}}) - (x_{\mathbb{F}} \ominus y_{\mathbb{F}}))$$

Dans cette formule, le terme $((x_{\mathbb{F}} - y_{\mathbb{F}}) - (x_{\mathbb{F}} \ominus y_{\mathbb{F}}))$ caractérise l'erreur commise par l'opération de soustraction, que nous noterons $e_{\ominus}$. La formule se simplifie alors en :

$$e_z = e_x - e_y + e_{\ominus}$$

Elle comporte deux éléments : d'une part la combinaison des déviations qui entachent les valeurs d'entrée et, d'autre part, la déviation introduite par l'opération élémentaire.

La figure 1 formule la déviation du calcul pour les quatre opérations de base. Pour chacune de ces formules, le calcul de l'erreur combine les déviations qui entachent les valeurs d'entrée avec l'erreur introduite par l'opération courante. On peut aussi observer que pour le produit et la division cette déviation est proportionnelle aux valeurs d'entrée.

L'ensemble de ces formules permet le calcul de la différence entre le résultat attendu sur les réels et celui obtenu sur les flottants pour une expression complexe. C'est à partir d'elles qu'est bâti notre solveur de contraintes sur les erreurs sur les flottants.

4 Domaines d'erreurs

Dans un *CSP* classique, à toute variable x est associé $\mathcal{D}_{\mathbf{x}}$ son domaine de valeurs. Celui-ci dénote l'ensemble des valeurs possibles que cette variable peut prendre. Dans le cas des nombres flottants, le domaine des valeurs est représenté par un intervalle de flottants à bornes dans $\mathbb{F}$:

$$\mathbf{x}_{\mathbb{F}} = [\underline{x}_{\mathbb{F}}, \overline{x}_{\mathbb{F}}] = \{x_{\mathbb{F}} \in \mathbb{F}, \underline{x}_{\mathbb{F}} \leq x_{\mathbb{F}} \leq \overline{x}_{\mathbb{F}}\}$$

avec $\underline{x}_{\mathbb{F}} \in \mathbb{F}$ et $\overline{x}_{\mathbb{F}} \in \mathbb{F}$.

Les erreurs de calcul constituent une autre dimension à prendre en compte. Elles nécessitent un domaine spécifique de par la nature distincte des éléments à représenter, mais aussi, des valeurs possibles des erreurs qui appartiennent à l'ensemble des réels. Un domaine des erreurs est donc introduit ; domaine associé à chaque variable du problème. Et, puisque les contraintes traitées ici sont réduites aux quatre opérations de base, et que ces quatre opérations sont appliquées à des flottants, i.e., un sous-ensemble fini des rationnels, ce domaine peut être représenté par un intervalle de rationnels à bornes dans $\mathbb{Q}$:

$$\mathbf{e}_x = [\underline{e}_x, \overline{e}_x] = \{e_x \in \mathbb{Q}, \underline{e}_x \leq e_x \leq \overline{e}_x\}$$

avec $\underline{e}_x \in \mathbb{Q}$ et $\overline{e}_x \in \mathbb{Q}$.

Un autre domaine d'erreur est nécessaire au bon fonctionnement de notre système, le domaine des erreurs des opérations $e_{\odot}$. Contrairement aux précédents

domaines, il n'est pas lié à chaque variable du problème mais à chaque *instance* d'une opération arithmétique du problème. Tout comme le domaine des erreurs attaché à une variable, il prend ses valeurs dans l'ensemble des rationnels. On aura donc :

$$\mathbf{e}_\odot = [\underline{e}_\odot, \overline{e}_\odot] = \{e_\odot \in \mathbb{Q}, \underline{e}_\odot \leq e_\odot \leq \overline{e}_\odot\}$$

avec $\underline{e}_\odot \in \mathbb{Q}$ et $\overline{e}_\odot \in \mathbb{Q}$.

Ce triptyque domaine de valeurs, domaine d'erreurs et domaine d'erreurs sur les opérations est nécessaire pour représenter l'ensemble des phénomènes liés d'une part aux valeurs que les variables peuvent prendre et d'autre part aux différentes erreurs qui entrent en jeu dans les calculs sur les flottants.

5 Fonctions de projection

Le processus de filtrage de notre solveur s'appuie sur des fonctions de projections classiques pour réduire les domaines des variables. Si les domaines de valeurs utilisent les fonctions de projections définies dans [11] et étendues dans [1] et [10], les domaines d'erreurs nécessitent leurs propres fonctions de projections.

Les projections des domaines d'erreurs sont obtenues par extension aux intervalles des formules de la figure 1. Ces formules étant sur les réels, elles ne présentent aucune difficulté particulière lors de leur extension aux intervalles. Par exemple, pour la soustraction, on obtient les quatre fonctions de projection suivantes :

$$\mathbf{e}_z \leftarrow \mathbf{e}_z \cap (\mathbf{e}_x - \mathbf{e}_y + \mathbf{e}_\ominus)$$

$$\mathbf{e}_x \leftarrow \mathbf{e}_x \cap (\mathbf{e}_z + \mathbf{e}_y - \mathbf{e}_\ominus)$$

$$\mathbf{e}_y \leftarrow \mathbf{e}_y \cap (-\mathbf{e}_z + \mathbf{e}_x + \mathbf{e}_\ominus)$$

$$\mathbf{e}_\ominus \leftarrow \mathbf{e}_\ominus \cap (\mathbf{e}_z - \mathbf{e}_x + \mathbf{e}_y)$$

où $\mathbf{e}_x$, $\mathbf{e}_y$ et $\mathbf{e}_z$ sont les domaines d'erreurs des variables x, y et z, d'une part, et $\mathbf{e}_\ominus$ est le domaine d'erreurs de la soustraction, d'autre part.

Les fonctions de projections sur les domaines des erreurs ne portent que sur les opérations arithmétiques et l'affectation qui transmet l'erreur de calcul de l'expression à la variable affectée. Les comparateurs n'effectuent des projections que sur les domaines de valeurs car l'erreur n'intervient pas dans les comparaisons.

L'ensemble de ces fonctions de projections sont utilisées pour réduire les différents domaines des variables jusqu'à atteindre un point fixe. Pour des raisons d'efficacité, mais aussi pour pallier la potentielle présence de convergences lentes, le calcul du point fixe est arrêté dès lors qu'aucune réduction de domaine n'est supérieure à 5%.

6 Liens entre domaine de valeurs et d'erreurs

Afin de permettre à l'un des domaines, qu'il soit de valeurs ou d'erreur, de bénéficier des réductions de l'autre domaine, il faut établir des relations entre ces deux domaines.

La première de ces relations, et sans doute la plus importante, lie le domaine des valeurs avec celui des erreurs sur les opérations. Elle provient de la garantie offerte par la norme IEEE 754 que les opérations arithmétiques de base sont correctement arrondies. Les quatre opérations de base étant correctement arrondies vers le plus proche flottant pair, on a :

$$(x \odot y) - ((x \odot y) - (x \odot y)^-)/2 \leq (x \cdot y)$$

$$(x \cdot y) \leq (x \odot y) + ((x \odot y)^+ - (x \odot y))/2$$

où x^- et x^+ sont, respectivement, le flottant qui précède, succède, à x. Autrement dit, le résultat sur les flottants est à un demi ulp, la distance qui sépare deux flottants successifs, du résultat sur les réels. Et l'erreur sur l'opération est donc contenue dans cet ulp :

$$-((x \odot y) - (x \odot y)^-)/2 \leq e_\odot \leq +((x \odot y)^+ - (x \odot y))/2$$

Cette équation établit une relation entre le domaine des valeurs et celui des erreurs sur les opérations : cette erreur ne peut pas être plus grande que le plus grand demi ulp du domaine de valeurs du résultat de l'opération. La projection du domaine de valeurs du résultat de l'opération sur l'erreur sur l'opération est obtenue en étendant cette formule aux intervalles :

$$\mathbf{e}_\odot \leftarrow \mathbf{e}_\odot \cap [-min((\underline{z} - \underline{z}^-), (\overline{z} - \overline{z}^-))/2,$$
$$+max((\underline{z}^+ - \underline{z}), (\overline{z}^+ - \overline{z}))/2]$$

Notez que la contraposée de cette propriété offre une seconde opportunité de lier domaine des valeurs et domaines des erreurs. En effet, puisque l'erreur sur une opération est inférieure en valeur absolue au plus grand demi ulp du domaine des valeurs du résultat de l'opération, alors si $|e_\odot| \geq \delta > 0$, les plus petites valeurs du domaine résultat de l'opération en question ne peuvent être support d'une solution. Pour ces petites valeurs, en valeur absolue proche de zéro, si leur demi ulp est plus petit que δ, elles ne peuvent être associées à une erreur sur l'opération assez grande pour être dans le domaine de $e_\odot$.

7 Contraintes sur les erreurs

Les contraintes habituellement disponibles dans un solveur de contraintes établissent des relations entre

les variables du problème. La dualité des domaines disponibles dans notre solveur requiert d'introduire une distinction entre domaine de valeurs et domaine d'erreurs. Afin de préserver la sémantique courante des expressions, les variables continuent de dénoter les valeurs possibles. C'est sous la forme d'une fonction dédiée, $err(x)$, qu'il est possible d'exprimer des contraintes sur les erreurs. Par exemple, $abs(err(x)) \geq \epsilon$ dénote une contrainte qui impose que l'erreur sur la variable x est, en valeur absolue, supérieure à ϵ. Notez que les erreurs prenant leurs valeurs dans $\mathbb{Q}$, la contrainte porte sur les rationnels.

Lorsqu'une contrainte mêle erreurs et variables, les variables, dont les domaines sont des flottants, sont promues en rationnels. La contrainte est alors une contrainte sur les rationnels.

8 Exemples

8.1 Implémentation

Le domaine d'erreurs ainsi que les fonctions de projections on été implémenté dans Objective-CP [8], un outil d'optimisation proposant plusieurs solveur, dont un de programmation par contraintes. Ce solveur supporte déjà les contraintes sur les flottants, à travers FPCS [12](Floating Point Constraint Solver). Les opérations sur les rationnels, par exemple dans les fonctions de projections des erreurs, sont réalisées à l'aide de la librairie GMP (GNU Multiple Precision Arithmetic Library) [7].

8.2 Le polynôme de Rump

Le premier exemple que nous proposons est le polynôme de Rump présenté en introduction. Sur ce problème, le solveur n'a qu'à propager les valeurs d'entrée de a et b pour obtenir la valeur du polynôme (dans la variable r). Le solveur affiche alors les éléments suivants :

```
a : 7.76170000e+04 (YES)
ea: [0.00000000e+00, 0.00000000e+00]
b : 3.30960000e+04 (YES)
eb: [0.00000000e+00, 0.00000000e+00]
r : 6.33825300e+29 (YES)
er: [-6.33825300e+29, -6.33825300e+29]
```

qui correspondent bien aux valeurs attendues. Cet exemple montre le fonctionnement correct des fonctions de projections sur les erreurs.

8.3 Second exemple

Ce second exemple porte sur le calcul de l'expression $z = (x + y) - (x/y)$ avec $x = 0.1$ et $y \in [0.2, 0.4]$, avec, pour x et y, des erreurs initiales à 0. y étant un intervalle, le filtrage ne peut instancier les valeurs des différentes variables.

```
float x = 0.1f;
float y = [0.2f, 0.4f];
float z;
z = (x + y) - (x/y);
```

Objective-CP réduit le domaine de z à $[-1.99999988e{-}1, 2.50000000e{-}1]$ et son domaine d'erreur à $[-7.45058060e{-}9, 1.49011612e{-}8]$. Pour Fluctuat, le domaine de z est réduit à $[-1.99999988e{-}1, 2.50000000e{-}1]$, soit le même intervalle que celui proposé par notre solveur. Par contre, le domaine de l'erreur sur z n'est réduit qu'à $[-5.96046448e{-}8, 5.96046448e{-}8]$, soit un intervalle plus grand que dans notre cas.

On peut supposer que cette différence est principalement liée à la soustraction pour laquelle l'erreur est traitée comme une différence dans notre cas alors qu'en terme d'erreur physique, les erreurs sont ajoutées.

8.4 Solve cubic

La fonction solve cubic calcule les racines réelles de l'expression $x^3 + ax^2 + bx + c = 0$. Cette version a été prise dans la GSL (GNU Scientific Library) [2], une bibliothèque largement utilisée, qui propose de nombreux outils et fonctions dédiés au calcul scientifique. Le corps de la fonction `solve_cubic` est écrit de la façon suivante :

```
int gsl_poly_solve_cubic (double a,
                          double b, double c, ...) {
  double q = (a * a - 3 * b);
  double r = (2 * a * a * a - 9 * a * b + 27 * c);

  double Q = q / 9;
  double R = r / 54;

  double Q3 = Q * Q * Q;
  double R2 = R * R;

  double CR2 = 729 * r * r;
  double CQ3 = 2916 * q * q * q;

  if (R == 0 && Q == 0) {
    ...
  } else if (CR2 == CQ3) {
    ...
  } else if (R2 < Q3) {
    ...
  } else {
    ...
  }
}
```

À la lecture de ce code, une question se pose sur la première condition, i.e. `R == 0 && Q == 0` : existe-t-il des valeurs d'entrée pour lesquelles R et Q sont

égal à zéro et s'il y en a, est-ce que R et Q sont calculés avec une erreur. Effectivement, il n'est pas recommandé de tester si une variable est égale à zéro dans les programmes numériques (hormis pour éviter une exception, comme une division par zéro). Afin de répondre à cette question, considérons les intervalles suivant pour les variables d'entrées, $a \in [14.0, 16.0]$, $b \in [-200, 200]$, et $c \in [-200, 200]$, de plus, considérons que ces variables ne sont pas entachées d'erreur, i.e., que l'erreur de calcul initiale sur ces variables est nulle. R et Q dépendent respectivement de r et q qui eux même dépendent directement des variables d'entrées. Ainsi, un CSP qui représente toutes ces relations des variables d'entrées jusqu'à la condition étudiée est :

$$
\begin{aligned}
&a \in [14.0, 16.0] \wedge b \in [-200, 200] \wedge c \in [-200, 200] \wedge \\
&err(a) = 0 \wedge err(b) = 0 \wedge err(c) = 0 \wedge \\
&q = (a * a - 3 * b) \wedge \\
&r = (2 * a * a * a - 9 * a * b + 27 * c) \wedge \\
&Q = q/9 \wedge \\
&R = r/54 \wedge \\
&R == 0 \wedge Q == 0
\end{aligned}
$$

Ces contraintes sont suffisantes pour s'assurer que le programme atteigne bien la première branche if. Cependant, ce CSP ne pose pas encore de contraintes sur l'erreur de R et de Q.

Dans un premier temps, il est intéressant de savoir si il existe, au sein de leurs domaines respectifs, des valeurs des variables d'entrée a, b, et c telles que R et Q sont exactement égal à 0. À ces fins, il faut ajouter les contraintes $err(Q) = 0$ et $err(R) = 0$ dans le CSP. Comme résultat, le solveur écrit que lorsque a est fixé à 15, b est fixé à 75, et c est fixé à 125, les erreurs sur Q et R sont égales à 0. Il est intéressant de noter que toutes les autres variables du programme sont mises à 0 avec une erreur à 0.

Dans un second temps, soulevons la question de l'existence de valeurs d'entrées pour lesquelles la condition est vraie alors que R et Q sont calculés avec des erreurs. Pour cela, il suffit de remplacer les contraintes sur les erreurs par $err(Q) > 0$ et $err(R) > 0$. Comme avec les contraintes précédentes, ce modèle est résolu dans Objective-CP et donne $a = 1.50100000e{+}01$, $b = 7.51000333e{+}01$, et $c = 1.25250167e{+}02$ comme solution. Avec ces valeurs d'entrées, Q et R sont toujours égals à 0 mais avec une erreur de $8.14913569e{-}16$ et $1.30826243e{-}14$ respectivement. De plus, les autres variables sont aussi égales à 0 mais avec une erreur supérieure à 0.

Par conséquent, légerement modifier les valeurs d'entrées d'une expression permet de passer d'un calcul correct à un calcul entaché d'erreurs d'arrondis. De plus, à cause de l'arrondi, une condition peut être vraie sur les flottants, mais fausse sur les réels.

9 Conclusion

Dans cet article, nous avons introduit un solveur de contraintes capable de raisonner sur les erreurs de calcul sur les flottants. Il repose sur un système de domaines duals, le premier représentant les valeurs possibles qu'une variable du problème peut prendre, le second représentant les erreurs commises lors du calcul de ces erreurs. Des domaines particuliers attachés aux instances des opérations arithmétiques des expressions numériques utilisées dans les différentes contraintes du problème représentent les erreurs dues à chacune de ces opérations. Augmenté des fonctions de projections sur les erreurs et de contraintes sur les erreurs, notre solveur offre des possibilités uniques de raisonner sur les erreurs de calcul.

La prochaine étape de nos travaux consiste à améliorer la recherche de solutions en présence de contraintes sur les erreurs et à ajouter des capacités d'optimisation globale pour, par exemple, déterminer pour quel jeu de valeurs d'entrée, l'erreur est maximale.

Références

[1] Bernard Botella, Arnaud Gotlieb, and Claude Michel. Symbolic execution of floating-point computations. *Software Testing, Verification and Reliability*, 16(2) :97–121, 2006.

[2] GSL Project Contributors. GSL - GNU scientific library - GNU project - free software foundation (FSF). http ://www.gnu.org/software/gsl/, 2010.

[3] Nasrine Damouche, Matthieu Martel, and Alexandre Chapoutot. Improving the numerical accuracy of programs by automatic transformation. *STTT*, 19(4) :427–448, 2017.

[4] Eva Darulova and Viktor Kuncak. Sound compilation of reals. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 235–248. ACM, 2014.

[5] Khalil Ghorbal, Eric Goubault, and Sylvie Putot. A logical product approach to zonotope intersection. In *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, volume 6174 of *Lecture Notes in Computer Science*, pages 212–226, 2010.

[6] Eric Goubault and Sylvie Putot. Static analysis of numerical algorithms. In *Static Analysis, 13th International Symposium, SAS 2006, Seoul, Korea, August 29-31, 2006, Proceedings*, volume 4134 of *Lecture Notes in Computer Science*, pages 18–34, 2006.

[7] Torbjörn Granlund and the GMP development team. GNU MP : The GNU Multiple Precision Arithmetic Library, 2016. `http://gmplib.org/`.

[8] Pascal Van Hentenryck and Laurent Michel. The objective-cp optimization system. In *19th International Conference on Principles and Practice of Constraint Programming (CP 2013)*, pages 8–29, 2013.

[9] IEEE. *754-2008 - IEEE Standard for floating point arithmethic*, 2008.

[10] Bruno Marre and Claude Michel. Improving the floating point addition and subtraction constraints. In *Proceedings of the 16th international conference on Principles and practice of constraint programming (CP'10)*, LNCS 6308, pages 360–367, St. Andrews, Scotland, 6–10th September 2010.

[11] Claude Michel. Exact projection functions for floating point number constraints. In *AI&M 1-2002, Seventh international symposium on Artificial Intelligence and Mathematics (7th ISAIM)*, Fort Lauderdale, Floride (US), 2–4th January 2002.

[12] Claude Michel, Michel Rueher, and Yahia Lebbah. Solving constraints over floating-point numbers. In *7th International Conference on Principles and Practice of Constraint Programming (CP 2001)*, pages 524–538, 2001.

[13] Mariano Moscato, Laura Titolo, Aaron Dutle, and César A. Muñoz. Automatic estimation of verified floating-point round-off errors via static analysis. In Stefano Tonetta, Erwin Schoitsch, and Friedemann Bitsch, editors, *Computer Safety, Reliability, and Security : 36th International Conference, SAFECOMP 2017, Trento, Italy, September 13-15, 2017, Proceedings*, volume 10488 of *Lecture Notes in Computer Science*, pages 213–229, Cham, 2017. Springer International Publishing.

[14] Siegfried M. Rump. Algorithms for verified inclusions : Theory and practice. In Ramon E. Moore, editor, *Reliability in Computing : The Role of Interval Methods in Scientific Computing*, pages 109–126. Academic Press Professional, Inc., San Diego, CA, USA, 1988.

[15] Laura Titolo, Marco Feliu, Mariano Moscato, and César Muñoz. An abstract interpretation framework for the round-off error analysis of floating-point programs. In Isil Dillig and Jens Palsberg, editors, *Proceedings of the 19th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2018)*, volume 10747 of *Lecture Notes in Computer Science*, pages 516–537, Los Angeles, CA, UAS, January 2018. Springer.

[16] Heytem Zitoun, Claude Michel, Michel Rueher, and Laurent Michel. Search strategies for floating point constraint systems. In *23rd International Conference on Principles and Practice of Constraint Programming, CP 2017*, LNCS 10416, pages 707–722, Melbourne, Autralia, 28–1st August 2017.

Une nouvelle méthode pour la recherche de modèles stables en programmation logique

Tarek Khaled * **Belaïd Benhamou** **Pierre Siegel**

Aix Marseille Université, Université de Toulon, CNRS, LIS, Marseille, France.
{tarek.khaled,belaid.benhamou,pierre.siegel}@univ-amu.fr

Abstract

Dans cet article, nous introduisons une nouvelle méthode de recherche de modèles stables pour les programmes logiques. Cette méthode est basée sur une sémantique relativement nouvelle, non encore exploitée qui capte et étend la sémantique des modèles stables (Gelfond et al., 1988). La méthode proposée dans cet article, effectue principalement un processus énumératif booléen adapté à la programmation par ensembles réponses (Answer Set Programming, ASP) et la sémantique utilisée. Elle a l'avantage d'utiliser un ensemble de clauses de Horn, ayant la même taille que le programme logique d'entrée et travaille à espace constant. Elle évite ainsi la lourdeur induite par la gestion des boucles, dont souffrent la plupart des solveurs ASP utilisant la complétion de Clark et qui rend leurs complexités spatiales exponentielles dans le pire des cas. De plus, l'énumération est effectuée sur un ensemble restreint de littéraux du programme logique représentant son strong backdoor (STB). Ce qui réduit la complexité temporelle de l'algorithme qui est calculé théoriquement en fonction de la taille de l'ensemble STB. Nous avons également introduit de nouvelles règles d'inférences que la méthode utilise pour élaguer l'arbre de recherche. En plus de la recherche de modèles stables, cette méthode pourrait générer si nécessaire une sorte d'extra-modèles exprimant l'extension faite à la sémantique de (Gelfond et al.). Dans ce papier, nous nous limitons uniquement à la recherche de modèles stables afin de comparer les résultats de notre méthode avec ceux des solveurs ASP existants. Nous avons expérimenté et comparé les performances de la méthode proposée sur une variété de problèmes combinatoires pour lesquels nous avons obtenu des résultats prometteurs.

*Papier doctorant : Tarek Khaled est auteur principal.

1 Introduction

La programmation par ensembles réponses (ASP) [18] est un paradigme de programmation déclarative non monotone très utilisé pour la formulation de problèmes en intelligence artificielle. Il fournit également un cadre général pour la résolution de problèmes de décision et d'optimisation [12]. En raison de la disponibilité de plusieurs solveurs performants, l'ASP est devenu très populaire ces dernières années. Parmi les solveurs les plus connus on peut citer $smodels$ [20] ou $Clasp$ [6] mais aussi ceux basés sur les solveurs SAT comme $ASSAT$ [17] et $Cmodels$ [15].

L'idée de base derrière l'ASP est d'exprimer un problème sous la forme d'un programme logique pour chercher selon une sémantique donnée (modèles stables, well-founded...etc) les solutions du problème originel [13]. Pour obtenir une expression concise du problème, les règles du programme logique source sont généralement exprimées dans la logique du premier ordre. Le problème est alors exprimé par un programme logique *non-terminal* qui contient souvent des prédicats avec des variables. Les grounders [7] sont alors conçus pour trouver un programme logique propositionnel équivalent au programme *non terminal* d'entrée.

Plusieurs sémantiques pour la programmation logique existent dans la littérature. Depuis l'introduction de la complétion de Clark [3], beaucoup d'autres sémantiques telles que la sémantique bien fondée (well-founded) [8] et la sémantique des modèles stables [9, 10] ont été introduites. Cette dernière est l'une des sémantiques les plus utilisés en programmation logique. Le but de toutes ces sémantiques est de donner une signification à un programme logique. En particulier, ils essaient de donner un sens à la négation par échec apparaissant dans les règles du programme. C'est souvent le sens donné à la négation par échec qui distinguent entre les différentes sémantiques.

Dans cet article, nous exploitons la sémantique intro-

duite dans [2] pour proposer une nouvelle méthode de recherche de modèles stables. Dans cette sémantique, les programmes logiques sont représentés par un ensemble de clauses de Horn ayant la même taille que le programme *terminal* d'entrée. Les avantages qu'offre cette sémantique sont la simple caractérisation des modèles stables ainsi que l'extension qu'elle apporte pour la sémantique des modèles stables. La représentation du programme en un ensemble de clauses de Horn permet d'obtenir une nouvelle méthode de résolution ayant de bonnes propriétés de complexité. La méthode que nous proposons profite pleinement des avantages qu'offre cette représentation. En particulier, la méthode évite la lourdeur induite par la gestion des boucles souvent utilisé par les solveurs ASP basé sur la complétion de Clark [3], ce qui les rends non constant en terme de complexité spatiale.

La nouvelle méthode est un processus énumératif booléen défini pour l'ASP et conçu en fonction de la sémantique cité précédemment et de ses caractéristiques. Cette méthode a l'avantage d'effectuer le processus énumératif uniquement sur un ensemble restreint de littéraux appelé ici strong backdoor (STB) [21] du programme logique. La complexité de l'algorithme est calculée en fonction de l'ensemble STB. Nous avons aussi introduit de nouvelles règles d'inférence que l'algorithme utilise pour élaguer l'arbre de recherche et ainsi augmenter son efficacité en pratique. Cette méthode calcule les différentes extensions du programme logique à partir desquelles on peut générer tous les modèles stables ainsi que les extra-modèles qui prolongent la sémantique des modèles stables [9]. Mais, ici, nous avons limité la recherche aux seuls modèles stables afin d'avoir une comparaison sûre avec les autres systèmes ASP.

Dans le reste de cet article, nous rappelons d'abord dans la section 2 les principales notions sur la programmation logique et le fondement de la sémantique [2] sur laquelle repose la méthode proposée. Puis nous donnons une description de la nouvelle méthode dans la section 3. Dans la section 4, nous montrons les résultats expérimentaux obtenus sur certains problèmes combinatoires que nous comparons avec ceux d'autres méthodes. La dernière section conclut le travail et donne quelques perspectives de recherche.

2 Préliminaires

Un programme logique π est composé d'un ensemble de règles de la forme *r :tête(r)* $\leftarrow$ *corps(r)*. En général, les règles sont données dans la logique du premier ordre. Les grounders sont utilisés pour calculer le programme $Ground(\pi)$ équivalent au programme π dont les règles sont exprimées en logique propositionnelle. Dans la suite, nous nous concentrons sur les programmes propositionnels, nous écrivons π pour signifier $Ground(\pi)$.

Il existe différentes classes de programmes logiques. Ils diffèrent par la présence ou l'absence de la négation classique et de la négation par échec dans les règles du programme. Un programme logique positif π est un ensemble de règles de la forme : $r = A_0 \leftarrow A_1, A_2, ..., A_m$, avec $(m \geq 0)$ et où $A_{i\in\{0,...,m\}}$ est un atome. Il n'y a pas de négation par échec ou de négation classique dans un programme logique positif.

Un programme logique général π est un ensemble de règles de la forme : $r = A_0 \leftarrow A_1, A_2, ..., A_m, not\, A_{m+1}, ..., not\, A_n, (0 \leq m < n)$ où $A_{i\in\{0...n\}}$ est un atome et not le symbole exprimant la négation par échec. Le corps positif de r est $corps^+(r) = \{A_1, A_2, ..., A_m\}$ et le négatif est $corps^-(r) = \{A_{m+1}, ..., A_n\}$. La projection positive de r est $r^+ = A_0 \leftarrow A_1, A_2, ..., A_m$. La signification intuitive de la règle r est la suivante : si on prouve tous les atomes de $corps^+(r)$, et qu'on n'arrive à prouver aucun des atomes de $corps^-(r)$, alors on infère A_0. Le réduit d'un programme π par rapport à un ensemble d'atomes X est le programme positif π^X obtenu à partir de π en supprimant chaque instance de règle contenant un atome $not\, A_i$ dans son corps négatif tel que $A_i \in X$ et tous les atomes $not\, A_j$ tels que $A_j \notin X$ dans les corps négatifs des règles restantes. Formellement, $\pi^X = \{r^+ : corps^-(r) \cap X = \emptyset\}$. Dans le reste de l'article, nous nous concentrerons sur les programmes logiques généraux.

La sémantique la plus connue pour les programmes logiques généraux est celle des modèles stables [9]. Un ensemble X d'atomes est un modèle stable de π ssi X est identique au modèle minimal d'Herbrand du réduit π^X obtenu à partir de π en considérant l'ensemble des atomes X. Ce modèle est aussi appelé le modèle canonique de π^X, il est dénoté par $Cn(\pi^X)$. Formellement, un ensemble X d'atomes est un modèle stable de π ssi $X = Cn(\pi^X)$.

En pratique, plusieurs solveurs ASP sont basés sur la complétion de Clark [3]. Parmi eux, les solveurs ASP basés sur la procédure DPLL et les solveurs SAT. Pour traiter le concept de négation par échec, Clark a proposé le concept de complétion pour les programmes logiques (notation $comp(\pi)$). Il est connu que chaque modèle stable de π est un modèle de la complétion mais l'inverse n'est vrai que si le programme est sans boucle [5]. Afin d'établir l'équivalence entre les modèles d'un programme logique et ceux de sa complétion, des formules de boucles doivent être ajoutées à la complétion [17]. Mais, le nombre de boucles peut varier d'une façon la exponentielle par rapport à la taille du programme logique donné, ce qui rendrai son traitement impraticable [16]. Par conséquent, la complexité spatiale des solveurs ASP adoptant cette approche n'est pas constante, elle pourrait varier exponentiellement dans le pire des cas. C'est le grand inconvénient des solveurs ASP utilisant la complétion de Clark.

Dans notre méthode, nous utiliserons une formalisation différente de celle de Clark. Nous proposons une nouvelle

méthode qui utilisera une représentation en clause de Horn ayant la même taille que le programme logique et qui fonctionne avec une complexité spatiale constante. Nous allons prouver que notre approche est une bonne alternative à la complétion de Clark [3] utilisée par la majorité des solveurs ASP. La méthode proposée est basée sur la sémantique introduite dans [2] et qui offre plusieurs avantages fonctionnels. Cette sémantique est basée sur la notion d'extension d'un ensemble de clauses représentant le programme d'entrée. La méthode que nous présentons ici est exemptée de la lourdeur due à la gestion de boucles utilisé par les solveurs basé sur la complétion de Clark. Intuitivement, pour un programme logique donné, la méthode calcule ses extensions en ajoutant à l'ensemble de clauses de Horn, des ensembles cohérents de littéraux ($not\ A_i$) du STB. Les modèles stables du programme logique peuvent être déduits de certaines extensions qui vérifie la condition discriminante [2]. Certains autres modèles supplémentaires ou extra-modèles peuvent être obtenus à partir des extensions qui ne vérifient pas cette condition (extra-extensions).

Plus précisément, la nouvelle sémantique est basée sur un langage propositionnel classique L ayant deux types de variables : un sous-ensemble de variables classiques $V = \{A_i : A_i \in L\}$ et un autre $nV = \{not\ A_i : not\ A_i \in L\}$. Pour chaque variable $A_i \in V$, il existe une variable correspondante $not\ A_i \in nV$ désignant la négation par échec de A_i. La nouvelle sémantique donne un lien entre les deux types de variables (celles de V et celles de nV). Ce lien est exprimé par l'ajout au langage propositionnel L d'un axiome exprimant l'exclusion mutuelle entre chaque littéral $A_i \in V$ et son littéral négatif correspondant $not\ A_i \in nV$. Cet axiome d'exclusion mutuelle est exprimé par l'ensemble de clauses $ME = \{(\neg A_i \vee \neg not\ A_i) : A_i \in V\}$.

Un programme logique $\pi = \{r : A_0 \leftarrow A_1, A_2, ..., A_m, not\ A_{m+1}, ..., not\ A_n\}$ où $(0 \leq m < n)$ est exprimé dans le langage propositionnel L par un ensemble de clauses de Horn propositionnelles $CR = \{A_0 \vee \neg A_1 \vee, ..., \neg A_m \vee \neg not\ A_{m+1}, ..., \neg not\ A_n\}$ où $(0 \leq m < n)$ qui représente l'ensemble des règles du programme logique. Chaque règle $r \in \pi$ est traduite par une clause de Horn. Pour compléter la représentation du programme π dans cette nouvelle sémantique, on rajoute à l'ensemble des clauses exprimant les règles l'ensemble de clauses $ME = \{(\neg A_i \vee \neg not\ A_i) : A_i \in V\}$ exprimant l'axiome d'exclusion mutuelle. La représentation logique du programme π dans le langage propositionnel L est donnée par l'ensemble de clauses $CR \cup ME$. Un programme logique est donc exprimé par un ensemble de clauses de Horn :

$$L(\pi) = \{ \bigcup_{r \in \pi} (A_0 \vee \neg A_1 \vee, ..., \neg A_m \vee \neg not\ A_{m+1}, ..., \neg not\ A_n) \bigcup_{A_i \in V} (\neg A_i \vee \neg not\ A_i)\}.$$

Nous pouvons remarquer que la taille du codage $L(\pi)$ est de l'ordre de $taille(\pi) + 2n$, n étant le nombre de variables propositionnelles de π appartenant à V. Le facteur $2n$ dans la taille de $L(\pi)$ correspond à l'ensemble de clauses ME représentant l'axiome d'exclusion mutuelle. Mais ce dernier peut être implémenté comme une règle d'inférence sans avoir besoin de mémoriser l'ensemble de clauses ME. La méthode que nous allons décrire travaillera alors avec la forme Horn clausale $L(\pi)$ ayant exactement la même taille que le programme π. Elle énumérera sur un sous-ensemble de littéraux jouant le rôle d'un strong backdoor (STB). Ce sous-ensemble est défini par $STB = \{not\ A_i : \exists r \in \pi, not\ A_i \in corps^-(r)\} \subseteq nV$. L'ensemble STB est le sous-ensemble des littéraux positifs de type $not\ A_i$ qui apparaissent dans π.

Étant donné un programme π et son ensemble STB. Une extension de $L(\pi)$ par rapport à STB (une extension de $(L(\pi), STB)$) est l'ensemble de clauses consistant obtenu à partir de $L(\pi)$ en ajoutant un ensemble maximal de littéraux $not\ A_i \in STB$. Formellement :

Définition 1 *Soit $L(\pi)$ le codage CNF d'un programme logique π, STB son strong backdoor et un sous-ensemble $S' \subseteq STB$, l'ensemble* $\mathrm{E} = \mathrm{L}(\pi) \cup \mathrm{S}'$ *de clauses est alors une extension de* $(\mathrm{L}(\pi), \mathrm{STB})$ *si les conditions suivantes sont vérifiées :*

1. E *est consistant,*
2. $\forall \mathrm{not\ A_i} \in \mathrm{STB} - \mathrm{S}', \mathrm{E} \cup \{\mathrm{not\ A_i}\}$ *est inconsistant.*

Exemple 1 *On considère le programme logique :*

$$\pi = \{\ a \leftarrow c, not\ b \qquad b \leftarrow a \qquad c \leftarrow not\ d \qquad a \leftarrow\ \}$$

La représentation causale du programme logique π est composé par l'ensemble $L(\pi) = CR \cup ME$ où $CR = \{a \vee \neg c \vee \neg not\ b, b \vee \neg a, c \vee \neg not\ d, a\}$, $ME = \{\neg a \vee \neg not\ a, \neg b \vee \neg not\ b, \neg c \vee \neg not\ c, \neg d \vee \neg not\ d\}$ et son ensemble strong backdoor $STB = \{not\ b, not\ d\}$. On peut voir que $(L(\pi), STB)$ admet une seule extension $E = L(\pi) \cup \{not\ d\}$. En effet, E est maximalement consistant par rapport à l'ensemble STB. C'est à dire, si par exemple nous ajoutons $not\ b$ à l'extension E, l'ensemble de clauses qui en résulte devient inconsistant.

Il a été démontré dans [2] que chaque modèle stable d'un programme logique π est représenté par une extension E de sa forme logique $L(\pi)$ qui vérifie la condition discriminante $(\forall \mathrm{A_i} \in \mathrm{V}, \mathrm{E} \models \neg \mathrm{not\ A_i} \Rightarrow \mathrm{E} \models \mathrm{A_i})$. Certaines extensions de $L(\pi)$ ne correspondent à aucun modèle stable. Ces extra-extensions coïncident avec des extra-modèles qui représentent une extension à la sémantique des modèles stables [9]. La caractérisation d'un modèle stable se fait de manière très simple grâce à une condition simple à vérifier que doivent satisfaire les extensions correspondant aux modèles stables. Cette condition est appelée condition discriminante dans [2]. Formellement, nous avons :

Théorème 1 *Si X est un modèle stable d'un programme logique π, alors il existe une extension E de $(L(\pi), STB)$ telle que* $\mathrm{X} = \{\mathrm{A_i} \in \mathrm{V} : \mathrm{E} \models \mathrm{A_i}\}$. *D'autre part, E vérifie la condition dite discriminante :* $(\forall A_i \in V, E \models \neg not\, A_i \Rightarrow E \models A_i)$.

Comme toute extension E est formée par un ensemble de clauses de Horn, alors la résolution unitaire est suffisante pour déduire tout atome A_i à partir de E ($E \models A_i$).

Il est également montré dans [2] que pour chaque modèle stable X du programme logique π il existe une extension correspondante E de $L(\pi)$ à partir duquel X peut être déduit par résolution unitaire.

Exemple 2 *L'extension $E = L(\pi) \cup \{not\, d\}$ trouvé dans l'exemple 1 vérifie la condition discriminante. Le modèle stable $M = \{a, b, c\}$ est déduit de E par résolution unitaire.*

3 Présentation de la nouvelle méthode de recherche de modèles stables

Nous présenterons ici un algorithme de recherche de modèles stables basé sur la nouvelle sémantique [2]. Une description préliminaire de cette méthode a été présentée dans [14]. Étant donné un programme logique π, cet algorithme calcule toutes les extensions de $(L(\pi), STB)$ à partir desquelles les modèles stables seraient déduits par résolution unitaire. Intuitivement, la recherche des extensions de $(L(\pi), STB)$ se fait par l'ajout progressive de littéraux $not\, A_i$ du STB en vérifiant à chaque ajout la consistance de l'ensemble de clauses obtenu. Ensuite, si on ne s'intéresse qu'aux modèles stables, il suffit de retenir que les extensions qui vérifient la condition discriminante. En d'autres mots, on élimine les extra-extensions qui ne vérifient pas celle-ci.

Il existe deux grandes approches dans la conception de systèmes ASP pour le calcul de modèles stables. L'une d'entre elles s'appuie sur les propriétés de la sémantique utilisée pour concevoir le système, et l'autre transforme le programmes logique considéré en un problème de satisfaisabilité booléenne (SAT) selon la complétion de Clark pour lequel on pourrait utiliser des solveurs SAT dans la résolution. Notre algorithme fait partie de la première approche. Nous avons adapté la procédure DPLL au cadre de l'ASP et de la nouvelle sémantique.

Le principe de notre méthode de calcul est le suivant : on construit incrémentalement une extension en alternant dans l'arbre de recherche des nœuds déterministes correspondant aux propagations unitaires et des nœuds non déterministes appelés points de choix définis par l'affectation d'une valeur de vérité (vrai ou faux) à un littéral de l'ensemble strong backdoor STB. Nous introduisons quelques nouvelles règles d'inférence qui permettent d'augmenter le nombre de propagations unitaires et, par conséquent, réduire l'espace de recherche. L'algorithme que nous présentons permet de calculer tous les modèles stables d'un programme logique donné.

3.1 Fondements théoriques de la méthode

Nous allons maintenant introduire quelques règles d'inférence que la méthode utilisera par la suite dans le processus d'énumération de modèles stables.

Définition 2 *Soit un programme π et $L(\pi)$ sa forme clausale. On définit sur $L(\pi)$ les deux règles d'inférence suivantes :* $\dfrac{A_i}{\neg not\, A_i}$ *et* $\dfrac{not\, A_i}{\neg A_i}$.

Avec la considération de ces deux règles dans le raisonnement de la méthode, on pourrait supprimer de $L(\pi)$ le sous-ensemble de clauses ME des exclusions mutuelles. On obtient ainsi une taille de $L(\pi)$ identique à celle du programme π. La forme clausale du programme logique π serait réduite à : $L(\pi) = \{ \bigcup_{r \in \pi} (r : A_0 \vee \neg A_1 \vee, ..., \neg A_m \vee \neg not\, A_{m+1}, ..., \neg not\, A_n)\}$. L'utilisation et l'implémentation des deux règles d'inférence introduites précédemment dans le processus de résolution permettraient d'accroître le nombre de propagations unitaires et élagueraient l'arbre de recherche.

La complexité d'une méthode d'énumération dans un espace de recherche représenté par un arbre binaire est souvent calculée en fonction du nombre de points de choix effectués. Dans le cas de notre méthode, l'énumération est faite sur le sous-ensemble de variables $STB = \{not\, A_i : \exists r \in \pi, A_i \in r^-\}$ qui représente le strong backdoor. Soit $C_{STB} = \{c_i = \neg not\, A_{i_1} \vee, ..., \vee \neg not\, A_{i_k} / \mid c_i \mid \geq 1, \forall j \in \{1..k\}, not\, A_{i_j} \in STB\}$ l'ensemble de clauses négatives possibles formées par les littéraux du STB ayant au moins un littéral. L'assignation non déterministe d'un point de choix correspondant à la variable $not\, A_j$ est faite en lui affectant en premier lieu la valeur de vérité vrai pour favoriser la maximalité en variables du STB interprétées à vrai de l'extension en cours. L'exploration de la branche correspondant à l'assignation de la valeur de vérité $faux$ à $not\, A_j$ (où en affectant $\neg not\, A_j$ à vrai) n'est nécessaire que dans le cas où la première branche aurait produit au moins une sous clause $c_i \in C_{STB}$. Cette propriété, que nous allons démontrer par la suite, permettrait de réduire considérablement la complexité de l'algorithme étudié. L'algorithme vérifie a chaque affectation d'un nouveau littéral la consistance du système de clauses présent en ce nœud. Comme nous avons un ensemble de clauses de Horn, une propagation par résolution unitaire assure ce teste de consistance.

Proposition 1 *Soient π un programme logique, $L(\pi)$ sa forme Horn clausale, $L(\pi)_I$ sa forme Horn clausale simplifiée par l'instanciation partielle I correspondant au nœud courant n de l'arbre de recherche, $STB = \{not\ A_i : \exists r \in \pi, not\ A_i \in r^-\}$ son strong backdoor, et C_{STB} l'ensemble de clauses négatives possibles construites sur les littéraux de l'ensemble STB. Si $not\ A_j \in STB$ est le littéral courant à affecter au nœud n et que $\forall c_i \in C_{STB}$, $L(\pi)_I \wedge not\ A_j \not\models c_i$, alors toute extension de $L(\pi)_I \wedge \neg not\ A_j$ est aussi une extension de $L(\pi)_I \wedge not\ A_j$.*

Preuve 1 *Le sous-ensemble de clauses $L(\pi)_I$ est le système de clauses simplifié, obtenu à partir de $L(\pi)$ par la considération des littéraux interprétés dans l'instanciation partielle I. L'ensemble de clauses $L(\pi)_I$ représente le sous-problème correspondant au nœud courant n de l'arbre de recherche. Par hypothèse $not\ A_j$ est le prochain littéral du STB à affecter en ce point n de l'arbre. Le système de clauses simplifié $L(\pi)_I$ correspondant au nœud n contient deux sortes de clauses : le sous ensemble de clauses de la forme $\neg not\ A_j \vee C_1$ contenant le littéral $\neg not\ A_j$ et où C_1 représente un ensemble de bouts de clauses, et le sous ensemble clauses de C_2 ne contenant pas le littéral $not\ A_j$. Soit $e = not\ A_{i_1} \wedge ... \wedge not\ A_{i_k}$, avec $not\ Ai_j \in STB$ une extension de $L(\pi)_I \wedge \neg not\ A_j$, montrons que e est aussi une extension de $L(\pi)_I \wedge not\ A_j$. On peut remarquer que $L(\pi)_I \wedge \neg not\ A_j \equiv C_2$ et que $L(\pi)_I \wedge not\ A_j \equiv C_1 \wedge C_2$. L'ensemble e est une extension de $L(\pi)_I \wedge \neg not\ A_j$, donc $C_2 \wedge e$ est consistant. Pour montrer que e est aussi une extension de $L(\pi)_I \wedge not\ A_j$, il suffit de montrer que $C_1 \wedge C_2 \wedge e$ est consistant. Procédons par l'absurde, en supposant que $C_1 \wedge C_2 \wedge e$ est inconsistant. Il en résulte que $C_1 \wedge C_2 \wedge e \models \square$ et donc $C_1 \wedge C_2 \models \neg e$. Ce qui veut dire que $C_1 \wedge C_2 \models \neg not\ A_{i_1} \vee ... \vee \neg not\ Ai_k \in C_{STB}$. Il en résulte que $L(\pi)_I \wedge not\ A_j \models \neg not\ A_{i_1} \vee ... \vee \neg not\ Ai_k \in C_{STB}$. Donc $L(\pi)_I \wedge not\ A_j \models c_i \in C_{STB}$ et cela contredit l'hypothèse.*

En d'autre mots, si aucune clause $c_i \in C_{STB}$ n'a été produite en un point de choix de l'arbre où on a interprété à vrai un littéral $not\ A_j \in STB$, il est alors inutile d'explorer la branche correspondant au littéral négatif $\neg not\ A_j$. Cela éviterait à la méthode d'explorer des branches redondantes et inutiles . En conséquence, cette propriété permet de réduire le nombre de points de choix de l'arbre de recherche. On a introduit ainsi une nouvelle coupure dans l'arbre de recherche qui pourrait réduire la complexité de l'algorithme et qui pourrait augmenter son efficacité dans la pratique.

Proposition 2 *Si $L(\pi)$ est la forme clausale d'un programme logique π et I l'instanciation partielle courante, alors la résolution unitaire est suffisante pour produire toute clause $c_i = \neg not\ A_{i_1} \vee ... \vee \neg not\ Ai_k \in C_{STB}$ à partir de $L(\pi)_I$.*

Preuve 2 *D'après le théorème de déduction automatique, montrer que $L(\pi)_I \models c_i$ est équivalent à $L(\pi)_I \wedge \neg c_i \models \perp$. Comme $L(\pi)$ est à la base un ensemble de clauses de Horn, il en résulte que l'ensemble de clauses simplifié $L(\pi)_I \wedge \neg c_i$ l'est aussi car $L(\pi)_I \wedge not\ A_{i_1} \wedge ... \wedge not\ Ai_k$ est trivialement un ensemble de clauses de Horn. Comme la résolution unitaire est suffisante pour décider la consistance de tout ensemble de clauses de Horn, alors elle est en particulier pour $L(\pi)_I \wedge \neg c_i$. En d'autres mots, la résolution unitaire est suffisante pour montrer $L(\pi)_I \wedge \neg c_i \models \perp$ et, par conséquent, suffisante pour montrer $L(\pi)_I \models c_i$.*

Pour appliquer la coupure induite par la proposition 1 en un point de choix donné de l'arbre de recherche, notre méthode doit prouver qu'aucune clause $c_i \in C_{STB}$ n'est produite en ce nœud. Pour ce faire, la méthode essaye de produire une telle clause en utilisant la résolution unitaire (Proposition 2).

Nous allons maintenant montrer comment exploiter l'apparition de certains littéraux purs (monotones) dans la forme clausale $L(\pi)$ du programme logique π. Ces littéraux sont souvent ignorés dans les implémentations des solveurs SAT basé sur DPLL, mais, pour les solveur ASP, ils jouent un rôle très important.

Proposition 3 *Soit un programme logique π, $L(\pi)$ sa forme clausale et $\neg A_i$ un littéral pur de $L(\pi)$, si X est un modèle stable ou un extra-modèle de π alors $\neg A_i \in X$.*

Preuve 3 *Le littéral $\neg A_i$ est pur dans $L(\pi)$, ce qui implique que A_i n'a pas d'occurrences dans $L(\pi)$, donc le littéral A_i ne pourra jamais être inféré. Par conséquent, A_i ne pourra faire partie d'un modèle stable / extra-modèle X. Par l'application de l'hypothèse du monde clos, nous avons $\neg A_i \in X$.*

Cette proposition permet de traiter les littéraux purs comme des mono-littéraux à propager. La propagation de ces derniers contribuent à la réduction des points de choix de l'arbre et, en conséquence, son élagation.

Proposition 4 *Soit un programme logique π et $L(\pi)$ sa forme clausale, si $\neg A_i$ est vrai dans un modèle stable X de π alors $not\ A_i$ doit être vrai dans X.*

Preuve 4 *Si $\neg A_i$ est vrai dans le modèle stable X, alors le seul cas où $\neg not\ A_i$ pourrait être produit est l'existence d'une sous-clause $A_i \vee \neg not\ A_i$ de $L(\pi)_X$. Mais dans ce cas, l'extension correspondant à X ne vérifie pas la condition discriminante. En conséquence, X ne serait n'est pas un modèle stable, ce qui est contradictoire avec l'hypothèse.*

Cette proposition peut se traduire dans l'algorithme par la règle d'inférence ($\frac{\neg A_i}{not\ A_i}$) qui permet d'élaguer l'arbre de recherche des modèles stables.

Proposition 5 *Soit un programme logique π et $L(\pi)$ sa forme clausale, si $\neg A_i$ est vrai dans un modèle partiel X de π alors $not\ A_i$ doit être faux, alors X ne peut pas être un modèle stable.*

Preuve 5 *Si $\neg not\ A_i$ est vrai dans le modèle partiel X et dans le même temps A_i est vrai dans ce dernier, alors l'extension correspondant à X ne vérifie pas la condition discriminante. Alors, X ne peut pas être un modèle stable de π.*

Toutes ces règles induisent des coupures dans l'arbre de recherche et réduisent de manière considérable l'espace de recherche.

3.2 Description de l'algorithmique

Nous présentons dans ce qui va suivre le nouvel algorithme de recherche de modèles stables. Le processus d'énumération de ce dernier est basé sur la procédure DPLL que nous avons adaptée au cas des ASP traités par la nouvelle sémantique [2]. Nous avons aussi introduit et implémenté plusieurs nouvelles règles d'inférences pour booster la méthode. Dans ce nouvel algorithme, la recherche de modèles stables alterne des phases de propagation unitaire déterministes et des phases de points de choix non déterministes où le processus de production de clauses $c_i \in C_{STB}$ est lancé sur la première branche du point de choix où un littéral $not\ A_i$ du STB est interprété à vrai. Le processus de production est inutile sur la deuxième branche du point de choix où le littéral $not\ A_i$ du STB est interprété à faux. Durant les deux phases alternées, l'algorithme affecte des valeurs de vérité aux littéraux à la manière DPLL. Si un conflit est rencontré au cours de la recherche, alors l'algorithme explore la branche correspondant à la deuxième valeur de vérité de la variable représentant le point de choix courant uniquement si une clause $c_i \in C_{STB}$ est produite. Sinon, un rebroussement (backtrack) est effectué.

Une extension est trouvée quand toutes les clauses sont satisfaites ou bien quand tous les littéraux du STB ont été affectés sans falsifier aucune clause. Dans les deux cas, l'algorithme exécute une phase dite de *complétion*. Dans le premier cas, la méthode complète l'interprétation courante par l'assignation à vrai de l'ensemble des variables $not\ A_i$ restant du STB et par l'assignation à faux de toutes les autres variables de V non encore affectés (hypothèse du monde clos). Dans le deuxième cas, la complétion consiste selon l'hypothèse du monde clos, à mettre à faux les variables non affectées. Dans les deux cas, un modèle minimal candidat est trouvé. L'algorithme vérifie alors si le modèle candidat est bien un modèle stable.

L'algorithme commence par un premier appel à la procédure propagation-unitaire qui propage tous les mono-littéraux jusqu'à ce que la liste de ces derniers $Lmono$ se vide. Puis un appel est fait pour traiter les littéraux purs qui à leur tour peuvent induire des mono-littéraux. Quand il n'y a plus de mono-littéraux ou des littéraux purs à assigner, l'algorithme essaye de produire une clause $c_i \in C_{STB}$ (proposition 2).

Si on arrive à produire une clause $c_i \in C_{STB}$, alors la seconde branche du point de choix courant sera explorée. Si, par contre, aucune clause n'est produite et qu'il ne reste pas de mono-littéraux ou littéraux purs à propager, alors la seconde branche de la variable point de choix devient inutile et donc coupée. L'énumération continue par le choix du prochain littéral dans STB à assigner et ce processus est réitéré jusqu'à la satisfaction de toutes les clauses ou l'affectation de tous les littéraux du STB sans apparition de la clause vide. Dans ce cas, une extension $E = L(\pi)_I$ est obtenue. Il ne reste plus qu'à effectuer la phase de complétion, ensuite vérifier si l' extension obtenue satisfait la condition dite discriminante pour induire un modèle stable. Le pseudo-code du schéma général de la méthode est donné dans l'algorithme 1.

Algorithm 1 Schéma général de la nouvelle méthode de recherche de modèles stables

Require: La forme clausale $l(\pi)$ d'un programme logique π

Ensure: Tous les modèles stables de π

```
1:  S = ∅
2:  repeat
3:      while STB ≠ ∅ et 'pas de conflit' do
4:          while Lmonos ≠ ∅ or Lpure ≠ ∅ do
5:              unit-propagation(L(π),Lmonos,I) ;
6:              inférence(L(π),Lpure,I) ;
7:              clause-production(L(π)) ;
8:          end while
9:          choisir littéral ;
10:     end while
11:     if pas de conflit then
12:         E = L(π)_I est une extension candidate ;
13:         E= complétion(E) ;
14:         if Conditions(E) then
15:             M=Atomes-positive(E) ;
16:             S = S ∪ M ;
17:         end if
18:     else
19:         backtrack
20:     end if
21: until Tout l'espace de recherche est exploré
```

La procédure de propagation-unitaire (Algorithme 2) prends en entrée la forme clausale $L(\pi)$, la liste de clauses unitaires $Lmonos$, et l'interprétation partielle courante I. Elle renvoie au retour, soit une interprétation I étendue par la propagation des mono-littéraux, soit un message de conflit si une clause vide est rencontrée. La procédure com-

Algorithm 2 procédure Unit-propagation

Require: La forme clausale $L(\pi)$ du programme π, la liste de clauses unitaires $Lmonos$, l'interprétation partielle courante I

Ensure: Une interprétation étendue I ou la détection d'un conflit,

while ($Lmonos = \emptyset$) and (non conflit) **do**
  $v \leftarrow next(Lmonos)$;
  $I \leftarrow I \cup \{v\}$;
  $L(\pi) \leftarrow L(\pi) \backslash \{c_i, v \in c_i\}$;
  $c_i \leftarrow c_i \backslash \{\neg v, \neg v \in c_i\}$;
  if $length(c_i) == 1$ **then**
    $Lmonos \leftarrow Lmonos \cup \{c_i\}$
  end if
  $Lmonos = Lmonos \backslash \{v\}$;
  $v' \leftarrow inference(v)$;
  $Lmonos = Lmonos \cup \{v'\}$;
end while
Si (pas de conflit) alors retourne I ;
else retourne conflit ;

mence par satisfaire toutes le clauses où apparaît le monolittéral v et rajoute v à l'interprétation partielle I. Puis, elle réduit les clauses dans lesquelles se trouve l'opposé de v ($\neg v$). Si une clause unitaire est crée, elle est alors rajoutée à la liste $Lmonos$ qui représente l'ensemble des monolittéraux. Si une clause vide est détectée, la procédure signale le conflit.

Ensuite, l'algorithme fait appel à la fonction inférence qui implémente certaines règles d'inférences basées sur des propriétés théoriques que nous avons démontrées dans la sous-section précédente. Ces règles d'inférence permettent de réduire l'ensemble des points de choix de l'arbre de recherche. Le littéral v' retourné par la procédure *inférence* sera traité comme un mono-littéral.

Enfin, La procédure *production-clauses* utilise la unit résolution pour produire les clauses $c_i \in C_{STB}$ selon la propositon 2.

3.3 La complexité de l'algorithme

Si n est le nombre de variables de la représentation clausale $L(\pi)$ du programme π, k le cardinal de l'ensemble STB et m le nombre de clauses de $L(\pi)$, alors la complexité temporelle de l'algorithme dans le pire des cas est approximativement O($knm2^k$). Nous pouvons remarquer que le facteur exponentiel de la fonction de complexité dépend du nombre k représentant la taille de l'ensemble STB et ne dépend pas du nombre de variables n comme dans les autres solveurs ASP. La valeur de k est généralement plus petite que celle de n, d'où une meilleure complexité temporelle.

Contrairement à la plupart des solveurs ASP qui utilisent la complétion Clark avec la gestion des boucles et qui donc ont une complexité spatiale exponentielle dans le pire des cas, notre méthode fonctionne à espace constant. En effet, la méthode utilise en entrée la forme clausale $L(\pi)$ dont la taille est identique à celle du programme initial π et qui ne varie pas pendant les exécutions. La complexité spatiale est constante, elle est d'ordre $O(\mid L(\pi) \mid) = O(\mid \pi \mid)$ dans le pire des cas. Cet algorithme peut être utilisé pour des programmes logiques avec et sans boucles et permet de calculer tous les modèles stables d'un programme général donné.

4 Implémentation et expérimentation

Sur la base de l'algorithme présenté précédemment, nous avons implémenté une première version d'un nouveau solveur ASP que nous désignons par $HC - asp$ pour signifier Horn Clause ASP. Ici nous avons limité le système à chercher uniquement les modèles stables d'un programme logique (pas ses extra-modèles) afin de pouvoir le comparer à d'autres solveurs ASP. Nous rappellerons que notre méthode est basée sur une sémantique relativement nouvelle qui étend celles des modèles stables. Elle permet de calculer des extra-modèles pour des programmes logiques même dans le cas où ces programmes n'ont pas de modèle stable. Le solveur est implémenté en C++ et nous avons utilisé $gringo$ [7] comme grounder. L'objectif principal de cette implémentation est d'étudier le comportement de notre approche dans la résolution de certains problèmes et de montrer à la communauté une nouvelle alternative pour implémenter des solveurs ASP.

Pour montrer l'efficacité de notre solveur $HC - asp$, nous l'avons comparé à d'autres systèmes ASP existants. Nous avons considéré dans la comparaison le solveur $Cmodels$ (version 3.86 avec $zChaff$ comme solveur SAT) qui convertit le programme logique d'entrée en CNF selon la complétion de Clark. Il délègue ensuite à $zChaff$ le calcul des modèles et lorsque l'assignation est totale, il vérifie si le modèle est un modèle stable. Nous avons également considéré deux autres solveurs ASP connus qui sont $Smodels$ (version 2.34) et $Clasp$ (version 3.3.3). Le solveur $Smodels$ est basé sur la sémantique des modèles stables et sur la sémantique bien fondée (well-founded). Sa mise en œuvre est basée sur une procédure de recherche bottom-top avec un backtracking systématique. Le système $Clasp$ est connu pour être très efficace même pour les problèmes de satisfiabilité booléenne. C'est un solveur ASP basé sur l'analyse de conflits, le retour-arrière non chronologique et sur l'apprentissage de clauses. Il est basé sur la complétion de Clark et doit, de ce fait, gérer les formules de boucle.

Tous les systèmes sont appliqués pour rechercher tous les modèles stables et $gringo$ est utilisé comme grounder pour chacun d'eux. Les programmes fonctionnent sur une

machine Ubuntu (16,10) de 4 Go avec un processeur Intel Core i5 (1,70 GHz x 4). La durée d'exécution est limitée à 24H pour tous les systèmes appliqués.

Nous avons expérimenté différents problèmes hautement combinatoires. Pour chacun d'entre eux, nous avons progressivement augmenté la taille du problème. Le nombre de modèles stables obtenus par tous les systèmes quand ils terminent l'exécution dans la limite du temps imparti sont donné dans la colonne "Modèles stables" des Tableaux 1,2 et 3. Nous rapportons ici les résultats obtenus sur quelques benchmarks connus qui sont le problème d'accessibilité, le problème des Pigeons, le problème de Ramsey, le problème des n-reines, le circuit hamiltonien et le problème de Schur. La tâche la plus importante dans l'ASP consiste à énumérer tous les modèles stables d'un programme logique. Nous avons choisi ces benchmarks en raison de leur nombre important de modèles stables. Ils sont très appropriés pour étudier le comportement de chacun des solveurs lorsque le nombre de modèles stables et la taille du problème augmentent.

Le problème d'accessibilité (1-4) [11] est basé sur un graphe complet orienté. Le problème consiste à rechercher tous les sous-graphes dans lesquelles tous les sommets sont encore joignables les uns aux autres à travers les arcs restants. Pour les instances étudiées ici, nous avons fait varier le nombre de sommets de 2 à 5. Le benchmark représentant le théorème de Ramsey $R(k, m)$ (5-8) consiste à chercher toutes les solutions qui assignent une couleur parmi deux couleurs à un arc d'un graphe complet de n sommets de tel sorte qu'il n'y a pas de clique de k nœuds de la première couleur et pas de clique de m nœuds de la deuxième couleur. Nous avons cherché toutes les solutions pour le nombre de Ramsey $R(5, 4)$ lorsque n varie de 5 à 8.

Le problème des Pigeons(9-16) consiste à rechercher toutes les combinaisons qui permettent de mettre n pigeons dans n pigeonnier de sorte que chaque pigeon soit mis dans un pigeonnier distinct. Pour notre expérimentation nous avons fait varier le nombre n de 4 à 11. Le problème de Schur (17-28) consiste à partitionner des nombres N dans M ensembles tels que si X et Y appartiennent à un ensemble alors $X + Y$ ne peut pas appartenir à cet même ensemble. Nous avons calculé toutes les solutions du problème pour $M = 4$ et N variant de 10 à 21.

Les résultats obtenus pour la recherche de tous les modèles stables des quatre benchmarks Accessibilité, Pigeons, Ramsey and Schur sont données dans le Tableau 1. Nous pouvons voir que, sauf pour le problème de Schur et les petites instances du problème d'accessibilité (R_2 et R_3) qui ont été résolus efficacement par tous les solveurs, notre méthode $HC - asp$ surpasse toutes les autres méthodes sur les autres benchmarks. Nous pouvons remarquer que le gain augmente lorsque la taille des problèmes et le nombre de modèles stables augmentent aussi. Pour le problème de Schur, nous pouvons voir que $HC - asp$, $Clasp$ et $Smodels$ ont des résultats comparables avec une légère supériorité de $Smodels$ suivie de $HC - asp$. Nous ne connaissons pas exactement la raison, mais nous croyons que la modélisation du problème est plus adapté à $Smodels$. Le système $Smodels$ pourrait être considéré au second rang, puisqu'il surclasse à la fois $Clasp$ et $Cmodels$ sur ces problèmes. Comme le nombre de solutions à trouver est important, le grand nombre de nogood ajoutés et la gestion des boucles ralentissent $Clasp$. Malgré cela, $Clasp$ surpasse drastiquement $Cmodels$. Certains benchamrks sont très difficiles pour $Cmodels$. En effet, c'est le seul système qui ne parvient pas à résoudre dans le temps imparti les instances $R\_4\_5\_7$, $R\_4\_5\_8$ de Ramsey, les instances $pi9/9$, $pi10/10$, $pi11/11$ du problème des Pigeons et les instances de Schur lorsque $N \geq 25$. Comme pour $Clasp$, $Cmodels$ utilise la technique de nogoods et de ce fait a les même inconvénients. On pourrait rajouter à cela la grande taille de la formule CNF avec la complétion de Clark et les formule de boucles. Tous ces facteurs conduisent à ralentir le solveur.

TABLE 1 – Les résultats obtenus sur les problèmes d'accessibilité, Ramesey, Pigeons et Schur

N°	Instances	#Modèles Stables	#Temps(sec)			
			HC-asp	Clasp	Smodels	Cmodels
1	R_2	1	0.0005	0.0001	0.0002	0.0003
2	R_3	18	0.0015	0.001	0.0005	0.015
3	R_4	1606	0.030	0.070	0.022	0.091
4	R_5	565080	7.12	12.59	7.62	9991.28
5	R_4_5_5	957	0.011	0.010	0.014	0.049
6	R_4_5_6	27454	0.2	0.5	0.33	18.62
7	R_4_5_7	1452289	12.64	28.76	18.00	•
8	R_4_5_8	137578233	1625.14	3329.66	2219.19	•
9	pi4/4	24	0.007	0.009	0.002	0.003
10	pi5/5	120	0.02	0.02	0.008	0.01
11	pi6/6	720	0.06	0.06	0.04	0.07
12	pi7/7	5040	0.23	0.55	0.30	0.87
13	pi8/8	40320	1.65	4.01	2.5	52.34
14	pi9/9	362880	21.63	47.11	34.60	4591.87
15	pi10/10	3628800	210.14	494.40	369.80	•
16	pi11/11	39916800	2728.53	6936.96	4247.58	•
17	Schur4/10	2964	0.16	0.12	0.10	0.36
18	Schur4/11	8326	0.47	0.36	0.28	1.33
19	Schur4/12	18539	0.98	0.82	0.65	4.82
20	Schur4/13	48987	2.6	3.26	1.77	39.04
21	Schur4/14	95311	5.29	5.7	4.43	169.85
22	Schur4/15	247147	14.20	15.61	12.36	1286.12
23	Schur4/16	408642	26.22	27.58	24.10	3500.16
24	Schur4/17	920149	65.61	66.84	49.84	15976.8
25	Schur4/18	1597576	134.87	137.89	102.88	48134.4
26	Schur4/19	3344347	287.85	301.96	207.86	•
27	Schur4/20	3832609	406.88	436.88	292.91	•
28	Schur4/21	7924530	965.80	987.62	707.90	•

L'autre benchmark que nous avons expérimenté est celui des n-reines. Les résultats obtenus sont présentés dans le Tableau 2. Nous pouvons voir que toutes les méthodes ont résolu efficacement les petites instances (10 à 12 reines). Les résultats de $HC - asp$, $Clasp$ et $Smodels$ sont comparables sur ces instances avec un léger avantage en faveur du système $Clasp$. $Cmodels$ a été surpassé par toutes les méthodes même sur les petites instances. Nous pouvons re-

TABLE 2 – Les résultats obtenus sur le problèmes des n-reines

#Taille	#Modèles stables	#Temps(sec)			
		HC-asp	Clasp	Smodels	Cmodels
10	724	0.68	0.23	0.66	0.27
11	2680	2.79	1.02	2.95	2.48
12	14200	12.2	8.75	15.19	41.44
13	73712	79.55	122.91	87.69	1642.93
14	365596	371.73	2631.83	496.98	•
15	2279184	2797.02	34337.21	3352.37	•
16	14772512	12087.40	•	23134.22	•
17	95815104	87088.00	•	•	•

TABLE 3 – Les résultats obtenus sur le problème du circuit hamiltonien

#Taille	#Modèles Stables	#Temps(sec)			
		HC-asp	Clasp	Smodels	Cmodels
5	24	0.012	0.003	0.003	0.003
6	120	0.026	0.016	0.014	0.02
7	720	0.09	0.10	0.08	0.12
8	5040	0.64	0.78	0.67	1.69
9	40320	5.76	7.87	6.00	80.75
10	362880	66.67	89.94	72.25	6177.89
11	3628800	910	1141.84	964.15	•
12	39916800	13173.42	22263.37	15964.15	•

marquer que notre méthode surpasse drastiquement toutes les autres méthodes dans tous les autres cas. On peut voir que seul le solveur $HC - asp$ est capable de résoudre tous les problèmes avant d'atteindre le temps limite. En effet, $Cmodels$ a dépassé le temps imparti pour les instances ayant 12 reines ou plus, $Clasp$ n'arrive pas à trouver à temps tous les modèles stables pour les deux instances ayant 16 et 17 reines et $Smodels$ a dépassé le temps autorisé lors de la résolution de l'instance avec 17 reines. Nous remarquons que l'écart entre $HC - asp$ et les autres systèmes augmente lorsque la taille du problème augmente.

Maintenant, nous donnons les résultats obtenus pour le circuit Hamiltonien. Le problème consiste à trouver tous les chemins d'un graphe qui visitent chaque sommet exactement une fois et revient au point de départ. La représentation que nous avons utilisée est inspirée de celle décrite dans [19]. Nous préciserons que le programme logique exprimant le problème contient des boucles. Nous avons calculé tous les circuits hamiltoniens de quelques graphes orientés complets dont le nombre de sommets varie de 5 à 12. Les comportements de tous les solveurs appliqués sont représentés dans le tableau 3. Les résultats montrent que toutes les méthodes ont résolu efficacement les instances ayant un nombre de sommets inférieur à huit (petites instances). Les performances des méthodes sur ces petites instances sont très proches. Cependant, lorsque la taille du problème augmente (le nombre de sommets est supérieur à 8), notre solveur surpasse tous les autres. $Smodels$ obtient de meilleurs résultats que ceux de $Clasp$ et ce dernier surclasse $Cmodels$. $Cmodels$ rencontre quelques difficultés pour résoudre les grandes instances de ce problème. Il ne parvient pas à résoudre dans la limite du temps les deux instances ayant 11 et 12 sommets.

L'objectif de cette section est de montrer les performances et le comportement de notre solveur par rapport aux autres systèmes ASP. Nous pouvons conclure des résultats obtenus que notre approche est une bonne alternative pour le développement de système ASP . En effet, notre méthode est en général la meilleur pour énumérer tous les modèles stables des différents problèmes expérimentés. L'écart entre notre système et les autres systèmes augmente lorsque la taille du problème augmente. Son avantage provient à la fois de sa complexité spatial constante et du fait qu'il effectue une énumération sur un sous-ensemble de littéraux représentant le strong backdoor du programme logique d'entrée. $Cmodels$ a montré ses limites sur tous les benchmarks. Ceci peut être expliqué par le fait qu'il adopte une approche SAT basée sur la complétion de Clark et la gestion des boucles mais c'est aussi dû au grand nombre de no-goods qui ont été ajoutés. $Clasp$ est très efficace quand il est utilisé pour chercher un seul modèle stable, mais quand il est appliqué pour l'énumération de tous les modèles, il souffre à cause du grand nombre de clauses rajouté, ces clauses ont pour rôle d'éviter les solutions redondantes. $Smodels$ est plus efficace que $Clasp$ pour l'énumération de tous les modèles stables. Ceci est probablement dû au retour arrière systémique simple utilisé dans ce solveur. Les résultats présentés dans cette section illustrent la capacité de notre approche à gérer efficacement l'énumération de tous les modèles stables. Nous précisons que notre implémentation n'inclut aucune optimisation (comme les redémarrages, les structures de données paresseuses, l'apprentissage de clauses ...). De meilleurs résultats sont attendus à l'avenir lorsque toutes ces techniques seront mises en œuvre. Notre approche semble être une bonne alternative que la communauté pourrait utiliser pour implémenter des solveurs ASP.

5 Conclusion

Dans cet article, nous avons proposé une nouvelle méthode basée sur une sémantique relativement nouvelle introduite dans [2] pour calculer les modèles stables. Cette méthode a l'avantage d'utiliser une représentation clausale dont la taille est identique à celle du programme logique source. Il a une complexité spatiale constante. La sémantique utilisée prévient la méthode de la lourdeur induite par la gestion des boucles dans la complétion de Clark utilisée par presque tous les solveurs ASP connus. L'autre avantage de notre approche est le processus énumératif simplifié qui est effectué uniquement sur un sous-ensemble de littéraux représentant le strong backdoor du programme logique source. Ceci conduit à une réduction considérable de la complexité temporelle. Nous avons également proposé

et mis en œuvre certaines règles d'inférence qui sont utilisées pour réduire la taille de l'espace de recherche. Nous avons expérimenté la méthode proposée sur une variété de problèmes combinatoires connus et les résultats obtenus ont montré que notre approche est une bonne alternative pour implémenter des solveurs ASP. En effet, avec une implémentation non optimisée, nous avons surpassé plusieurs solveurs ASP efficaces comme $Clasp$, $Smodels$ et $Cmodels$.

Comme travail futur, nous cherchons d'abord à optimiser notre implémentation en introduisant les techniques utilisées dans les solveurs SAT modernes tels que les structures de données paresseuses, l'apprentissage de clauses et le redémarrage.

La symétrie s'est révélée efficace dans l'ASP lors de la résolution de problèmes combinatoires [4, 1]. Nous cherchons à intégrer l'élimination des symétries dans le solveur et à étudier leurs avantages dans la résolution de ces problèmes.

Nous envisageons d'étendre notre approche à d'autres classes de programmation logique ou à des parties de logiques non monotones plus générales.

Références

[1] Belaid Benhamou. Dynamic and static symmetry breaking in answer set programming. *LPAR*, pages 112–126, 2013.

[2] Belaïd Benhamou and Pierre Siegel. A new semantics for logic programs capturing and extending the stable model semantics. *Tools with Artificial Intelligence (ICTAI)*, pages 25–32, 2012.

[3] Keith L Clark. Negation as failure. *Logic and data bases*, pages 293–322, 1978.

[4] Christian Drescher, Oana Tifrea, and Toby Walsh. Symmetry-breaking answer set solving. *AI Communications*, 24 :177–194, 2011.

[5] Francois Fages. Consistency of clark's completion and existence of stable models. *Methods of Logic in Computer Science*, 1 :51–60, 1994.

[6] Martin Gebser, Benjamin Kaufmann, Andrá Neumann, and Torsten Schaub. Conflict-driven answer set solving. *IJCAI*, 7 :386–392, 2007.

[7] Martin. Gebser, Torsten Schaub, and Sven Thiele. Gringo : A new grounder for answer set programming. *International Conference on Logic Programming and Nonmonotonic Reasoning*, 7 :266–271, 2007.

[8] A Van Gelder, KA Ross, and JS Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM (JACM)*, 38 :619–649, 1991.

[9] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. *ICLP/SLP*, 50 :1070–1080, 1988.

[10] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New generation computing*, 9 :365–385, 1991.

[11] Tomi Janhunen. Some (in) translatability results for normal logic programs and propositional theories. *Journal of Applied Non-Classical Logics*, 16 :35–86, 2006.

[12] Tomi Janhunen and Ilkka Nimelä. The answer set programming paradigm. *AI Magazine*, 37 :13–24, 2016.

[13] Benjamin. Kaufmann, Nicola Leone, Simona Perri, and Torsten Schaub. Grounding and solving in answer set programming. *AI Magazine*, 37 :25–32, 2016.

[14] Tarek Khaled, Belaid Benhamou, and Pierre Siegel. Vers une nouvelle méthode de calcul de modéles stables et extensions en programmation logique. *JIAF*, pages 151–160, 2017.

[15] Yuliya Lierler and Marco Maratea. Cmodels-2 : Sat-based answer set solver enhanced to non-tight programs. *Logic Programming and Nonmonotonic Reasoning*, page 346–350, 2004.

[16] Vladimir Lifschitz and Alexander Razborov. Why are there so many loop formulas ? *ACM Transactions on Computational Logic (TOCL)*, 7 :261–268, 2006.

[17] Fangzhen Lin and Yuting Zhao. Assat : Computing answer sets of a logic program by sat solvers. *Artificial Intelligence*, pages 115–137, 2004.

[18] Victor W. Marek and Miros L. Truszczynski. Stable models and an alternative logic programming paradigm. *The Logic Programming Paradigm*, pages 375–398, 1999.

[19] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25 :241–273, 1991.

[20] Patrik Simons, Ilkka Nimelä, and Timo Soininen. Extending and implementing the stable model semantic. *Artificial Intelligence*, 138 :181–234, 2002.

[21] Ryan Williams, Carla P Gomes, and Bart Selman. Backdoors to typical case complexity. *International joint conference on artificial intelligence*, 18 :1173–1178, 2003.

Élimination des symétries dans une nouvelle méthode de recherche de modèles stables

Tarek Khaled * **Belaïd Benhamou**

Aix Marseille Université, Université de Toulon, CNRS, LIS, Marseille, France.
{tarek.khaled,belaid.benhamou}@univ-amu.fr

Abstract

La notion de symétrie est largement étudiée dans différents domaines, en particulier dans la programmation par contraintes où plusieurs travaux ont été réalisés. L'élimination des symétries a permis d'améliorer de manière significative les performances de nombreux solveurs. Généralement, les problèmes combinatoires contiennent un nombre important de symétries rendant leurs résolutions impraticables par les solveurs qui ne les considèrent pas. En effet, ces symétries guident souvent les solveurs à explorer inutilement des branches symétriques et redondantes. Dans ce papier, nous nous sommes intéressé à la programmation par ensembles réponses (Answer Set Programming, ASP). L'ASP est un paradigme bien connu dans la représentation et le raisonnement sur les connaissances. Cependant, seuls quelques travaux concernant l'exploitation des symétries dans l'ASP ont été faits. Dans ce travail, nous étudions la détection et l'élimination des symétries pour une nouvelle méthode ASP que nous introduisons. Cette méthode est basée sur une nouvelle sémantique qui étend celles des modèles stables. Pour montrer l'impact de l'élimination des symétries dans notre méthode, nous l'avons expérimenté avec et sans symétries sur une grande variété de problèmes combinatoires et nous l'avons comparé à d'autres systèmes ASP connus. Les résultats obtenus sont très prometteurs.

1 Introduction

La programmation par ensembles réponses (ASP) est un important paradigme utilisé pour la représentation des connaissances et le raisonnement non-monotone. Cette approche est utilisé dans une variété de problèmes hautement combinatoires, parmi eux les problèmes de graphe, de planification et le modèle checking. L'ASP est appliqué dans la robotique, la bio-informatique mais aussi dans l'industrie. C'est un outil de modélisation très expressif capable de représenter un nombre important de problèmes. C'est devenu une approche très populaire du fait de la disponibilité de nombreux systèmes performants comme $Clasp$[13], DLV [18] et $Smodels$ [25] et d'autres systèmes basé sur les solveurs SAT comme $ASSAT$ [21] et $Cmodels$ [19].

Le paradigme ASP a émergé de la recherche sur la représentation des connaissances et le raisonnement nonmonotone. De nombreuses recherches ont été effectuées pour définir la sémantique des programmes logiques et l'objectif principal est de donner un sens précis à la négation par échec. La première sémantique avait été proposée par Clark [7]. Cette sémantique est utilisée dans plusieurs solveurs ASP, mais la sémantique la plus connue pour l'ASP est celle des modèles stables [15]. Dernièrement, une nouvelle sémantique [6] a été proposé. Dans cette sémantique, un programme logique est représenté par un ensemble de clauses de Horn ayant la même taille que le programme d'entrée. L'avantage de cette sémantique est la caractérisation facile des modèles stables et l'extension qu'elle fournit pour la sémantique des modèles stables [15].

Nous proposons une nouvelle méthode pour la recherche de modèles stables qui repose sur un processus d'énumération booléenne défini pour le paradigme ASP selon la sémantique introduite dans [6]. Cette méthode a l'avantage d'effectuer le processus énumératif uniquement sur une restriction de littéraux représentant le strong backdoor (STB) du programme logique considéré [26]. La méthode cherche toutes les extensions possibles de la représentation clausale du programme logique et à partir desquelles on peut générer tous les modèles stables. Il pourrait également produire des extra-modèles qui correspondent à des extensions supplémentaires qui ne sont pas capturées par la sémantique des modèles stables. Ces extra-modèles permettent d'étendre la sémantique des modèles stables. Afin

*Papier doctorant : Tarek Khaled est auteur principal.

de comparer nos résultats à d'autres systèmes ASP, nous avons limité la recherche aux seuls modèles stables. Les modèles stables sont déduits d'un sous-ensemble d'extensions satisfaisant ce que nous appellons ici *la condition discriminante*.

D'autre part, la symétrie est étudiée dans plusieurs domaines, comme les mathématiques et l'intelligence artificielle. On dit qu'un objet est symétrique, lorsque la permutation de ses éléments laisse l'objet inchangé. La symétrie est une notion fondamentale dans les problèmes de satisfiabilité, elle permet de réduire la complexité de résolution d'un certain nombre de problèmes combinatoires. Le principe de la symétrie dans la logique propositionnelle a d'abord été présenté par Krishnamurthy dans [17]. La symétrie a été étudiée en profondeur dans [3, 4] où une élimination de symétrie dynamique est proposée pour la méthode DPLL [10] lors de la résolution du problème de satisfiabilité. Une approche statique qui élimine les symétries dans une phase de prétraitement a été introduite dans [8]. Cette approche statique consiste à ajouter des contraintes exprimant la symétrie dans la représentation initiale du problème de satisfiabilité. Cette technique a été améliorée dans [1].

Beaucoup de problèmes combinatoires exprimés dans le paradigme ASP contiennent un grand nombre de symétries. Par exemple, le problème des Pigeons est connu pour exiger un temps exponentiel de résolution lorsque les symétries ne sont pas éliminées. En effet, le solveur ASP explore tous les espaces de recherche symétriques. Il est possible d'éviter d'explorer ces espaces de recherche redondants en éliminant les symétries existantes. Jusqu'à présent, seuls quelques travaux sur l'élimination des symétries dans l'ASP ont été faits. Par exemple, la méthode présentée dans [11] traite les symétries dans l'ASP avec une approche statique et où l'encodage est basé sur la représentation corps-atomes d'un programme logique. Une autre approche est étudiée dans [5]. Cette dernière méthode élimine les symétries à la fois de façon statique et dynamique.

Dans cet article, nous discutons d'abord la nouvelle méthode ASP, puis nous traitons de la détection et de l'élimination des symétries dans la représentation en clauses de Horn que la nouvelle sémantique [6] utilise pour exprimer des programmes logiques. Nous divisons le problème de l'élimination des symétries en trois parties. Nous commençons par créer un graphe coloré qui représente la forme clausale du programme logique donné de telle sorte que les automorphismes du graphe soient identiques aux symétries de la représentation clausale. Ensuite, un ensemble de générateurs représentant le groupe d'automorphismes du graphe est calculé en utilisant des outils tel que $saucy$ [1], $autom$ [24] ou $nauty$ [22]. Enfin, des contraintes d'élimination de symétrie (symmetry-breaking predicates, SBP) sont construites et ajoutées à la formulation clausale initiale. Notre approche s'inspire des travaux présentés dans [1, 2].

Le reste de l'article est organisé comme suit. Tout d'abord, nous rappelons dans la section 2 quelques notions sur la programmation logique et la sémantique [6] sur laquelle repose notre méthode de recherche de modèles stables. Nous décrivons ensuite dans la section 3 la nouvelle méthode de recherche de modèles stables. Dans la section 4, nous donnons les définition et les propriétés théoriques de la symétrie dans cette nouvelle représentation. Nous décrivons dans la section 5 la méthode de détection des symétries avant de montrer l'approche de l'élimination de ces dernières dans la section 6. La section 7 donne les résultats expérimentaux obtenus sur certains problèmes combinatoires. La section 8 conclut le travail et donne quelques perspectives de rcherche.

2 Préliminaires

Nous présentons dans ce qui suit des notions sur la permutation, la programmation par ensembles réponses et les principales bases théoriques de la sémantique utilisée [6].

2.1 Permutations

Une permutation d'un ensemble Ω est une bijection définie de Ω vers lui même. Notons $Perm(\Omega)$ l'ensemble de toutes les permutations de Ω. La paire $(Perm(\Omega), \star)$ où $\star$ est une loi de composition de la permutation de $Perm(\Omega)$, forme le groupe de permutation de Ω. Ce groupe est représenté par l'ensemble générateur $(Gen(\Omega), \star)$. Gen est un sous-ensemble de $Perm$ et chaque élément de $Perm$ peut être écrit comme une composition d'éléments de Gen. L'orbite d'un élément ω sur lequel le groupe $Perm(\Omega)$ est appliqué $\omega^{Perm(\Omega)} = \{\omega^\sigma : \omega^\sigma = \sigma(\omega), \sigma \in Perm(\Omega)\}$.

2.2 Programmes logiques et programmation par ensemble réponses (ASP)

Un programme logique général π est un ensemble de règles de la forme *r :tête(r)* $\leftarrow$ *corps(r)*. En général, le programme est donnée dans la logique du première ordre et des systèmes "grounders" comme $gringo$ [14] ou $lparse$ sont utilisé pour produire un programme propositionnel équivalent. Un programme logique général π est un ensemble de règles de la forme : $r = A_0 \leftarrow A_1, A_2, ..., A_m, not\, A_{m+1}, ..., not\, A_n, (0 \leq m < n)$ où $A_{i \in \{0...n\}}$ est un atome et not le symbole exprimant la négation par échec. Le corps positif de r est $corps^+(r) = \{A_1, A_2, ..., A_m\}$ et le négatif est $corps^-(r) = \{A_{m+1}, ..., A_n\}$. La projection positive de r est $r^+ = A_0 \leftarrow A_1, A_2, ..., A_m$. La signification intuitive de la règle r est la suivante : si on prouve tous les atomes de $corps^+(r)$, et qu'on n'arrive à prouver aucun des atomes de $corps^-(r)$, alors on infère A_0. Le réduit d'un programme π par rapport à un ensemble d'atomes X est le

programme positif π^X obtenu à partir de π en supprimant chaque instance de règle contenant un atome $not\ A_i$ dans son corps négatif tel que $A_i \in X$ et tous les atomes $not\ A_j$ tels que $A_j \notin X$ dans les corps négatifs des règles restantes. Formellement, $\pi^X = \{r^+ : corps^-(r) \cap X = \emptyset\}$. Dans le reste de l'article, nous nous concentrerons sur les programmes logiques généraux.

La sémantique la plus connue pour les programmes logiques généraux est celle des modèles stables [15]. Un ensemble X d'atomes est un modèle stable de π ssi X est identique au modèle minimal d'Herbrand du réduit π^X obtenu à partir de π en considérant l'ensemble des atomes X. Ce modèle est aussi appelé le modèle canonique de π^X, il est dénoté par $Cn(\pi^X)$. Formellement, un ensemble X d'atomes est un modèle stable de π ssi $X = Cn(\pi^X)$.

En pratique, plusieurs solveurs ASP sont basés sur la complétion de Clark [7]. Parmi eux, les solveurs ASP basés sur la procédure DPLL et les solveurs SAT. Pour traiter le concept de négation par échec, Clark a proposé le concept de complétion pour les programmes logiques (notation $comp(\pi)$). Il est connu que chaque modèle stable de π est un modèle de la complétion mais l'inverse n'est vrai que si le programme est sans boucles [12]. Afin d'établir l'équivalence entre les modèles stables d'un programme logique et ceux de sa complétion, des formules de boucles doivent être en général ajoutées à la complétion [21]. Mais, le nombre de formules de boucles pourrait varier d'une façon exponentielle par rapport à la taille du programme logique donnée. Par conséquent, la complexité spatiale des solveurs ASP adoptant cette approche n'est pas constante, elle pourrait varier exponentiellement dans le pire des cas [20].

2.3 La sémantique utilisée dans notre méthode

Une nouvelle sémantique est proposée dans [6]. Cette sémantique utilise une représentation sous la forme d'un ensemble de clauses de Horn pour exprimer le programme logique considéré. Cette représentation a l'avantage d'avoir la même taille que celle du programme logique d'entrée.

la nouvelle sémantique est basée sur un langage propositionnel classique L ayant deux types de variables : un sous-ensemble de variables classiques $V = \{A_i : A_i \in L\}$ et un autre $nV = \{not\ A_i : not\ A_i \in L\}$. Pour chaque variable $A_i \in V$, il existe une variable correspondante $not\ A_i \in nV$ désignant la négation par échec de A_i. La nouvelle sémantique donne un lien entre les deux types de variables (celles de V et celles de nV). Ce lien est exprimé par l'ajout au langage propositionnel L d'un axiome exprimant l'exclusion mutuelle entre chaque littéral $A_i \in V$ et son littéral négatif correspondant $not\ A_i \in nV$. Cet axiome d'exclusion mutuelle est exprimé par l'ensemble de clauses $ME = \{(\neg A_i \vee \neg not\ A_i) : A_i \in V\}$.

Un programme logique $\pi = \{r : A_0 \leftarrow A_1, A_2, ..., A_m, not\ A_{m+1}, ..., not\ A_n\}$ où $(0 \leq m < n)$ est exprimé dans le langage propositionnel L par un ensemble de clauses de Horn propositionnelles $CR = \{A_0 \vee \neg A_1 \vee, ..., \neg A_m \vee \neg not\ A_{m+1}, ..., \neg not\ A_n\}$ où $(0 \leq m < n)$ qui représente l'ensemble des règles du programme logique. Chaque règle $r \in \pi$ est traduite par une clause de Horn. Pour compléter la représentation du programme π dans cette nouvelle sémantique, on rajoute à l'ensemble des clauses exprimant les règles, l'ensemble de clauses $ME = \{(\neg A_i \vee \neg not\ A_i) : A_i \in V\}$ exprimant l'axiome d'exclusion mutuelle. La représentation logique du programme π dans le langage propositionnel L est donnée par l'ensemble de clauses $CR \cup ME$. Un programme logique est donc exprimé par un ensemble de clauses de Horn :

$$HC(\pi) = \{ \bigcup_{r \in \pi} (A_0 \vee \neg A_1 \vee, ..., \neg A_m \vee \neg not\ A_{m+1}, ..., \neg not\ A_n) \bigcup_{A_i \in V} (\neg A_i \vee \neg not\ A_i)\}.$$

L'ensemble ME peut être implémenté comme deux règles d'inférences. Il n'est plus nécessaire de mémoriser ME dans $HC(\pi)$. La représentation $HC(\pi)$ aura la même taille que le programme logique π. L'algorithme que nous allons introduire dans la section suivante travaille sur la représentation $HC(\pi)$. C'est un algorithme énumératif qui effectue l'énumération sur un sous-ensemble de littéraux qui représentent le strong backdoor (STB) [26] du programme logique d'entrée. L'ensemble STB est formé par les littéraux de la forme $not\ A_i$ qui apparaissent dans le programme logique π. Il est défini par $STB = \{not\ A_i : \exists r \in \pi, not\ A_i \in corps^-(r) \subseteq nV\}$. La méthode calcule principalement les extensions de $HC(\pi)$ qui encode les modèles stables. Étant donné un programme π et son ensemble STB, une extension de $HC(\pi)$ par rapport au STB (ou simplement une extension de la paire $(HC(\pi), STB)$ est l'ensemble des clauses consistantes dérivées de $HC(\pi)$ quand on ajoute un ensemble maximal de littéraux $not\ A_i \in STB$. Formellement :

Définition 1 *Soit $HC(\pi)$ le codage CNF d'un programme logique π, STB son strong backdoor et un sous-ensemble $S' \subseteq STB$, l'ensemble $\mathrm{E} = \mathrm{HC}(\pi) \cup \mathrm{S}'$ de clauses est alors une extension de $(\mathrm{HC}(\pi), \mathrm{STB})$ si les conditions suivantes sont vérifiées :*

1. E *est consistant,*
2. $\forall \mathrm{not\ A_i} \in \mathrm{STB} - \mathrm{S}', \mathrm{E} \cup \{\mathrm{not\ A_i}\}$ *est inconsistant.*

Il a été démontré dans [6] que chaque modèle stable d'un programme logique π est représenté par une extension E de sa forme logique $HC(\pi)$ qui vérifie la condition $(\forall \mathrm{A_i} \in \mathrm{V}, \mathrm{E} \models \neg \mathrm{not\ A_i} \Rightarrow \mathrm{E} \models \mathrm{A_i})$. Cette condition est appelée condition discriminante dans [6]. Formellement, nous avons :

Théorème 1 *Si X est un modèle stable d'un programme logique π, alors il existe une extension E de $(HC(\pi), STB)$ telle que $\mathrm{X} = \{\mathrm{A_i} \in \mathrm{V} : \mathrm{E} \models \mathrm{A_i}\}$. D'autre part, E vérifie la condition dite discriminante : $(\forall A_i \in V, E \models \neg not\, A_i \Rightarrow E \models A_i)$.*

Exemple 1 *On considère le programme logique : $\pi = \{\ a \leftarrow not\, b\,;\, b \leftarrow not\, a\ \}$ La représentation clausale du programme logique π est composé par l'ensemble $HC(\pi) = CR \cup ME$ où $CR = \{a \vee \neg not\, b, b \vee \neg not\, a\}$, $ME = \{\neg a \vee \neg not\, a, \neg b \vee \neg not\, b\}$ et son strong backdoor est donné par $STB = \{not\, a, not\, b\}$. Nous pouvons voir que $(HC(\pi), STB)$ admet deux extensions $E_1 = HC(\pi) \cup \{not\, a\}$ et $E_2 = HC(\pi) \cup \{not\, b\}$. En effet, E_1 and E_2 sont maximalement consistant par rapport au STB. Ces deux extensions vérifie la condition discriminante. Alors, le programme logique a deux modèles stables $M_1 = \{b\}$ et $M_2 = \{a\}$tels que $E_1 \models M_1$ et $E_2 \models M_2$.*

3 Présentation de la nouvelle méthode de recherche de modèles stables

Nous décrivons ici la nouvelle méthode de recherche pour les modèles stables basé sur la nouvelle sémantique présenté précédemment [6]. Une description préliminaire de cette méthode a été présentée dans [16]. Pour un programme logique donné π, cette méthode calcule toutes les extensions de $(HC(\pi), STB)$ à partir desquelles les modèles stables sont déduits par résolution unitaire. Intuitivement, la recherche des extensions de $(HC(\pi), STB)$ se fait par l'addition progressive des littéraux $not\, A_i$ de l'ensemble STB à $HC(\pi)$ et en vérifiant la consistance de l'ensemble obtenu à chaque nœud. Si nous nous concentrons uniquement sur les modèles stables, il suffit de ne considérer que les extensions vérifiant la condition discriminante. En d'autres termes, nous effectuons des coupures dans l'arbre de recherche pour ignorer les extensions qui ne vérifient pas cette condition.

Le processus d'énumération construit de façon incrémentale une extension en alternant dans l'arbre de recherche entre les nœuds déterministes correspondant aux propagations unitaires et les nœuds non déterministes qui représentent les points de choix. Les points de choix sont définis par l'affectation des valeurs de vérité (vrai ou faux) à certains littéraux de l'ensemble STB. Des règles d'inférence sont utilisées par la méthode pour augmenter le nombre de propagations unitaires et réduire ainsi l'espace de recherche. La méthode proposée calcule tous les modèles stables d'un programme logique donné et, en général, pourrait rechercher certains extra-modèles lorsque des modèles stables n'existent pas. Mais, dans ce travail, nous avons limité la recherche qu'aux modèles stables.

3.1 Fondements théoriques de la méthode

Nous allons maintenant introduire quelques règles d'inférence que la méthode utilisera par la suite dans le processus d'énumération de modèles stables.

Définition 2 *Soit un programme π et $HC(\pi)$ sa forme clausale. On définit sur $HC(\pi)$ les deux règles d'inférence suivantes : $\dfrac{A_i}{\neg not\, A_i}$ et $\dfrac{not\, A_i}{\neg A_i}$.*

Ces deux règles d'inférence sont une implémentation efficace de l'ensemble de clauses $ME = \bigcup_{A_i \in V} \{(\neg A_i \vee \neg not\, A_i)\}$ de $HC(\pi)$ qui exprime l'exclusion mutuel entre la paire d'atomes A_i et $\neg not\, A_i$.

Dans le cas de notre méthode, l'énumération est effectuée uniquement sur un sous-ensemble de variables STB. Soit $C_{STB} = \{c_i = \neg not\, A_{i_1} \vee, ..., \vee \neg not\, A_{i_k} / \ |\ c_i\ | \geq 1, \forall j \in \{1..k\}, not\, A_{i_j} \in STB\}$ l'ensemble de toutes les clauses négatives possibles formées par certains littéraux de l'ensemble STB. Le traitement non déterministe d'un point de choix correspondant à $not\, A_j$ est fait d'abord par son affectation à la valeur $Vrai$ pour favoriser la maximalité d'extension courante. L'exploration de la branche correspondant à l'affectation de la valeur de vérité $Faux$ à $not\, A_j$ n'est nécessaire que lorsque la première branche produit au moins une sous-clause $c_i \in C_{STB}$. Cette propriété pourrait considérablement réduire la complexité en temps de la méthode étudiée, nous la prouverons dans la proposition 1.

Proposition 1 *Soient π un programme logique, $HC(\pi)$ sa forme Horn clausale, $HC(\pi)_I$ sa forme Horn clausale simplifiée par l'instanciation partielle I correspondant au nœud courant n de l'arbre de recherche, $STB = \{not\, A_i : \exists r \in \pi, not\, A_i \in r^-\}$ son strong backdoor, et C_{STB} l'ensemble de clauses négatives possibles construites sur les littéraux de l'ensemble STB. Si $not\, A_j \in STB$ est le littéral courant à affecter au nœud n et que $\forall c_i \in C_{STB}$, $HC(\pi)_I \wedge not\, A_j \not\models c_i$, alors toute extension de $HC(\pi)_I \wedge \neg not\, A_j$ est aussi une extension de $HC(\pi)_I \wedge not\, A_j$.*

Preuve 1 *Le sous-ensemble de clauses $HC(\pi)_I$ est le système de clauses simplifié, obtenu à partir de $HC(\pi)$ par la considération des littéraux interprétés dans l'instanciation partielle I. L'ensemble de clauses $HC(\pi)_I$ représente le sous-problème correspondant au nœud courant n de l'arbre de recherche. Par hypothèse $not\, A_j$ est le prochain littéral du STB à affecter en ce point n de l'arbre. Le système de clauses simplifié $HC(\pi)_I$ correspondant au nœud n contient deux sortes de clauses : le sous ensemble de clauses de la forme $\neg not\, A_j \vee C_1$ contenant le littéral $\neg not\, A_j$ et où C_1 représente un ensemble de bouts de clauses, et le sous ensemble clauses de C_2 ne contenant pas le littéral $\neg not\, A_j$. Soit $e = not\, A_{i_1} \wedge ... \wedge not\, A_{i_k}$,*

avec $not\ Ai_j \in STB$ une extension de $HC(\pi)_I \wedge \neg not\ A_j$, montrons que e est aussi une extension de $HC(\pi)_I \wedge not\ A_j$. On peut remarquer que $HC(\pi)_I \wedge \neg not\ A_j \equiv C_2$ et que $HC(\pi)_I \wedge not\ A_j \equiv C_1 \wedge C_2$. L'ensemble e est une extension de $HC(\pi)_I \wedge \neg not\ A_j$, donc $C_2 \wedge e$ est consistant. Pour montrer que e est aussi une extension de $HC(\pi)_I \wedge not\ A_j$, il suffit de montrer que $C_1 \wedge C_2 \wedge e$ est consistant. Procédons par l'absurde, en supposant que $C_1 \wedge C_2 \wedge e$ est inconsistant. Il en résulte que $C_1 \wedge C_2 \wedge e \models \Box$ et donc $C_1 \wedge C_2 \models \neg e$. Ce qui veut dire que $C_1 \wedge C_2 \models \neg not\ A_{i_1} \vee ... \vee \neg not\ Ai_k \in C_{STB}$. Il en résulte que $HC(\pi)_I \wedge not\ A_j \models \neg not\ A_{i_1} \vee ... \vee \neg not\ Ai_k \in C_{STB}$. Donc $HC(\pi)_I \wedge not\ A_j \models c_i \in C_{STB}$ et cela contredit l'hypothèse.

En d'autre mots, si aucune clause $c_i \in C_{STB}$ n'a été produite en un point de choix de l'arbre où on a interprété à vrai un littéral $not\ A_j \in STB$, il est alors inutile d'explorer la branche correspondant au littéral négatif $\neg not\ A_j$. Cela éviterait à la méthode d'explorer des branches redondantes et inutiles . En conséquence, cette propriété permet de réduire le nombre de points de choix de l'arbre de recherche.

Proposition 2 *Si $HC(\pi)$ est la forme clausale d'un programme logique π et I l'instanciation partielle courante, alors la résolution unitaire est suffisante pour produire toute clause $c_i = \neg not\ A_{i_1} \vee ... \vee \neg not\ Ai_k \in C_{STB}$ à partir de $HC(\pi)_I$.*

Preuve 2 *D'après le théorème de déduction automatique, montrer que $HC(\pi)_I \models c_i$ est équivalent à $HC(\pi)_I \wedge \neg c_i \models \perp$. Comme $HC(\pi)$ est à la base un ensemble de clauses de Horn, il en résulte que l'ensemble de clauses simplifié $HC(\pi)_I \wedge \neg c_i$ l'est aussi car $HC(\pi)_I \wedge not\ A_{i_1} \wedge ... \wedge not\ Ai_k$ est trivialement un ensemble de clauses de Horn. Comme la résolution unitaire est suffisante pour décider la consistance de tout ensemble de clauses de Horn, alors elle est en particulier pour $HC(\pi)_I \wedge \neg c_i$. En d'autres mots, la résolution unitaire est suffisante pour montrer $HC(\pi)_I \wedge \neg c_i \models \perp$ et, par conséquent, suffisante pour montrer $HC(\pi)_I \models c_i$.*

Pour appliquer la coupure induite par la proposition 1 en un point de choix donné de l'arbre de recherche, notre méthode doit prouver qu'aucune clause $c_i \in C_{STB}$ n'est produite en ce nœud. Pour ce faire, la méthode essaye de produire une telle clause en utilisant la résolution unitaire (Proposition 2).

3.2 Description de l'algorithmique

Nous présentons dans ce qui va suivre le nouvel algorithme de recherche de modèles stables. Le processus d'énumération de ce dernier est basé sur la procédure DPLL que nous avons adaptée au cas des ASP traités par la nouvelle sémantique [6]. Nous avons aussi introduit et implémenté plusieurs nouvelles règles d'inférences pour booster la méthode. Dans ce nouvel algorithme, la recherche de modèles stables alterne des phases de propagation unitaire déterministes et des phases de points de choix non déterministes où le processus de production de clauses $c_i \in C_{STB}$ est lancé sur la première branche du point de choix où un littéral $not\ A_i$ du STB est interprété à vrai. Le processus de production est inutile sur la deuxième branche du point de choix où le littéral $not\ A_i$ du STB est interprété à faux. Durant les deux phases alternées, l'algorithme affecte des valeurs de vérité aux littéraux à la manière DPLL. Si un conflit est rencontré au cours de la recherche, alors l'algorithme explore la branche correspondant à la deuxième valeur de vérité de la variable représentant le point de choix courant uniquement si une clause $c_i \in C_{STB}$ est produite. Sinon, un rebroussement (backtrack) est effectué.

Une extension est trouvée quand toutes les clauses sont satisfaites ou bien quand tous les littéraux du STB ont été tous affectés sans falsifier aucune clause. Dans les deux cas, l'algorithme exécute une phase dite de "complétion". Dans le premier cas, la méthode complète l'interprétation courante par l'assignation à vrai de l'ensemble des variables $not\ A_i$ restant du STB et par l'assignation à faux de toutes les autres variables non encore affectés (hypothèse du monde clos). Dans le deuxième cas, la complétion consiste selon l'hypothèse du monde clos, à mettre à faux les variables non affectées. Dans les deux cas, un modèle minimal candidat est trouvé. L'algorithme vérifie alors si le modèle candidat est bien un modèle stable.

L'algorithme commence par un premier appel à la procédure propagation-unitaire qui propage tous les mono-littéraux jusqu'à ce que la liste de ces derniers $Lmono$ se vide. Puis un appel est fait pour traiter les littéraux purs qui à leur tour peuvent induire des mono-littéraux. Quand il n'y a plus de mono-littéraux ou des littéraux purs à assigner, l'algorithme essaye de produire une clause $c_i \in C_{STB}$ (proposition 2). Si on arrive à produire une clause $c_i \in C_{STB}$, alors la seconde branche du point de choix courant sera explorée. Si, par contre, aucune clause n'est produite et qu'il ne reste pas de mono-littéraux ou littéraux purs à propager, alors la seconde branche de la variable point de choix devient inutile et donc coupée. L'énumération continue par le choix du prochain littéral dans STB à assigner et ce processus est réitéré jusqu'à la satisfaction de toutes les clauses ou l'affectation de tous les littéraux du STB sans apparition de la clause vide. Dans ce cas, une extension $E = HC(\pi)_I$ est obtenue. Il ne reste plus qu'à effectuer la phase de complétion, ensuite vérifier si l' extension obtenue satisfait la condition dite discriminante et celle de maximalité en variable $not\ A_i$ pour induire un modèle stable. Le pseudo-code du schéma général de la méthode est donné dans l'algorithme 1.

Algorithm 1 Schéma général de la nouvelle méthode de recherche de modèles stables

Require: La forme clausale $l(\pi)$ d'un programme logique π
Ensure: Tous les modèles stables de π

$S = \emptyset$
repeat
  while $STB \neq \emptyset$ et 'pas de conflit' **do**
    while $Lmonos \neq \emptyset$ or $Lpure \neq \emptyset$ **do**
      unit-propagation($HC(\pi)$,$Lmonos$,I) ;
      inférence($HC(\pi)$,$Lpure$,I) ;
      clause-production($HC(\pi)$) ;
    end while
    choisir littéral ;
  end while
  if pas de conflit **then**
    $E = HC(\pi)_I$ est une extension candidate ;
    E= complétion(E) ;
    if Conditions(E) **then**
      M=Atomes-positive(E) ;
      $S = S \cup M$;
    end if
  else
    backtrack
  end if
until Tout l'espace de recherche est exploré

Si n est le nombre de variables de la représentation clausale $HC(\pi)$ du programme π, k le cardinal de l'ensemble STB et m le nombre de clauses de $HC(\pi)$, alors la complexité temporelle de l'algorithme dans le pire des cas est approximativement O($knm2^k$). Nous pouvons remarquer que le facteur exponentiel de la fonction de complexité dépend du nombre k représentant la taille de l'ensemble STB et ne dépend pas du nombre de variables n comme dans les autres solveurs ASP. La valeur de k est généralement plus petite que celle de n, d'où une meilleure complexité temporelle.

Contrairement à la plupart des solveurs ASP qui utilisent la complétion Clark avec la gestion des boucles et qui donc ont une complexité spatiale exponentielle dans le pire des cas, notre méthode fonctionne à espace constant. En effet, la méthode utilise la forme clausale $HC(\pi)$ dont la taille est identique à celle du programme initial π et qui ne varie pas pendant les exécutions. La complexité spatiale est constante, elle est d'ordre $O(\mid HC(\pi) \mid) = O(\mid \pi \mid)$ dans le pire des cas.

4 Définition et propriétés de la symétrie

La notion de symétrie est très largement étudié dans le domaine de la programmation par contraintes. Nous donnerons dans ce qui suit les principales définitions et propriétés de la symétrie dans le cadre de la représentation clausale $HC(\pi)$ d'un programme logique π. Nous définissons d'abord la symétrie sémantique :

Définition 3 *Soit $HC(\pi)$ la représentation clausale de π et $L_{HC(\pi)}$ l'ensemble des littéraux de $HC(\pi)$. Une symétrie sémantique σ de $HC(\pi)$ est une permutation définie sur l'ensemble $L_{HC(\pi)}$, tel que $HC(\pi)$ et $\sigma(HC(\pi))$ ont les mêmes modèles stables.*

En d'autres termes, une symétrie sémantique de la représentation clausale d'un programme logique est une permutation de ses littéraux qui préserve les modèles stables. Nous allons maintenant définir la symétrie syntaxique :

Définition 4 *Soit $HC(\pi)$ la représentation clausale de π et $L_{HC(\pi)}$ l'ensemble de littéraux de $HC(\pi)$. Une symétrie syntaxique σ de $HC(\pi)$ est une permutation définie sur $L_{HC(\pi)}$, tel que $HC(\pi) = \sigma(HC(\pi))$.*

En d'autres mots, une symétrie syntaxique de la représentation clausale d'un programme logique est une permutation de ses littéraux qui laisse toutes les clauses du programme inchangé.

Exemple 2 *On considère le programme logique π de l'exemple 1 où $L_{HC(\pi)} = \{a, b, not\, a, not\, b\}$. La permutation $\sigma = (a, b)(nota, notb)$ définie pour $L_{HC(\pi)}$ est une symétrie syntaxique de $HC(\pi)$ ($\sigma(HC(\pi)) = HC(\pi)$).*

Définition 5 *Les deux littéraux l et l' de $L_{HC(\pi)}$ sont symétrique s'il existe une symétrie σ de $HC(\pi)$ tel que $\sigma(l) = l'$.*

Nous allons définir l'orbite d'un littéral :

Définition 6 *Soit $HC(\pi)$ la représentation clausale de π, l'orbite du littéral $l \in L_{HC(\pi)}$ sur lequel le groupe de symétrie $(Sym(HC(\pi)), \star)$ est appliqué est $l^{Sym(HC(\pi))} = \{\sigma(l) : \sigma \in Sym(HC(\pi))\}$*

Nous donnons ci-dessous une propriété qui relie entre la symétrie syntaxique et la symétrie sémantique :

Proposition 3 *Chaque symétrie syntaxique de la représentation clausale $HC(\pi)$ est une symétrie sémantique de $HC(\pi)$.*

Preuve 3 *Il est trivial de voir qu'une symétrie syntaxique de $HC(\pi)$ est toujours une symétrie sémantique de $HC(\pi)$. En effet, si σ est une symétrie syntaxique de $HC(\pi)$, alors $\sigma(HC(\pi)) = HC(\pi)$, il en résulte que $HC(\pi)$ et $\sigma(HC(\pi))$ ont les mêmes modèles stables.*

Si I est un modèle stable de $HC(\pi)$ et σ une symétrie syntaxique, nous pouvons obtenir un autre modèle stable de $HC(\pi)$ en appliquant σ sur les littéraux qui apparaissent dans I. Formellement :

Proposition 4 *Soit σ une symétrie syntaxique de $HC(\pi)$, I est un modèle stable de π ssi $\sigma(I)$ est un modèle stable de π.*

Exemple 3 *Dans l'exemple 2, l'orbite du littéral* a *est* $a^{Sym(L_\pi)} = \{a, b\}$ *et celui du littéral* $not\, a$ *est* $not\, a^{Sym(L_\pi)} = \{not\, a, not\, b\}$. *Tous les littéraux d'une même orbite sont tous symétriques. Par exemple, dans l'exemple 1 il y a deux modèles stables symétriques de* $HC(\pi)$. *Le premier c'est* $M_1 = \{b\}$ *et le second c'est* $\sigma(M_1) = \{a\}$. *Ce sont des modèles stables symétriques de* $HC(\pi)$.

Si $(Perm(HC(\pi)), \star)$ désigne le groupe de permutations de $L_{HC(\pi)}$ et $Sym(L_{HC(\pi)}) \subset Perm(L_{HC(\pi)})$ le sous-ensemble de permutations de $L_{HC(\pi)}$ formant les symétries syntaxiques de $HC(\pi)$, alors $(Sym(HC(\pi)), \star)$ est un sous-groupe de $(Perm(HC(\pi)), \star)$ représentant le groupe des symétries de $HC(\pi)$.

5 Détection des symétries

La détection des symétries est basée sur la recherche d'automorphismes de graphes. Nous commençons par représenter l'ensemble de clause de Horn $HC(\pi)$ par un graphe coloré $G_{HC(\pi)}$ puis calculer son ensemble d'automorphismes qui devrait être identique au groupe des symétries de $HC(\pi)$. Cette technique est très connue en logique propositionnelle [8, 1].

Soit $G_{HC(\pi)} = (V, E)$ le graphe associé à $HC(\pi)$, où V est un ensemble de sommets et $E \subseteq V \times V$ un ensemble d'arêtes. Un automorphisme (symétrie) de $G_{HC(\pi)}$ est une permutation des sommets qui laisse le graphe inchangé. Seul les sommets ayant la même couleur peuvent être permutés ensemble. Soit $HC(\pi)$ la représentation clausale de π, le graphe coloré associé $G_{HC(\pi)}(V, E)$ de $HC(\pi)$ est définit comme suit :

- Chaque littéral positive A_i de $HC(\pi)$ est représenté par un sommet $A_i \in V$ de couleur 1 dans $G_{HC(\pi)}$. Le littéral négatif $\neg A_i$ associé à A_i est aussi représenté par un sommet $\neg A_i$ de couleur 1 dans $G_{HC(\pi)}$. Ces deux sommets sont reliés par une arête de E dans le graphe $G_{HC(\pi)}$.
- Chaque littéral $not\, A_i \in STB$ associé à A_i est représenté par un sommet $not\, A_i$ de couleur 2 dans $G_{HC(\pi)}$. Ce sommet est connecté a celui représentant A_i par une arête de E dans le graphe $G_{HC(\pi)}$.
- Chaque littéral $not\, A_i \notin STB$ associé à A_i est représenté par un sommet $not\, A_i$ de couleur 3 dans $G_{HC(\pi)}$. Ce sommet est connecté a celui représentant A_i par une arête de E dans le graphe $G_{HC(\pi)}$.
- Chaque clause représentant une règle $c_i \in CR$ de $HC(\pi)$ est représenté par un sommet $c_i \in CR$ de couleur 4 dans $G_{HC(\pi)}$. Une arête connecte ce sommet c_i à ceux représentant les sommets de ses littéraux.
- Chaque clause représentant une exclusion mutuelle $c_i \in ME$ de $HC(\pi)$ est représenté par un sommet $c_i \in V$ de couleur 5 dans $G_{HC(\pi)}$. Une arête connecte ce sommet c_i aux deux sommets représentant ses littéraux.

Cette construction de graphe garantit que seuls les sommets ayant la même couleur pourraient être permutés ensemble. Le graphe $G_{HC(\pi)}$ préserve le groupe de symétries syntaxique de $HC(\pi)$. C'est à dire, le groupe de symétrie syntaxique de la représentation $HC(\pi)$ d'un programme logique π est identique au groupe d'automorphisme de son graphe $G_{HC(\pi)}$. Nous pourrions utiliser ensuite un système de détection d'automorphisme comme $saucy$, $autom$ ou $nauty$ pour détecter le groupe de symétrie syntaxique de $HC(\pi)$. Ces systèmes retournent un ensemble de générateurs du groupe de symétrie à partir desquelles nous pouvons déduire chaque symétrie de $HC(\pi)$. Notre approche est différente de celle présentée dans [11]. Cette méthode utilise la représentation body-atom d'un programme logique pour encoder un graphe orienté. Elle est également différente de la technique introduite dans [5] qui utilise un solveur ASP basé sur l'approche SAT et sur la complétion de Clark.

6 Elimination des symétries

Soit un programme logique π, les symétries de sa représentation clausale $HC(\pi)$ induisent des classes d'équivalence dans l'espace de solutions / no-goods du programme. Toutes les interprétations symétriques d'une solution / no-good I de π sont des solutions / no-goods de π. La symétrie est alors une relation d'équivalence qui partitionne l'ensemble des interprétations. Il est alors possible de représenter chaque classe d'équivalence par une interprétation représentante. Ici nous utilisons la technique du *lex leader* [2] qui ordonne tous les littéraux de $HC(\pi)$ selon un ordre lexicographique, et sélectionne la plus petite interprétation lexicographique de chaque classe d'équivalence. Soit $I = \{not\, A_1, not\, A_2, ...not\, A_n\}$ l'ensemble STB ordonné lexicographiquement. Les prédicats d'élimination des symétries (SBPs) sont construits en fonction du groupe de symétrie détecté $Sym(HC(\pi)) = \{\sigma 1, \sigma 2, ..., \sigma_n\}$ de $HC(\pi)$ avec l'ordre suivant $not\, A_1 \leq not\, A_2 \leq ... \leq not\, A_n$ sur les variables $not\, A_i$ du STB. Nous générons des SBP partielle qui élimine les symétries correspondant aux générateurs du groupe de symétrie. Les SBP partielles sont données par l'ensemble de formules suivante :

$$PC(\sigma) = \left[\bigwedge_{1 \leq k \leq m} (p_{k-1} \rightarrow (not\, A_k \leq not\, A_k^\sigma)) \right] \wedge \left[\bigwedge_{1 \leq k \leq m-1} (p_{k-1} \rightarrow (not\, A_k \geq not\, A_k^\sigma) \rightarrow p_k) \right]$$

$PC(\sigma)$ est le prédicat de permutation correspondant au générateur de symétrie σ où p_i est une variable auxiliaire définie comme suit :

$$p_i = g_{prev(i,I)} \rightarrow \bigwedge_{k \in K} \left[\bigwedge_{j \in prev(k,K)} g_j \right] \rightarrow l_k$$

Nous avons deux prédicats d'ordre $l_i = (not\, A_i \leq not\, A_i^\sigma)$ et $g_i = (not\, A_i \geq not\, A_i^\sigma)$. La fonction $g_{prev(i,I)}$ donne le prédécesseur de la variable auxiliaire p_i dans l'ensemble I. C'est à dire, $g_{prev(i,I)} = p_{i-1}$.

$$PLL(Sym(HC(\pi)) = \bigwedge_{\sigma \in Gen(Sym(HC(\pi))} PC(\sigma)$$

$PLL(Sym(HC(\pi))$ représente la formule d'élimination partielle des symétries.

7 Expérimentation

Sur la base de l'algorithme présenté précédemment, nous avons implémenté une première version d'un nouveau solveur ASP que nous désignons par $HC-asp$ pour signifier Horn Clause ASP. Nous avons enrichi ce système en lui rajoutant l'élimination des symétries pour obtenir la variante $HC-asp-sym$. La méthode d'élimination des symétries utilisée ici est une approche statique qui élimine les symétries dans une phase de prétraitement. Notre système ASP prend en entrée un programme logique terminal π produit par le grounder $gringo$ [14]. Ce programme logique est exprimé dans sa forme clausale $HC(\pi)$. Le système construit un graphe coloré $G_{HC(\pi)}$ qui représente $HC(\pi)$ que $saucy$ [9] utilise pour détecter le groupe des symétries de $HC(\pi)$. Les prédicats d'élimination partielle des symétries sont calculés en fonction du groupe des symétries détectées puis ajoutés au codage $HC(\pi)$. Après cela, un solveur ASP pourrait être utilisé en tant que boîte noire sur l'ensemble de clauses résultant.

Pour montrer l'efficacité de notre solveur $HC - asp$, nous l'avons comparé à d'autres systèmes ASP existants. Nous avons considéré dans la comparaison le solveur $Cmodels$ (version 3.86 avec $zChaff$ comme solveur SAT). Nous avons également considéré deux autres solveurs ASP connus qui sont $Smodels$ (version 2.34) et $Clasp$ (version 3.3.3). Tous les systèmes sont appliqués pour rechercher tous les modèles stables et $gringo$ est utilisé comme grounder pour chacun d'eux. Les programmes fonctionnent sur une machine Ubuntu (16,10) de 4 Go avec un processeur Intel Core i5 (1,70 GHz x 4). La durée d'exécution est limitée à 24H pour tous les systèmes expérimentés. Nous avons expérimenté différents problèmes hautement combinatoires. Pour chacun d'entre eux, nous avons progressivement augmenté la taille du problème. Le nombre de modèles stables obtenus par tous les systèmes s'ils terminent l'exécution sont donné dans la colonne "Modèles stables" des Tableaux 1,2 et 3. Nous rapportons ici les résultats obtenus sur quelques benchmarks connus qui sont le problème d'accessibilité, le problème des Pigeons, le problème de Ramsey, le problème des n-reines, le circuit hamiltonien et le problème de Schur. La tâche la plus importante dans l'ASP consiste à énumérer tous les modèles stables d'un programme logique. Nous avons choisi ces benchmarks en raison de leur nombre important de modèles stables. Ils sont très appropriés pour étudier le comportement de chacun des solveurs lorsque le nombre de modèles stables et la taille du problème augmentent. Les tableaux 1, 2 et 3 donnent le temps d'exécution des différents benchmarks pour tous les solveurs ASP. Ils donnent également des informations sur le système $HC-asp-sym$: #sym représente le nombre de symétries détectée, #SBP le nombre de SBPs ajoutés et #nssmodels le nombre de modèles sables non symétriques. Le temps d'exécution est celui de l'énumération de toutes les solutions incluant le pré-traitement sur la symétrie.

Les résultats obtenus pour la recherche de tous les modèles stables des quatre benchmarks Accessibilité(1-4), Pigeons(9-16), Ramsey(4-8) et le problème de Schur (17-28) sont dans le tableau 1. En général, $HC - asp$ surpasse tous les autres systèmes ASP. Nous pouvons observer que l'usage de l'élimination des symétries réduit considérablement l'espace des solutions de tous les benchmarks, et de ce fait, réduit le temps d'exécution. C'est à dire, $HC - asp - sym$ a de meilleurs résultats que ceux de $HC - asp$ et ceux des autres systèmes. Clairement, nous pouvons voir que l'élimination des symétries réduit significativement le nombre de modèles stables trouvés. Le gain augmente lorsque la taille du problème et le nombre de modèles stables augmentent aussi. Le problème des pigeons est une bonne illustration de cette observation. Pour le problème des pigeons de taille 9 à 11, l'espace des solutions est compressé par environ 95%. Cela réduit considérablement le temps d'exécution. Pour le problème de Ramsey et celui de Schur, nous pouvons voir que $HC - asp$, $Clasp$, et $Smodels$ ont des résultats comparable. De nouveau, $HC - asp - sym$ a de meilleurs résultats sur ces problèmes.

L'autre benchmark que nous avons expérimenté est celui des n-reines. Les résultats obtenus sont présentés dans le tableau 2. Nous pouvons voir que toutes les méthodes ont résolu efficacement les petites instances (10 à 12 reines). $HC - asp$ surpasse de manière significative toutes les autres méthodes lorsque la taille du problème augmente. L'élimination des symétries a grandement contribué à améliorer les résultats de $HC - asp$. Le nombre de solutions trouvés et le temps consacré à l'exploration de tout l'espace de recherche sont réduits quand on élimine les symétries dans $HC - asp - sym$.

Le dernier benchmark est celui du circuit hamiltonien [23]. Nous avons trouvé tous les circuits hamiltoniens de quelques graphes orientés complets dont le nombre de sommets varie de 5 à 12.Les comportements de tous les solveurs sont représentés dans le tableau 3. Les résultats

TABLE 1 – Les résultats obtenus sur les problèmes d'Accessibilité, Ramsey, Pigeons et Schur

N°	Instances	#Modèles stables					HC-asp-sym		
			HC-asp	Clasp	Smodels	Cmodels	#Sym	#SBP	#nssmodels	Temps
1	R_2	1	0.0005	0.0001	0.0002	0.0003	1	6	1	0.0002
2	R_3	18	0.0015	0.001	0.0005	0.015	2	23	9	0.0004
3	R_4	1606	0.030	0.070	0.022	0.091	2	54	725	0.011
4	R_5	565080	7.12	12.59	7.62	9991.28	3	134	176733	2.94
5	R_4_5_5	957	0.011	0.010	0.014	0.049	3	32	232	0.003
6	R_4_5_6	27454	0.2	0.5	0.33	18.62	5	71	6021	0.054
7	R_4_5_7	1452289	12.64	28.76	18.00	•	5	99	316803	3.34
8	R_4_5_8	137578233	1625.14	3329.66	2219.19	•	6	155	12365132	167.71
9	pi4/4	24	0.007	0.009	0.002	0.003	4	84	6	0.0005
10	pi5/5	120	0.02	0.02	0.008	0.01	5	147	12	0.0011
11	pi6/6	720	0.06	0.06	0.04	0.07	6	189	70	0.0048
12	pi7/7	5040	0.23	0.55	0.30	0.87	7	249	840	0.069
13	pi8/8	40320	1.65	4.01	2.5	52.34	8	336	2424	0.42
14	pi9/9	362880	21.63	47.11	34.60	4591.87	9	702	19634	1.26
15	pi10/10	3628800	210.14	494.40	369.80	•	10	846	170354	19.92
16	pi11/11	39916800	2728.53	6936.96	4247.58	•	11	1044	893760	55.77
17	Schur4/10	2964	0.16	0.12	0.10	0.36	8	158	325	0.0005
18	Schur4/11	8326	0.47	0.36	0.28	1.33	8	204	847	0.0064
19	Schur4/12	18539	0.98	0.82	0.65	4.82	9	272	1687	0.025
20	Schur4/13	48987	2.6	3.26	1.77	39.04	9	350	2358	0.5
21	Schur4/14	95311	5.29	5.7	4.43	169.85	10	402	5863	1.3
22	Schur4/15	247147	14.20	15.61	12.36	1286.12	10	498	10354	3.68
23	Schur4/16	408642	26.22	27.58	24.10	3500.16	11	535	20789	6.8
24	Schur4/17	920149	65.61	66.84	49.84	15976.8	11	602	25893	10.23
25	Schur4/18	1597576	134.87	137.89	102.88	48134.4	12	647	45283	20.34
26	Schur4/19	3344347	287.85	301.96	207.86	•	12	732	60352	36.22
27	Schur4/20	3832609	406.88	436.88	292.91	•	13	802	65821	80.34
28	Schur4/21	7924530	965.80	987.62	707.90	•	13	897	92358	120.85

TABLE 2 – Les résultats obtenus sur le problèmes des n-reines

#Taille	#Modèles stables					HC-asp-sym			
		HC-asp	Clasp	Smodels	Cmodels	#Sym	#SBP	#sol	Temps
10	724	0.68	0.23	0.66	0.27	3	468	240	0.10
11	2680	2.79	1.02	2.95	2.48	3	528	852	0.56
12	14200	12.2	8.75	15.19	41.44	3	810	4981	3.69
13	73712	79.55	122.91	87.69	1642.93	3	921	24934	20.53
14	365596	371.73	2631.83	496.98	•	3	918	107371	120.26
15	2279184	2797.02	34337.21	3352.37	•	3	1221	788141	854.31
16	14772512	12087.40	•	23134.22	•	3	1188	4310853	4976.44
17	95815104	87088.00	•	•	•	3	1275	29227320	29429.41

TABLE 3 – Les résultats obtenus sur le problème du circuit hamiltonien

#Taille	#Modèles stables					HC-asp-sym			
		HC-asp	Clasp	Smodels	Cmodels	#Sym	#SBP	#sol	Temps
5	24	0.012	0.003	0.003	0.003	3	90	6	0.0005
6	120	0.026	0.016	0.014	0.02	3	132	48	0.0031
7	720	0.09	0.10	0.08	0.12	4	186	288	0.014
8	5040	0.64	0.78	0.67	1.69	4	246	1992	0.13
9	40320	5.76	7.87	6.00	80.75	5	348	5040	0.64
10	362880	66.67	89.94	72.25	6177.89	5	396	40320	6.63
11	3628800	910	1141.84	964.15	•	6	591	412118	107.26
12	39916800	13173.42	22263.37	15964.15	•	6	612	7327392	4604.26

montrent que toutes les méthodes ont résolu de manière efficace les instances ayant un nombre de sommets inférieur à huit (petites instances). $HC-asp-sym$ obtient de meilleurs résultats que les autres méthodes. Le bénéfice de la symétrie est plus important lorsque la taille du problème augmente.

8 Conclusion

Dans cet article, nous avons fourni une nouvelle méthode pour rechercher des modèles stables basée sur une sémantique relativement nouvelle introduite dans [6]. Cette méthode a l'avantage d'utiliser une représentation en clause de Horn dont la taille est identique à celle du programme logique source. Elle a une complexité spatiale constante. La sémantique utilisée prévient la méthode de la lourdeur induite par la gestion des boucles faite par tous les solveurs basé sur la complétion de Clark. L'autre avantage de notre approche est le processus énumératif simplifié qui est effectué uniquement sur un sous-ensemble de littéraux représentant le strong backdoor du programme logique. Cela conduit à une réduction considérable de la complexité temporelle. Nous avons également proposé l'intégration

de l'élimination des symétries afin d'éviter d'explorer des sous-espaces isomorphes. Nous avons expérimenté la méthode proposée avec et sans élimination des symétries sur une variété de problèmes combinatoires connus et les résultats obtenus ont montré que notre approche est une bonne alternative pour implémenter des solveurs ASP et que l'élimination des symétries conduit à une amélioration significative de cette méthode.

Nous envisageons par la suite d'intégrer l'élimination des symétries de manière dynamique et comparer par rapport à l'approche statique. L'extension de notre approche à d'autres classes de programmation logique ou à des parties de certaines logiques non monotones plus générales est aussi envisageable.

Références

[1] F.A Aloul, A. Ramani, I.L. Markov, and K.A. Sakallah. Solving difficult sat instances in the presence of symmetry. *DAC*, pages 731–736, 2002.

[2] F.A Aloul, A. Ramani, I.L. Markov, and K.A. Sakallah. Solving difficult sat instances in the presence of symmetry. *DAC*, pages 1117–1137, 2003.

[3] B. Benhamou and L. Sais. Theoretical study of symmetries in propositional calculus and application. *CADE*, 607 :281–294, 1992.

[4] B. Benhamou and L. Sais. Tractability through symmetries in propositional calculus. *The Journal of Automated Reasoning*, pages 89–102, 1994.

[5] Belaid Benhamou. Dynamic and static symmetry breaking in answer set programming. *LPAR*, pages 112–126, 2013.

[6] Belaïd Benhamou and Pierre Siegel. A new semantics for logic programs capturing and extending the stable model semantics. *Tools with Artificial Intelligence (ICTAI)*, pages 25–32, 2012.

[7] Keith L Clark. Negation as failure. *Logic and data bases*, pages 293–322, 1978.

[8] James M. Crawford, Matthew L. Ginsberg, Eugene M. Luks, and Amitabha Roy. Symmetry-breaking predicates for search problems. *KR*, pages 148–159, 1996.

[9] P.T Darga, K.A. Sakallah, and I.L. Markov. Faster symmetry discovery using sparsity of symmetries. *DAC*, pages 149–154, 2008.

[10] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Communications of the ACM*, 5 :394–397, 1962.

[11] Christian Drescher, Oana Tifrea, and Toby Walsh. Symmetry-breaking answer set solving. *AI Communications*, 24 :177–194, 2011.

[12] Francois Fages. Consistency of clark's completion and existence of stable models. *Methods of Logic in Computer Science*, 1 :51–60, 1994.

[13] Martin Gebser, Benjamin Kaufmann, Andrá Neumann, and Torsten Schaub. Conflict-driven answer set solving. *IJCAI*, 7 :386–392, 2007.

[14] Martin. Gebser, Torsten Schaub, and Sven Thiele. Gringo : A new grounder for answer set programming. *International Conference on Logic Programming and Nonmonotonic Reasoning*, 7 :266–271, 2007.

[15] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. *ICLP/SLP*, 50 :1070–1080, 1988.

[16] Tarek Khaled, Belaid Benhamou, and Pierre Siegel. Vers une nouvelle méthode de calcul de modéles stables et extensions en programmation logique. *JIAF*, pages 151–160, 2017.

[17] B. Krishnamurty. Short proofs for tricky formulas. *Acta Inf.*, 22 :253–275, 1985.

[18] Nicola. Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The dlv system for knowledge representation and reasoning. *ACM Transactions on Computational Logic (TOCL)*, 7 :499–562, 2006.

[19] Yuliya Lierler and Marco Maratea. Cmodels-2 : Sat-based answer set solver enhanced to non-tight programs. *Logic Programming and Nonmonotonic Reasoning*, pages 346–350, 2004.

[20] Vladimir Lifschitz and Alexander Razborov. Why are there so many loop formulas ? *ACM Transactions on Computational Logic (TOCL)*, 7 :261–268, 2006.

[21] Fangzhen Lin and Yuting Zhao. Assat : Computing answer sets of a logic program by sat solvers. *Artificial Intelligence*, pages 115–137, 2004.

[22] B. McKay. Practical graph isomorphism. *Numerical mathematics and computing.*, pages 45–87, 1981.

[23] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25 :241–273, 1991.

[24] J.F. Puget. Automatic detection of variable and value symmetries. *CP*, pages 475–489, 2005.

[25] Patrik Simons, Ilkka Nimelä, and Timo Soininen. Extending and implementing the stable model semantic. *Artificial Intelligence*, 138 :181–234, 2002.

[26] Ryan Williams, Carla P Gomes, and Bart Selman. Backdoors to typical case complexity. *International joint conference on artificial intelligence*, 18 :1173–1178, 2003.

Une approche SAT incrémentale pour le problème de satisfiabilité minimale en logique modale S5

Jean-Marie Lagniez, Daniel Le Berre, Tiago de Lima et Valentin Montmirail *

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

Résumé

De récents travaux sur la satisfiabilité en logique modale S5 ont montrés qu'utiliser SAT surpasse les approches existantes. Ici, nous allons plus loin en étudiant le problème de la satisfiabilité minimale en logique modale S5 (MinS5-SAT), c'est-à-dire trouver un modèle S5 avec le plus petit nombre de mondes possibles. Il peut être résolu de manière standard avec un solveur MaxSAT ou un solveur d'optimisation pseudo-booléen. Nous montrons que c'est, dans notre cas, équivalent à l'extraction d'un ensemble maximal consistant (MSS). Nous montrons qu'une nouvelle approche incrémentale, basée sur SAT, peut être proposée pour prendre en compte les relations d'équivalences entre les mondes possibles dans un modèle S5. Cette approche spécialisée présente de meilleures performances sur les benchmarks face aux solveurs considérés. Nos résultats démontrent que la connaissance du domaine est la clé pour construire un outil efficace basé sur SAT.

Abstract

Recent work on the practical aspects on the modal logic S5 satisfiability problem showed that using a SAT-based approach outperforms other existing approaches. Here, we go one step further and study the related minimal S5 satisfiability problem (MinS5-SAT), the problem of finding an S5 model with the smallest number of possible worlds. It can be solved using a standard pseudo-Boolean or MaxSAT solver. We show in this paper that it is also equivalent to the extraction of a maximal satisfiable set (MSS). We show that a new incremental, SAT-based approach can be proposed by taking into account the equivalence relation between the possible worlds on S5 models. That specialized approach presented the best performance on our benchmarks compared to the solvers considered. Our results demonstrate once again that domain knowledge is key to build efficient SAT-based tools.

*Papier doctorant : Valentin Montmirail est auteur principal.

1 Introduction

Au cours des vingt dernières années, les logiques modales ont été utilisées dans divers domaines de l'intelligence artificielle comme la vérification formelle [12], les bases de données [13] et l'informatique distribuée [34]. Plus récemment, la logique modale S5 a été manipulée pour de la compilation de connaissances [3] et utilisée pour la planification contingente [28]. Différents solveurs ont été conçus pour les logiques modales pour décider la satisfiabilité depuis les années 90 [9, 33], certains assez récemment [27, 5]. Malgré la variété des techniques employées, aucun solveur, à notre connaissance, ne garantit que lorsqu'un modèle est trouvé, il est l'un des plus petits modèles possibles (en nombre de mondes). La plupart ne retournent pas le modèle.

Fournir un modèle, un certificat de satisfiabilité, est important pour vérifier la réponse donnée par le solveur [20]. Ceci est vrai, autant pour l'auteur du solveur que pour un utilisateur de celui-ci. Vérifier les modèles est obligatoire de nos jours dans beaucoup de compétitions, parmi celles-ci la compétition SAT [31]. Il a également été démontré que ces modèles peuvent aider à améliorer les procédures basées sur un oracle NP [24] : une procédure nécessitant un nombre polynomial d'appels à un "oracle vrai/faux" peut être transformée en une procédure nécessitant seulement un nombre logarithmique d'appels lorsque l'oracle fournit un modèle. Un autre exemple de l'importance pour un oracle de fournir un modèle peut être trouvé dans la technique de rotation de modèles [2], une méthode de détection des clauses incluses dans tous les MUS d'une formule donnée via l'analyse de modèles renvoyés par un oracle SAT. Même si la théorie ne garantit pas une réduction du nombre d'appels à l'oracle, en pratique, elle produit un énorme gain de performance (jusqu'à un facteur 5). Trouver le plus petit modèle peut être encore plus important dans certains contextes. Le modèle fourni a généralement de l'informa-

tion à l'utilisateur, comme en vérification de matériel [12] où le modèle est en fait une explication du problème trouvé dans la conception. Plus le modèle est petit, plus l'emplacement du problème est précis. Il se peut également que le modèle soit analysé par l'utilisateur ou affiché sur un écran. Ainsi, la taille est primordiale. Il existe une littérature sur la minimisation des modèles pour SAT [17, 32]. Nous nous intéressons ici à minimiser les modèles pour S5-SAT.

Notre objectif dans cet article est de proposer des techniques pour calculer le plus petit modèle S5, en nombre de mondes, pour une formule donnée. Nous appelons ce problème MinS5-SAT. Nous nous concentrons exclusivement ici sur la logique modale S5, pour lequel le problème de satisfiabilité a la particularité d'être NP-complet [19], comme le problème de satisfiabilité en logique propositionnelle (SAT). Nous proposons et comparons ici différentes techniques : (1) La première technique est basée sur une traduction de la formule d'entrée en logique propositionnelle. Le paramètre donné à la traduction est le nombre n de mondes possibles que le modèle est supposé avoir. Une recherche linéaire ou dichotomique peut être utilisée pour minimiser le modèle S5 ; (2) La deuxième technique ajoute des variables-sélecteurss au codage précédent pour activer ou désactiver les mondes. Trouver un modèle S5 minimal dans ce cas équivaut à un problème MaxSAT [4], ou, plus surprenant, à un problème d'extraction de MSS. Ainsi, nous pouvons compter sur des solveurs MaxSAT ou des extracteurs de MSS pour résoudre le problème original ; (3) La dernière technique va plus loin que les deux approches précédentes, grâce à une propriété spécifique de la logique modale S5. Nous interprétons l'ensemble des variables-sélecteurss provoquant l'incohérence de la formule pour atteindre le nombre de mondes optimal plus rapidement. Nous comparons ces différentes techniques et montrons empiriquement laquelle convient le mieux à l'ensemble d'instances testées. Cet ensemble est composé d'instances aléatoires ou conçues spécialement (*crafted*). Cependant, nous savons grâce à la communauté SAT [31] que la performance d'un solveur peut être significativement différente lorsque le problème est généré de façon aléatoire, quand le problème est *crafted* ou quand il modélise un problème réel qui possède une structure. Ainsi, nous avons généré de nouveaux benchmarks S5, traduits de problèmes de planification avec des informations incomplètes sur l'état initial [11], afin de tester aussi nos approches sur des problèmes structurés.

Pour résumer cet article, nous présentons d'abord la logique modale S5 et définissons le problème MinS5-SAT. Ensuite, nous fournissons une première approche pour résoudre MinS5-SAT en utilisant un oracle SAT et des variables-sélecteurss. Nous fournissons une traduction du problème MinS5-SAT en un problème équivalent d'extraction de MSS. Nous présentons une propriété de la logique modale S5 qui accélère notre approche basée sur SAT.

2 Préliminaires

2.1 La logique modale S5

Un problème central associé à toute logique est le problème de satisfiabilité, c'est-à-dire le problème de décider si une formule donnée possède un modèle. Les premiers résultats de complexité pour la satisfiabilité en logiques modales ont été obtenus par Ladner [19]. Il a montré que le problème de satisfiabilité dans la logique modale K (K-SAT) est dans PSPACE et que le problème de satisfiabilité dans la logique modale S5 (S5-SAT) est NP-complet. Dans cet article, nous nous intéressons à cette dernière. Dans ce qui suit, soit $\mathbb{P}$ un ensemble infini dénombrable non vide de variables propositionnelles. Le langage $\mathcal{L}$ de la logique modale est l'ensemble des formules ϕ définies par la grammaire BNF suivante, où p varie sur $\mathbb{P}$:

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

Les opérateurs $\rightarrow$ et $\leftrightarrow$, définis par les abréviations usuelles, sont également utilisés. Une formule du type $\Box\phi$ signifie que 'ϕ est nécessairement vraie'. Une formule du type $\Diamond\phi$ signifie que 'ϕ est possiblement vraie'.

Exemple 1. Soit $\mathbb{P} = \{a, b\}$. $\phi = ((\Box\neg a \vee \Diamond b) \wedge \Diamond a \wedge \Box b)$ est une formule de logique modale.

Les formules dans $\mathcal{L}$ sont interprétées en utilisant des structures S5 [18], qui sont définies comme suit :

Définition 1 (Structure S5)**.** Une structure S5 est un triplet $M = \langle W, R, \mathcal{I}\rangle$, où : W est un ensemble non-vide de mondes possibles ; R est une relation binaire sur W qui est une relation d'équivalence ($\forall w.\forall v.\ (w, v) \in R$) ; $\mathcal{I}$ est une fonction associant, à chaque $p \in \mathbb{P}$, l'ensemble des mondes de W où p est vraie.

Notez que comme R est une relation d'équivalence, nous l'omettrons dans le reste de cet article.

Définition 2 (Structure S5 pointée)**.** Une structure S5 pointée est une paire $\langle M, w\rangle$, où M est une structure S5 et w, appelé le monde courant, est un monde possible dans W.

Par souci de simplicité, nous désignerons par 'structure' une 'structure S5 pointée'. Nous définissons la taille d'une structure $\langle M, w\rangle$, notée $|M|$, comme son nombre de mondes possibles. Ci-dessous, nous définissons la relation de satisfiabilité entre de telles structures et des formules dans $\mathcal{L}$.

Définition 3 (Relation de satisfiabilité)**.** La relation de satisfiabilité $\models$ entre les formules de logique modale et les structures $M = \langle W, R, \mathcal{I}\rangle$ est récursivement définie comme

suit :

$\langle M, w\rangle \vDash \top$

$\langle M, w\rangle \vDash p$ ssi $w \in \mathcal{I}(p)$

$\langle M, w\rangle \vDash \neg\phi$ ssi $\langle M, w\rangle \nvDash \phi$

$\langle M, w\rangle \vDash \phi \wedge \psi$ ssi $\langle M, w\rangle \vDash \phi$ et $\langle M, w\rangle \vDash \psi$

$\langle M, w\rangle \vDash \phi \vee \psi$ ssi $\langle M, w\rangle \vDash \phi$ ou $\langle M, w\rangle \vDash \psi$

$\langle M, w\rangle \vDash \Box\phi$ ssi pour tout $v \in W$ nous avons $\langle M, v\rangle \vDash \phi$

$\langle M, w\rangle \vDash \Diamond\phi$ ssi il existe $v \in W$ tel que $\langle M, v\rangle \vDash \phi$

Définition 4 (Satisfiabilité)**.** Une formule ϕ est satisfiable si et seulement s'il existe une structure $\langle M, w\rangle$ qui satisfait ϕ. Une telle structure est appelée un "modèle de ϕ".

Exemple 2. Soit une structure $\langle M, w_0\rangle$ satisfaisant la formule ϕ de l'exemple 1 : $W = \{w_0, w_1, w_2\}$, $\mathcal{I} = \{\langle a, \{w_0\}\rangle, \langle b, \{w_0, w_1, w_2\}\rangle\}$. La taille de $\langle M, w_0\rangle$ est 3.

Comme S5-SAT est NP-complet [19], nous avons proposé une réduction de ce problème à SAT dans [5]. La fonction de réduction, notée "tr", prend comme paramètre le nombre de mondes n et est définie comme suit :

Définition 5 (Fonction de traduction "tr")**.** Soit $\phi \in \mathcal{L}$.

$$\mathrm{tr}(\phi, n) = \mathrm{tr}'(\phi, 1, n) \quad \mathrm{tr}'(p, i, n) = p_i$$
$$\mathrm{tr}'(\neg\psi, i, n) = \neg\,\mathrm{tr}'(\psi, i, n)$$
$$\mathrm{tr}'(\psi \wedge \chi, i, n) = \mathrm{tr}'(\psi, i, n) \wedge \mathrm{tr}'(\chi, i, n)$$
$$\mathrm{tr}'(\psi \vee \chi, i, n) = \mathrm{tr}'(\psi, i, n) \vee \mathrm{tr}'(\chi, i, n)$$
$$\mathrm{tr}'(\Box\psi, i, n) = \bigwedge_{j=1}^{n} (\mathrm{tr}'(\psi, j, n))$$
$$\mathrm{tr}'(\Diamond\psi, i, n) = \bigvee_{j=1}^{n} (\mathrm{tr}'(\psi, j, n))$$

La fonction préserve la satisfiabilité pour un n est assez grand (voir [5] pour plus de détails). L'intuition derrière cette réduction est la suivante : comme tous les mondes sont accessibles, le fait que ψ soit vraie dans tous les mondes ($\Box\psi$) peut être vu comme une conjonction et le fait que ψ soit vraie dans au moins un monde ($\Diamond\psi$) peut être vu comme une disjonction.

Exemple 3. Soit ϕ la formule de l'exemple 1. Sa traduction $\mathrm{tr}(\phi, 2)$ est $((\neg a_1 \vee b_1 \vee b_2) \wedge (\neg a_2 \vee b_1 \vee b_2) \wedge (a_1 \vee a_2) \wedge (b_1 \wedge b_2))$.

2.2 Noyau incohérent

Les solveurs SAT récents sont incrémentaux, c'est-à-dire qu'ils sont capables de vérifier la satisfiabilité d'une formule sous hypothèses [1] et sont capables de fournir un noyau (une raison de l'incohérence de la formule). L'utilisation des noyaux incohérents est la clé de nombreuses applications, telles que MaxSAT, MCS ou MUS par exemple. Un noyau incohérent est défini comme suit :

Définition 6 (Noyau incohérent sous hypothèse)**.** Soit Σ une formule satisfiable en Forme Normale Conjonctive (CNF) construite en utilisant les variables booléennes de $\mathbb{P}$. Soit A un ensemble cohérent de littéraux construits en utilisant des variables booléennes à partir de $\mathbb{P}$, de telle sorte que $(\Sigma \wedge \bigwedge_{a\in A} a)$ soit incohérent. $C \subseteq A$ est un noyau incohérent (UNSAT core) de Σ sous hypothèses A si et seulement si $(\Sigma \wedge \bigwedge_{c\in C} c)$ est incohérent.

Définition 7 (Solveur SAT sous hypothèses)**.** Soit Σ une formule en CNF. Un solveur SAT pour Σ, étant donné les hypothèses A, est une procédure qui fournit une paire $\langle r, s\rangle$ avec $r \in \{\mathrm{SAT}, \mathrm{UNSAT}\}$ tel que si $r = \mathrm{SAT}$ alors s est un modèle de Σ, sinon si $r = \mathrm{UNSAT}$ alors s est un noyau incohérent de Σ sous hypothèses A.

2.3 MSS et co-MSS

Le problème du calcul d'un MSS (Maximal Satisfiable Set) consiste à extraire un ensemble maximal de clauses d'une formule en CNF qui sont cohérentes ensemble. Le sous-ensemble de correction minimale (MCS ou co-MSS) est le complément de son MSS. Pour simplifier les définitions, nous considérons les CNF comme des ensembles de clauses.

Définition 8. Soit une formule incohérente Σ en CNF. $S \subseteq \Sigma$ est un Maximal Satisfiable Subset (MSS) de Σ si et seulement si S est satisfiable et $\forall c \in \Sigma \setminus S$, $S \cup \{c\}$ est incohérent.

Définition 9. Soit une formule incohérent Σ en CNF. $C \subseteq \Sigma$ est un Minimal Correction Subset (MCS ou co-MSS) de Σ si et seulement si $\Sigma\backslash C$ est satisfiable et $\forall c \in C, \Sigma\backslash(C\backslash\{c\})$ est incohérent.

3 Le problème MinS5-SAT

Comme indiqué dans [5], le nombre nécessaire de mondes pour S5-satisfaire une formule en NNF (Forme Normale Négative) est borné par $\mathrm{dd}(\phi) + 1$, où $\mathrm{dd}(\phi)$ est donné dans la Définition 10 ci-dessous.

Définition 10 (Diamond-Degree)**.** Le *diamond degree* de $\phi \in \mathcal{L}$, noté $\mathrm{dd}(\phi)$, est défini de manière récursive, comme suit :

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

Ainsi, nous avons que $\mathrm{tr}(\phi, \mathrm{dd}(\phi) + 1)$ est équisatisfiable à ϕ. Même si, en pratique, nous obtenons de très bons résultats en utilisant cette valeur comme borne supérieure,

elle semble loin de la valeur optimale dans tous les cas. Par exemple, dans le modèle de l'exemple 1, on peut voir des redondances : deux mondes contiennent la même évaluation. Dans les contextes où la taille du modèle retourné est critique, il est logique de vouloir la minimiser.

Par conséquent, dans cet article, nous nous intéressons à trouver un modèle pour une formule dans S5 avec le plus petit nombre de mondes possibles. Nous appelons cela le problème de satisfiabilité minimale en logique modale S5 (MinS5-SAT) et nous le définissons comme suit :

Définition 11 (Satisfiabilité minimale en S5)**.** Une formule ϕ est MinS5-satisfaite par une structure $\langle M, w\rangle$ (notée $\langle M, w\rangle \models_{\min} \phi$) si et seulement si $\langle M, w\rangle \models \phi$ et ϕ n'a pas d'autre modèle $\langle M', w'\rangle$ tel que $|M'| < |M|$.

Définition 12 (Problème de satisfiabilité minimale en S5)**.** Le problème de satisfiabilité minimale en S5 (MinS5-SAT) est le suivant : étant donné ϕ une formule de $\mathcal{L}$, trouver une structure $\langle M, w\rangle$ telle que $\langle M, w\rangle \models_{\min} \phi$.

Remarquons que trouver un modèle minimal pour ϕ n'est pas aussi simple que de fusionner les mondes avec les mêmes valuations en un seul monde dans n'importe quel modèle de ϕ. La minimalité ne peut être garantie de cette façon. Revenons à l'exemple 2 pour illustrer cela. Nous avons $\mathrm{dd}(\phi) + 1 = 3$. Si nous supprimons la redondance, nous obtenons un modèle de taille 2. Cependant, ϕ est également satisfaite par la structure suivante contenant un seul monde : $W = \{w_0\}$, $\mathcal{I} = \{\langle a, \{w_0\}\rangle, \langle b, \{w_0\}\rangle\}$, qui est un modèle minimal.

Un moyen très simple de résoudre ce problème consiste à utiliser le solveur S52SAT [5] avec une stratégie de recherche linéaire. Grossièrement, la procédure commence par essayer des structures de taille $b = 1$. Si aucun modèle n'est trouvé, on incrémente la valeur de b. On itère ainsi jusqu'à ce qu'un modèle de ϕ soit trouvé ou que la borne supérieure $\mathrm{dd}(\phi) + 1$ soit atteinte. Cette stratégie est appelée 1toN. Il est bien sûr également possible de le faire dans l'ordre inverse : la procédure commence par $b = \mathrm{dd}(\phi) + 1$ et décrémente la valeur de b (cela s'appelle Nto1). Une autre possibilité consiste à utiliser une recherche dichotomique (appelée Dicho).

Cependant, ces approches sont très naïves. Prenons par exemple 1toN. Lorsque la solution optimale est un modèle de taille m, l'approche effectuera m traductions de S5 vers SAT puis m appels à un solveur SAT. Le problème est qu'une telle stratégie ne profite pas de la réponse UNSAT précédente du solveur pour résoudre la formule à l'étape suivante.

Les solveurs SAT modernes sont capables de profiter des appels précédents lorsqu'ils sont utilisés de façon incrémentale [1]. La façon habituelle d'y parvenir est d'ajouter des variables-sélecteurss (hypothèses) à la formule d'entrée et d'obtenir en sortie, un noyau incohérent en termes de ces sélecteurs. Nous proposons ici un moyen d'ajouter de telles variables-sélecteurss dans la traduction de S5 vers SAT.

4 Une traduction sous hypothèses

La traduction proposée dans [5] est basée sur une idée simple mais efficace : Soit ϕ la formule d'entrée, chaque sous-formule de la forme $\Box\psi$ est traduite en $\bigwedge_{i=1}^{n} \mathrm{tr}(\psi, i, n)$, alors que les sous-formules de la forme $\Diamond\psi$ sont traduites en $\bigvee_{i=1}^{n} \mathrm{tr}(\psi, i, n)$, où n est le nombre de mondes possibles du modèle en cours de construction. Si nous fixons $n = \mathrm{dd}(\phi) + 1$, nous avons la garantie d'avoir une formule équisatisfiable en sortie.

Afin de profiter de la capacité des solveurs SAT modernes à retourner une raison de l'incohérence d'une formule, nous pouvons ajouter des variables-sélecteurss s_i pour activer ou désactiver les mondes w_i. Nous changeons la traduction de $\Box\psi$ en $\bigwedge_{i=1}^{n}(s_i \rightarrow \mathrm{tr}(\psi, i, n))$ et celle de $\Diamond\psi$ en $\bigvee_{i=1}^{n}(s_i \wedge \mathrm{tr}(\psi, i, n))$. En revanche, la CNF résultante est plus grande, la taille du modèle S5 sera désormais déterminée par le nombre de variables-sélecteurss à vrai. La fonction de traduction complète est donnée ci-dessous :

Définition 13 (Traduction sous hypothèses)**.** Soit $\phi \in \mathcal{L}$.

$$\mathrm{tr_s}(\phi, n) = \mathrm{tr_s}'(\phi, 1, n) \quad \mathrm{tr_s}'(p, i, n) = p_i$$
$$\mathrm{tr_s}'(\neg\psi, i, n) = \neg\, \mathrm{tr_s}'(\psi, i, n)$$
$$\mathrm{tr_s}'(\psi \wedge \delta, i, n) = \mathrm{tr_s}'(\psi, i, n) \wedge \mathrm{tr_s}'(\delta, i, n)$$
$$\mathrm{tr_s}'(\psi \vee \delta, i, n) = \mathrm{tr_s}'(\psi, i, n) \vee \mathrm{tr_s}'(\delta, i, n)$$
$$\mathrm{tr_s}'(\Box\psi, i, n) = \bigwedge_{j=1}^{n} (\neg s_j \vee (\mathrm{tr_s}'(\psi, j, n))$$
$$\mathrm{tr_s}'(\Diamond\psi, i, n) = \bigvee_{j=1}^{n} (s_j \wedge (\mathrm{tr_s}'(\psi, j, n))$$

Un modèle S5 doit contenir au moins un monde possible (le monde courant). Sans perte de généralité, nous considérons que le monde courant est le numéro 1, donc s_1 est toujours vraie. Dans le reste de cet article, nous notons $\mathrm{tr_s}(\phi)$ la formule $\mathrm{tr_s}(\phi, \mathrm{dd}(\phi) + 1) \wedge s_1$ et l'ensemble de tous les sélecteurs de $\mathrm{tr_s}(\phi)$ est indiqué par $\mathcal{S}(\phi)$, c'est-à-dire, $\mathcal{S}(\phi) = \{s_i \mid 1 \leq i \leq \mathrm{dd}(\phi) + 1\}$.

Exemple 4 (Exemple de '$\mathrm{tr_s}$')**.** Revenons à l'exemple 1 et réutilisons la formule $\phi = ((\Box\neg a \vee \Diamond b) \wedge \Diamond a \wedge \Box b)$. Sa traduction $\mathrm{tr_s}(\phi, 3)$ est :

$$(\neg s_1 \vee (\neg a_1 \vee (s_1 \wedge b_1) \vee (s_2 \wedge b_2) \vee (s_3 \wedge b_3))) \wedge$$
$$(\neg s_2 \vee (\neg a_2 \vee (s_1 \wedge b_1) \vee (s_2 \wedge b_2) \vee (s_3 \wedge b_3))) \wedge$$
$$(\neg s_3 \vee (\neg a_3 \vee (s_1 \wedge b_1) \vee (s_2 \wedge b_2) \vee (s_3 \wedge b_3))) \wedge$$
$$((s_1 \wedge a_1) \vee (s_2 \wedge a_2) \vee (s_3 \wedge a_3)) \wedge ((\neg s_1 \vee b_1) \wedge$$
$$(\neg s_2 \vee b_2) \wedge (\neg s_3 \vee b_3)) \wedge s_1$$

Intuitivement, chaque formule avec un indice i est une formule qui est vraie dans le monde possible i. Si la variable-sélecteur s_i est fausse, alors le monde i n'est pas présent dans le modèle. Ci-dessous, une formule équivalente à $\mathrm{tr_s}(\phi, 3)$ mais avec s_1 et s_2 activées et s_3 désactivée.

$$(\neg\top \vee (\neg a_1 \vee (\top \wedge b_1) \vee (\top \wedge b_2) \vee (\bot \wedge b_3))) \wedge$$
$$(\neg\top \vee (\neg a_2 \vee (\top \wedge b_1) \vee (\top \wedge b_2) \vee (\bot \wedge b_3))) \wedge$$
$$(\neg\bot \vee (\neg a_3 \vee (\top \wedge b_1) \vee (\top \wedge b_2) \vee (\bot \wedge b_3))) \wedge$$
$$((\top \wedge a_1) \vee (\top \wedge a_2) \vee (\bot \wedge a_3)) \wedge ((\neg\top \vee b_1) \wedge$$
$$(\neg\top \vee b_2) \wedge (\neg\bot \vee b_3)) \wedge \top$$

Cette formule est équivalente à $(\neg a_1 \vee b_1 \vee b_2) \wedge (\neg a_2 \vee b_1 \vee b_2) \wedge (a_1 \vee a_2) \wedge (b_1 \wedge b_2)$, qui est la même que celle présentée dans l'exemple 3. Cela correspond au problème de décider si ϕ est satisfiable dans un modèle avec 2 mondes.

Comme nous pouvons le voir, le problème de résoudre le problème de satisfiabilité minimale en logique modale S5 est maintenant équivalent au problème de satisfaire $\mathrm{tr_s}(\phi, n)$ et de minimiser le nombre de s_i, pour $i > 1$, affectées à vraie (ou, de manière équivalente, maximiser le nombre de s_i affectées à faux). Évidemment, cela peut être vu comme un problème d'optimisation pseudo-booléenne (PBO) (Chapitre PB dans [4]), où la fonction d'optimisation à minimiser est le nombre de variables-sélecteurs assignées à vrai. Ce problème est également souvent résolu de nos jours sous la forme d'une instance du problème MaxSAT partiel [4], qui consiste à satisfaire toutes les *hard clauses* et le nombre maximum de *soft clauses*. Dans notre cas, les *hard clauses* sont celles générées par la fonction de traduction, et les *soft clauses* sont les clauses unitaires $\{\neg s_i \mid s_i \in \mathcal{S}(\phi) \text{ et } i > 1\}$ construites à partir des variables-sélecteurs.

Nous pouvons donc utiliser des solveurs MaxSAT partiel et des solveurs PBO. Cependant, ce n'est pas le seul moyen, comme nous le montrons dans la section suivante. En considérant la structure des modèles S5, l'extraction d'un MSS peut également être utilisée pour décider le problème.

Sans perte de généralité, dans les sections suivantes, nous représentons un ensemble de clauses unitaires comme l'ensemble des variables-sélecteurss le composant (ex. : $\{s_2, s_3, s_4\}$ au lieu de $\{\neg s_2, \neg s_3, \neg s_4\}$).

5 L'optimalité pour la cardinalité est égale à l'optimalité pour l'inclusion

Dans la section 2.3, nous avons défini le problème MSS. Il est également possible de définir une version partielle du problème MSS, où l'objectif est de calculer un MSS de telle sorte qu'un sous-ensemble donné des clauses (les *hard clauses*) doit être satisfait. Ce problème est lié au problème MaxSAT partiel. En fait, une solution à un problème MaxSAT partiel est l'un des plus gros MSS qui satisfasse l'ensemble des *hard clauses*. En général, un MSS partiel n'est pas une solution à un problème MaxSAT partiel mais, dans le cas spécifique de MinS5-SAT, un MSS partiel est également une solution au problème MaxSAT partiel correspondant, ce qui signifie que dans ce contexte spécifique, l'optimalité pour l'inclusion (MSS) est équivalente à l'optimalité pour la cardinalité (MaxSAT).

Proposition 1. Soit Σ une formule en CNF équisatisfiable à $\mathrm{tr_s}(\phi, \mathrm{dd}(\phi)+1)$, et soit χ une formule du type $\bigwedge_{i=2}^{\mathrm{dd}(\phi)+1} \neg s_i$. Un MSS de $(\Sigma \wedge \chi)$, où Σ est l'ensemble des *hard clauses*, est aussi solution du problème MaxSAT partiel.

La preuve de la proposition 1 utilise le lemme suivant.

Lemme 1. Soit $\Sigma = \mathrm{tr_s}(\phi, n)$, et soit χ une formule du type $\bigwedge_{s_i \in \mathcal{S}'} \neg s_i$, où $\mathcal{S}' \subseteq \mathcal{S}(\phi)$. Si $(\Sigma \wedge \chi)$ est satisfiable alors la formule $(\Sigma \wedge \chi')$ aussi, où χ' est obtenu depuis χ en replaçant les occurrences d'une variable-sélecteur $s \in \mathcal{S}'$ par une autre $s' \in \mathcal{S}(\phi) \setminus \mathcal{S}'$.

Esquisse de la preuve du Lemme 1. La preuve est faite par une récurrence sur la longueur de la formule ϕ. Initialisation, $\phi = p$, pour un certain $p \in \mathbb{P}$. Nous avons $\psi = p_1$, $\chi = \neg s_1$ et $\mathcal{S}(\phi) = \{s_1\}$, ce qui signifie que l'affirmation est vraie (car $\mathcal{S}(\phi) \setminus \mathcal{S}' = \emptyset$). Nous avons plusieurs cas d'hérédité de récurrence. Puisque leurs preuves sont toutes semblables, nous n'en montrons qu'une seule ici. Soit $\phi = \Box\phi'$. Nous avons $\psi = \bigwedge_{i=1}^{n}(\neg s_i \vee \mathrm{tr_s}(\phi', i, n))$, $\chi = \bigwedge_{s_i \in \mathcal{S}'} \neg s_i$, et $\mathcal{S}(\phi) = \{s_1, \ldots, s_n\}$. Soit χ', obtenue par χ où s_i est remplacée par $s_j \in \mathcal{S}(\phi) \setminus S'$. Si $(\psi \wedge \chi)$ est satisfaite par un modèle M alors nous construisons un nouveau modèle M'. Ce modèle est égal à M sauf que les valeurs de vérité de toutes les variables propositionnelles avec l'indice i sont les mêmes que celles avec l'indice j. Nous avons immédiatement que si $M \models \chi$ alors $M' \models \chi'$. Nous avons aussi que si $M \models \neg s_i$ alors $M' \models \neg s_j$. Enfin, pour chaque $1 \leq i \leq n$, si $M \models \mathrm{tr_s}(\phi', i, n))$ alors $M' \models \mathrm{tr_s}(\phi', j, n))$, par l'hypothèse de récurrence (puisque la longueur de ϕ' est strictement inférieure à celle de ϕ). Par conséquent, $M' \models \psi \wedge \chi'$. □

Démonstration de la Prop. 1. Raisonnons par l'absurde, supposons qu'il existe un MSS $\delta_1 = (\psi \wedge \chi_1)$, où $\chi_1 = \bigwedge_{s \in S_1} \neg s$, qui n'est pas le plus grand. Donc il existe un autre MSS $\delta_2 = (\psi \wedge \chi_2)$, où $\chi_2 = \bigwedge_{s \in S_2} \neg s$ et tel que $|S_1| < |S_2|$. Maintenant, soit $S_3 = S_2 \setminus \{s\} \cup \{s'\}$, où $s \in S_2$ et $s' \in S_1$. Par le lemme 1, la formule $\delta_3 = (\psi \wedge \chi_3)$, où $\chi_3 = \bigwedge_{s \in S_3} \neg s$ est satisfiable, car elle est équivalente à δ_2 avec l'une des variables-sélecteurs de S_2 dans χ_2 remplacée par une autre variable-sélecteur. Il est facile de voir que l'on peut continuer à remplacer les sélecteurs dans cet ensemble jusqu'à ce que nous ayons l'ensemble S_k, tel que $S_1 \subseteq S_k$. La formule $\delta_k = (\psi \wedge \chi_k)$, où $\chi_k = \bigwedge_{s \in S_k} \neg s$ est satisfiable, en appliquant le lemme 1 $|S_1|$ fois. Par conséquent δ_k est un MSS qui contient δ_1, ce qui contredit l'hy-

pothèse. Cela signifie que chaque MSS de la formule initiale est maximal. Par conséquent, chaque MSS de $(\psi \wedge \chi)$ est aussi solution du problème MaxSAT partiel. □

Comme conséquence directe de la proposition 1, on peut donc toujours trouver un MSS tel que les indices des variables-sélecteurs à l'intérieur sont contigus. Cela signifie que nous pouvons envisager une optimisation qui réduit l'espace de recherche (casse les symétries), en ajoutant l'équation qui suit :

$$(\bigwedge_{i=1}^{n-1} \neg s_i \rightarrow \neg s_{i+1}) \quad (1)$$

En donnant en entrée $\mathrm{tr_s}(\phi, n)$ ainsi que $\mathcal{S}(\phi)$, nous pouvons résoudre le problème MinS5-SAT avec un solveur MaxSAT ou PBO. Si nous ajoutons également l'équation 1 à l'entrée, nous pouvons alors utiliser un extracteur de MSS. Cependant, nous démontrons dans la section suivante que nous pouvons améliorer les performances en considérant une approche dédiée utilisant un solveur SAT incrémental et des noyaux incohérents.

6 Seule la taille du noyau compte

Considérons l'exemple suivant : soit ϕ la formule en entrée et soit $\mathrm{dd}(\phi) + 1 = 10$. Nous traduisons ϕ en utilisant les variables-sélecteurs et nous cherchons un modèle pour cette formule. Supposons que, après un certain temps de calcul, nous concluons que 4 mondes ne peuvent pas être désactivés complètement, c'est-à-dire que si les variables-sélecteurss s_i, s_j, s_k et s_l sont affectées à faux, nous avons une incohérence. Nous pouvons déduire de cette information que nous avons besoin d'au moins 7 mondes pour trouver un modèle S5 à ϕ. Cela vient du fait que les "4 mondes qui ne peuvent pas être complètement désactivés" peuvent être, en fait, n'importe quel groupe de 4 mondes. En effet, dans la suite, nous démontrons que si nous avons un groupe de sélecteurs m formant un noyau incohérent et que la borne est égale à n, alors nous avons besoin d'au moins $(n - m + 1)$ mondes pour espérer trouver un modèle S5 à la formule d'entrée.

Proposition 2. Soit $\phi \in \mathcal{L}$ tel que $\mathrm{dd}(\phi)+1 = n$. Si C est un noyau incohérent de $\mathrm{tr_s}(\phi)$ sous les hypothèses $\mathcal{S}(\phi)$ alors $\mathrm{tr_s}(\phi, n')$ est insatisfiable pour tous $n' \in \{1, \ldots, (n - |C|)\}$.

Lemme 2. Si C est un noyau incohérent de $\mathrm{tr_s}(\phi)$ sous les hypothèses $\mathcal{S}(\phi)$ alors n'importe quel ensemble de littéraux $C' = \{\neg s \mid s \in \mathcal{S}(\phi)\}$ tel que $|C'| = |C|$ est un noyau incohérent de ϕ.

Démonstration. Supposons que C est un noyau incohérent de $\mathrm{tr_s}(\phi)$ sous les hypothèses $\mathcal{S}(\phi)$. Donc $(\phi \wedge \bigwedge_{l \in C} l)$ est incohérent. Maintenant, raisonnons par l'absurde, supposons aussi qu'il existe un ensemble $C' = \{\neg s \mid s \in \mathcal{S}(\phi)\}$ tel que $|C'| = |C|$ et $(\phi \wedge \bigwedge_{s \in C} \neg s)$ est satisfiable. Par le lemme 1, nous obtenons un ensemble D à partir de C' en remplaçant les variables-sélecteurs dans C' par celles de C telles que $(\phi \wedge \bigwedge_{s \in D} \neg s)$ est satisfiable. Comme $D = C$, nous avons une contradiction. Par conséquent, tout ensemble de littéraux C' obtenu ainsi est un noyau incohérent de ϕ. □

Démonstration de la Prop. 2. La formule a n mondes possibles. Le solveur SAT renvoie un noyau incohérent C de taille m. Donc l'une des variables-sélecteurs doit être affectée à vrai. Mais en raison du lemme 2, nous devons mettre au moins une variable-sélecteur à vrai pour tous les noyaux incohérents possibles de taille m. Dit autrement, nous devons avoir $(n - m + 1)$ variables-sélecteurs vraies ensemble ou la formule sera nécessairement incohérente. Cela signifie aussi que $\forall b' \in [1 \ldots (n - m)]$ $\mathrm{tr_s}(\phi, b')$ est incohérent. □

En utilisant cette propriété, il est donc possible de construire un algorithme itératif basé sur un solveur SAT incrémental. Le solveur SAT va être capable de retourner un noyau incohérent, et en l'interprétant dans le cadre de la logique modale S5 comme expliqué dans la Prop. 2, nous pouvons affiner la borne utilisée dans la traduction. Une telle technique est présentée dans l'algorithme 1. La procédure commence par essayer des structures de taille $b = 1$. Si aucun modèle n'est trouvé, on itère le processus, augmentant chaque fois la valeur de b en la remplaçant par $(\mathrm{dd}(\phi) + 1 - |s| + 1)$ (où $|s|$ est la taille du noyau incohérent). On itère jusqu'à ce qu'un modèle de ϕ soit trouvé ou que la borne supérieure $\mathrm{dd}(\phi)+1$ soit atteinte. Notez que $|s|$ diminue strictement à chaque étape, car nous augmentons strictement le nombre de variables-sélecteurs satisfaites. La procédure SAT-solver() qui apparaît dans l'algorithme est le solveur SAT 'glucose' présenté dans [8, 1]. La procédure *getS5Model*() est une procédure qui génère le modèle S5 à partir du modèle propositionnel donné en argument. En raison du manque d'espace, nous présentons seulement l'approche $1toN_c$ mais l'approche $Dicho_c$ est similaire.

7 Résultats Expérimentaux

Nous avons comparé plusieurs approches différentes au problème MinS5-SAT : S52SAT [5] avec cinq stratégies différentes : $1\mathrm{toN}_c$, 1toN, Nto1, Dicho_c, Dicho. CNF plus un solveur MaxSAT : maxHS-b [6], mscg2015b [7], et MSUnCore [15]. Traduction vers Pseudo-Booléen (PB) plus un solveur *PBO* : NaPS [30], SAT4J-PB [22], SCIP [23]. CNF plus le cassage de symétrie plus l'extracteur de MCS : LBX [26]. Pour voir l'impact de notre minimisation, nous utilisons le solveur de logique modale S5

Algorithme 1 : S52SAT 1toN$_c$

Données : $\phi \in \mathcal{L}$
Résultat : $\langle M, w\rangle$ t.q $\langle M, w\rangle \models_{min} \phi$, sinon UNSAT

```
1  b ← 1 ;
2  ⟨r, s⟩ ← SAT-solver(tr_s(φ, b)) ;
3  n ← dd(φ) + 1 ;
4  tant que (r ≠ SAT ∧ (b ≤ n)) faire
5  |  b ← (n − |s| + 1) ;
6  |  ⟨r, s⟩ ← SAT-solver(tr_s(φ, b)) ;
7  si (r ≠ SAT) alors retourner UNSAT ;
8  sinon
9  |  ⟨M, w⟩ ← getS5Model(s) ;
10 |  retourner ⟨M, w⟩;
```

S52SAT avec glucose (4.0) comme solveur SAT embarqué [1] (avec sa mise en cache activée). Nous avons sélectionné les solveurs MaxSAT qui ont montré de bonnes performances dans la compétition MaxSAT 2016 [1]. Malgré nos recherches, nous n'avons pas pu trouver de benchmarks pour la logique modale S5. Donc, nous avons choisi d'utiliser les benchmarks suivants pour les logiques modales K, KT et S4 : TANCS-2000 QBF (MQBF) [25] ; LWB K, KT et S4 [10], avec 56 formules choisies parmi chacune des 18 classes paramétrées, générées à partir du script donné par les auteurs de [27] ; et les benchmarks $3CNF_{KSP}$ générés aléatoirement [29] de profondeur modale égale à 1 et 2. Nous avons gardé seulement les benchmarks satisfaits dans leur logique d'origine pour voir l'impact d'une utilisation potentielle d'une résolution S5 en tant que préprocesseur pour d'autres logiques modales.

Nous avons également proposé de nouveaux benchmarks basés sur la planification avec des incertitudes dans l'état initial, afin de vérifier la performance des différentes approches sur des benchmarks structurés. Dans de tels problèmes de planification, certains fluents f peuvent être initialement vrais, faux, ou indéfinis. Dans ce dernier cas, deux situations initiales différentes sont possibles. Par conséquent, au lieu d'un seul état initial s_0, nous pouvons avoir plusieurs états initiaux différents, ce qui est cohérent avec les connaissances disponibles sur le système (voir [11] pour plus de détails et d'applications). Par construction, tous les benchmarks ont un modèle S5 à minimiser. Nous augmentons la valeur de l'horizon délimité jusqu'à atteindre la plus petite valeur pour laquelle il existe un plan comme expliqué dans le chapitre de Planning dans [4]. Nous avons inclus les benchmarks de planification conformante traditionnels, à savoir : Bomb-in-the-toilet, Anneau, Cube, Omelette et Safe (voir [16] pour plus de détails) modélisés ici comme planification avec des incertitudes sur l'état initial. Nous avons également effectué des évaluations sur des benchmarks classiques : Blocksworld, Logistics, et Grid, dans lesquels les auteurs de [16] ont introduit une incertitude sur l'état initial. Tous les benchmarks sont disponibles en téléchargement [2]. Pour sélectionner les benchmarks minimisables, nous utilisons une limite de temps de 1500 secondes. Nous avons réussi à résoudre 28 benchmarks sur les 119 disponibles. Nous avons essayé d'autres solveurs : Spartacus [14] a résolu 15 instances et SPASS [9] en a résolu 5, avec les deux étant un sous-ensemble des 28 résolus par S52SAT. Notre générateur a des temps d'exécution négligeables et est disponible au téléchargement [3]. Chacun de ces benchmarks a un plan de taille N (où N peut être différent pour chaque benchmark) qui a été vérifié. Nous avons ensuite généré des benchmarks de logique modale S5 à partir de ces instances en fixant l'horizon à $N, N+1, \ldots, N+9$ ayant ainsi 280 benchmarks, tous S5-satisfiables, pour tester nos techniques de minimisation.

Les benchmarks et les différents solveurs (notamment S52SAT, qui est celui qui traduit les formules en logique propositionnelle) sont disponibles [4]. Les expérimentations se sont déroulées sur un cluster de 4 coeurs Xeon, 3.3 GHz, exécutant CentOS 6.4. La limite de mémoire est fixée à 32 Go et la limite d'exécution est fixée à 900 secondes par solveur par benchmark. Dans les Tables suivants, nous fournissons le nombre de benchmarks pour lesquels un modèle S5 minimal est trouvé. En gras, le meilleur résultat pour chaque ligne / colonne. Le VBS (Virtual Best Solver) représente l'union des benchmarks résolus par toutes les approches.

Logic WorkBench (LWB) Benchmarks Les résultats sur LWB sont affichés dans la Table 1c. La différence dans les résultats entre les approches utilisant S52SAT et les solveurs MaxSAT provient du fait que les solveurs MaxSAT ne peuvent pas prendre en compte les propriétés inhérentes à la logique modale S5. Ils possèdent des contraintes de cardinalité utilisées pour compter le nombre de clauses satisfaites/falsifiées pour renvoyer le plus petit modèle. Les approches 1toN et Dicho ont plus de succès que Nto1 du fait que les benchmarks peuvent généralement être résolus avec peu de mondes possibles, donc en commençant par la borne supérieure théorique, l'approche est moins efficace. En comparant les résultats de 1toN et 1toN$_c$ en nombre de benchmarks résolus, on pourrait penser que les variables-sélecteurs ne font pas beaucoup de différence. Mais le temps d'exécution fournit une image différente, comme dans le diagramme de dispersion représenté par la Figure 1g. L'abscisse correspond au temps utilisé par 1toN$_c$ alors que l'ordonnée correspond au temps utilisé par 1toN pour résoudre ces problèmes. Comme prévu, 1toN$_c$ effectue moins d'itérations et appelle donc le solveur SAT moins

1. http ://www.maxsat.udl.cat/16/

2. https://goo.gl/7i6thX
3. http://www.cril.fr/~montmirail/planning-to-s5/
4. http://www.cril.fr/~montmirail/s52SAT

souvent. Nous remarquons que le solveur a pris moins de 10 secondes pour la majorité des cas. Il s'avère qu'il est intéressant de considérer cette approche comme un prétraitement pour un solveur de satisfiabilité minimale en logique modale plus générique (par exemple, pour les logiques K, KT et S4).

3CNF$_{KSP}$ benchmarks Les formules $3CNF_{KSP}$ générées aléatoirement [29] de profondeur 1 et 2 se composent de 1000 formules, où 457 sont satisfiables en logique modale K. Les résultats sont affichés dans la Table 1b. Comme pour LWB, 1toN et Dicho sont meilleures que Nto1 car les modèles minimaux trouvés sont relativement petits. Il est intéressant de noter que la profondeur modale des formules influence le résultat. Ceci est surprenant du fait que, dans S5, toutes les formules peuvent être réduites à une profondeur modale de 1. En fait, de nombreuses instances avec une profondeur modale égale à 2, qui sont SAT dans la logique modale K, sont UNSAT dans la logique modale S5.

Randomly Modalized QBF benchmarks À l'origine, cet ensemble de référence contient 1016 formules, parmi lesquelles 617 sont SAT tandis que 399 sont UNSAT en K. Les résultats sont affichés dans la Table 1a. Les approches Dicho, Dicho$_c$, 1toN et 1toN$_c$ sont meilleures que les autres. De plus, il est intéressant de voir que l'ensemble **qbf** est en fait S5-satisfiable, même s'il est normalement utilisé pour évaluer les solveurs de la logique modale. Il est intéressant de noter que les performances d'un solveur MaxSAT ou PBO sont globalement plus mauvaises que l'extraction d'un MSS. Cependant, si nous ajoutons le cassage de symétrie de l'équation 1 alors les performances deviennent équivalentes.

Planning avec état initial incertain Comme dans les benchmarks aléatoires et *crafted* précédent, nous pouvons voir dans la Table 1d que l'utilisation des variables-sélecteurs nous permet de résoudre plus de benchmarks. Mais, étonnamment, ici la meilleure approche est d'utiliser une recherche dichotomique au lieu d'une recherche linéaire de 1 à N. Ceci est principalement dû à la taille du plus petit modèle, qui est rarement petit, comme c'était le cas dans LWB par exemple. De plus, chaque appel au solveur SAT prend plus de temps car les instances sont plus difficiles à résoudre en pratique.

Coût de minimisation Nous pouvons voir sur la Figure 1e que cela ne nécessite qu'un calcul additionnel acceptable d'obtenir le plus petit modèle possible au lieu du premier retourné par le solveur (de moins de 10s à moins de 40s). D'autre part, comme on peut le voir sur la Figure 1f la minimisation peut réduire drastiquement la taille du modèle renvoyé. De plus, nous pouvons également voir la différence structurelle entre les instances générées aléatoirement et les instances d'applications réelles. Il y a aussi un gain par rapport aux autres solveurs capables de sortir un modèle (Figure 1h), tel que Spartacus [14]. Notez que nous ne pouvons comparer que des modèles Spartacus sur des problèmes de planification car Spartacus est dédié aux logiques modales K, KT et S4, pas S5. Cependant, sur ces benchmarks spécifiques, puisque la profondeur modale est 1 et tous les modèles K sur ces benchmarks sont des modèles S5, nous pouvons comparer leur taille. Notez également que Spartacus affiche un tableau ouvert et saturé (qui indique l'existence d'un modèle) et non un modèle complet qui devrait être encore plus grand (voir [20] pour plus détails).

8 Conclusion

Nous avons défini dans cet article un nouveau problème d'optimisation que nous appelons satisfiabilité minimale en S5 (MinS5-SAT). Nous avons démontré que ce problème peut être réduit au problème de l'extraction d'un ensemble de clauses de satisfaction maximale (MSS) et peut donc être résolu avec un extracteur de MSS ou l'un des solveurs PBO ou MaxSAT les plus récents. Nous avons également montré que, grâce à une propriété inhérente à la logique modale S5, ce problème peut également être résolu en utilisant des noyaux incohérents dans une procédure SAT incrémentale. Cette dernière approche est celle qui a obtenu les meilleures performances dans nos expériences. La technique utilisée a obtenu des gains énormes dans la taille des modèles de sortie, par rapport aux autres approches qui n'essaient pas la minimisation. De plus, le surcoût imposé par la minimisation est acceptable. Par conséquent, nous croyons que trouver des modèles minimaux pour les formules logiques modales est une tâche intéressante.

Une piste possible pourrait d'étendre les résultats sur S5 aux autres logiques modales NP-complete, une autre piste pourrait être d'essayer de résoudre le problème de satisfiabilité minimale en logique modale (MinML-SAT), plus général, pour lequel le problème standard est typiquement PSPACE-difficile. On sait que la satisfiabilité en S5 implique la satisfiabilité en K. Si l'on trouve un modèle minimal S5 de taille n pour une formule ϕ alors le modèle minimal K pour ϕ a au plus n mondes.

Remerciements

Nous remercions les relecteurs anonymes pour leurs commentaires avisés. Une partie de ces travaux ont étés soutenues par le Ministère de l'Enseignement Supérieur et de la Recherche et le Conseil Régional Nord-Pas de Calais par le biais du "Contrat de Plan État Région (CPER) DATA" ainsi que par une subvention EC FEDER. Ces travaux ont déjà étés publiés à IJCAR'18 [21].

Benchs	1toN	Nto1	Dicho	1toN$_c$	Dicho$_c$	maxHS	MSCG	MSUnCore	NaPS	SAT4J	SCIP	LBX	VBS
qbf (56)	**56**	55	**56**	**56**	**56**	**56**	**56**	**56**	55	48	**56**	**56**	56
qbfS (171)	**171**	0	**171**	**171**	**171**	0	156	0	144	140	155	167	171
Total (227)	**227**	55	**227**	**227**	**227**	56	212	56	199	188	211	223	227

(a) #instances résolues dans $MQBF$

Benchs	1toN	Nto1	Dicho	1toN$_c$	Dicho$_c$	maxHS	MSCG	MSUnCore	NaPS	SAT4J	SCIP	LBX	VBS
md=1 (62)	55	0	26	**62**	40	40	30	38	42	35	42	47	62
md=2 (27)	17	0	9	**27**	17	12	12	12	17	12	20	17	27
Total (89)	72	0	35	**89**	57	52	42	50	59	47	62	64	89

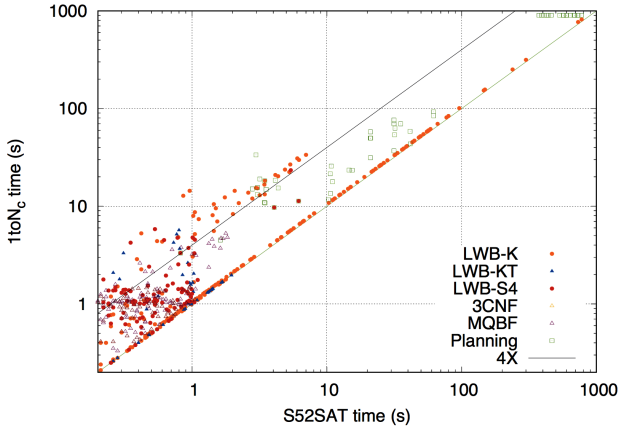
(b) #instances résolues dans $3CNF_{KSP}$

Method # Benchs	**K** (185)	**KT** (279)	**S4** (160)	**Total** (624)
1toN	**185**	**279**	**160**	**624**
Nto1	17	34	2	53
Dicho	119	175	78	372
1toN$_c$	**185**	**279**	**160**	**624**
Dicho$_c$	135	201	100	436
maxHS	17	25	72	114
MSCG	74	65	103	242
MSUnCore	19	30	80	129
NaPS	126	64	71	261
SAT4J	18	27	58	103
SCIP	104	158	112	374
LBX	118	173	92	383
VBS	185	279	160	624

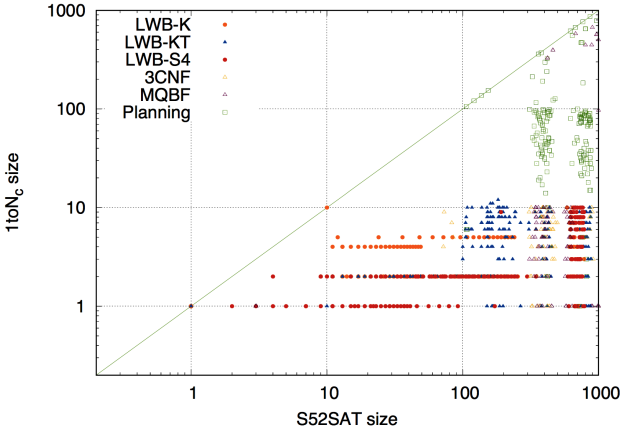
(c) #Instances résolues dans la catégorie LWB

Method # Benchs	**block** (20)	**bomb** (30)	**cube** (100)	**omelet** (30)	**ring** (40)	**safe** (60)	**Total** 280
1toN	10	25	**100**	0	**40**	0	175
Nto1	5	10	20	0	20	0	55
Dicho	**20**	20	70	0	**40**	0	150
1toN$_c$	**20**	25	**100**	10	**40**	30	225
Dicho$_c$	**20**	**30**	**100**	**18**	**40**	**34**	**242**
maxHS	17	22	82	0	**40**	5	166
MSCG	**20**	25	72	0	**40**	4	161
MSUnCore	10	22	68	0	38	0	138
NaPS	**20**	**30**	74	2	**40**	8	174
SAT4J	**20**	20	58	0	33	0	131
SCIP	**20**	**30**	**100**	5	**40**	10	200
LBX	12	26	79	0	**40**	0	157
VBS	20	30	100	18	40	34	242

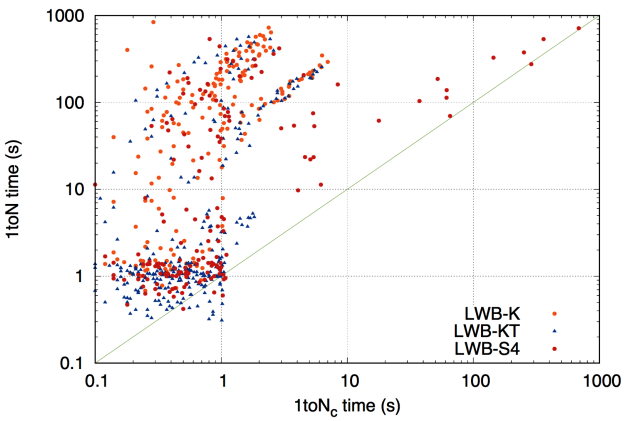
(d) #Instances résolues dans la catégorie Planning



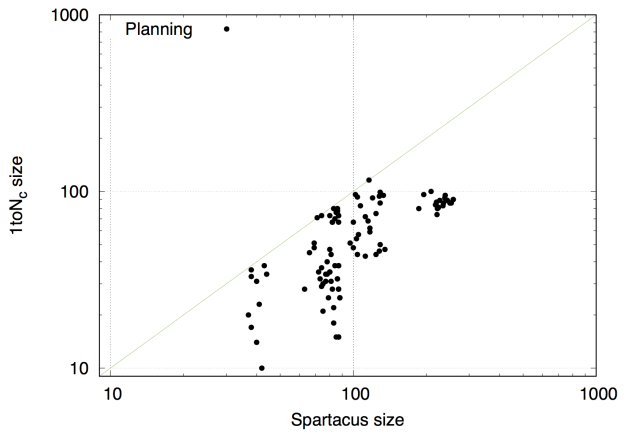
(e) S52SAT vs S52SAT-1toN$_c$ (time)



(f) S52SAT vs S52SAT-1toN$_c$ (size)



(g) Diagramme de dispersion de 1toN vs 1toN$_c$



(h) Spartacus vs S52SAT-1toN$_c$ (size)

FIGURE 1: Résultats des expérimentations et analyse du coût de minimisation

Références

[1] G. Audemard, J-M. Lagniez, and L. Simon. Improving Glucose for Incremental SAT Solving with Assumptions : Application to MUS Extraction. In *Proc. of SAT'13*, 2013.

[2] A. Belov and J. Marques-Silva. Accelerating MUS Extraction With Recursive Model Rotation. In *Proc. of FMCAD'11*, 2011.

[3] M. Bienvenu, H. Fargier, and P. Marquis. Knowledge Compilation in the Modal Logic S5. In *Proc. of AAAI'10*, 2010.

[4] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*. IOS Edition, 2009.

[5] T. Caridroit, J-M. Lagniez, D. Le Berre, T. de Lima, and V. Montmirail. A SAT-Based Approach for Solving the Modal Logic S5-Satisfiability Problem. In *Proc. of AAAI'17*, 2017.

[6] J. Davies and F. Bacchus. Exploiting the Power of MIP Solvers in MaxSAT. In *Proc. of SAT'13*, 2013.

[7] A. José dos Reis Morgado, A. S. Ignatiev, and J. Marques Silva. MSCG : Robust Core-Guided MaxSAT Solving. *J. SAT*, 9, 2014.

[8] N. Eén and N. Sörensson. An Extensible SAT-solver. In *Proc. of SAT'03*, 2003.

[9] C. Weidenbach et al. SPASS Version 3.5. In *Proc. of CADE'09*, 2009.

[10] P. Balsiger et al. A Benchmark Method for the Propositional Modal Logics K, KT, S4. *Journal of Automated Reasoning*, 24(3), 2000.

[11] T. Eiter et al. Planning under Incomplete Knowledge. In *Proc. of CL'00*, 2000.

[12] M. Fairtlough and M. Mendler. An Intuitionistic Modal Logic with Applications to the Formal Verification of Hardware. In *Proc. of CSL'94*, 1994.

[13] M. Fitting. Modality and Databases. In *Proc. of TABLEAUX'00*, 2000.

[14] D. Götzmann, M. Kaminski, and G. Smolka. Spartacus : A Tableau Prover for Hybrid Logic. *ENTCS*, 262, 2010.

[15] F. Heras, A. Morgado, and J. Marques-Silva. Core-Guided Binary Search Algorithms for Maximum Satisfiability. In *Proc. of AAAI'11*, 2011.

[16] J. Hoffmann and R. I. Brafman. Conformant Planning via Heuristic Forward Search : A New Approach. *A.I*, 170(6-7), 2006.

[17] M. Iser, C. Sinz, and M. Taghdiri. Minimizing Models for Tseitin-Encoded SAT Instances. In *Proc. of SAT'13*, 2013.

[18] S. A. Kripke. Semantical Analysis of Modal Logic I. Normal Propositional Calculi. *Zeitschr. math. Logik und Otundlagend. Math*, 9(56), 1963.

[19] R. E. Ladner. The Computational Complexity of Provability in Systems of Modal Propositional Logic. *SIAM Journal of Computation*, 6(3), 1977.

[20] J-M. Lagniez, D. Le Berre, T. de Lima, and V. Montmirail. On Checking Kripke Models for Modal Logic K. In *Proc. of PAAR@IJCAR'16*, 2016.

[21] J-M. Lagniez, D. Le Berre, T. de Lima, and V. Montmirail. An Assumption-Based Approach for Solving The Minimal S5-Satisfiability Problem. In *Proc. of IJCAR'18*. Springer, 2018.

[22] D. Le Berre and A. Parrain. The SAT4J library, release 2.2. *Journal of SAT*, 7(2-3), 2010.

[23] S. J. Maher and T. Fischer et. al. The SCIP Optimization Suite 4.0. Technical Report 17-12, ZIB, Takustr.7, 14195 Berlin, 2017.

[24] J. Marques-Silva and M. Janota. On the Query Complexity of Selecting Few Minimal Sets. *ECCC*, 21, 2014.

[25] F. Massacci and F. M. Donini. Design and Results of TANCS-2000 Non-classical (Modal) Systems Comparison. In *Proc. of TABLEAUX'00*, 2000.

[26] C. Mencía, A. Previti, and J. Marques-Silva. Literal-Based MCS Extraction. In *Proc. of IJCAI'15*, 2015.

[27] C. Nalon, U. Hustadt, and C. Dixon. $K_S P$: A Resolution-Based Prover for Multimodal K. In *Proc. of IJCAR'16*, 2016.

[28] A. Niveau and B. Zanuttini. Efficient Representations for the Modal Logic S5. In *Proc. of IJCAI'16*, 2016.

[29] P. F. Patel-Schneider and R. Sebastiani. A New General Method to Generate Random Modal Formulae for Testing Decision Procedures. *J.A.I.R*, 18, 2003.

[30] M. Sakai and H. Nabeshima. Construction of an ROBDD for a PB-Constraint in Band Form and Related Techniques for PB-Solvers. *IEICE Transactions*, 98-D(6), 2015.

[31] L. Simon, D. Le Berre, and E. A. Hirsch. The SAT2002 Competition. *Annals of Mathematics and Artificial Intelligence*, 43(1), 2005.

[32] T. Soh and K. Inoue. Identifying Necessary Reactions in Metabolic Pathways by Minimal Model Generation. In *Proc. of ECAI'10*, volume 215 of *FAIA*, 2010.

[33] A. Tacchella. *SAT System Description. In *Proc. of DL'99*, volume 22. CEUR-WS.org, 1999.

[34] T. Murphy VII, K. Crary, and R. Harper. Distributed Control Flow with Classical Modal Logic. In *Proc. of CSL'05*, 2005.

Transformation de Modèles et Contraintes pour l'Ingénierie Dirigée par les Modèles

Théo Le Calvar[1,2] **Fabien Chhel**[2] **Frédéric Jouault**[2] **Frédéric Saubion**[1]

[1] Université d'Angers - LERIA - Angers
[2] Groupe ESEO - Angers
prénom.nom@{univ-angers.fr, eseo.fr}

Abstract

Dans le contexte de l'ingénierie dirigée par les modèles, la transformation de modèles est une technique puissante et générique qui produit des modèles cibles à partir de modèles sources. À partir d'un modèle source donné, les approches classiques de transformation sélectionnent et renvoient généralement un modèle unique parmi tous les modèles conformes au métamodèle cible. À cause de cela, ces approches ne sont pas adaptées aux situations où le choix du modèle cible peut bénéficier de l'optimisation ou de l'apport de l'utilisateur.

Dans cet article, nous proposons une approche qui combine la transformation de modèles avec la programmation par contraintes et l'optimisation pour transformer chaque modèle source en un ensemble de modèles cibles. Les utilisateurs peuvent ensuite les explorer à l'aide d'un solveur. Notre approche est basée sur le concept de variables ponts, qui est un nouveau mécanisme permettant de faire le pont entre un moteur de transformation et un solveur de contraintes. Nous présentons également un langage dédié à l'écriture de telles transformations couplées à des contraintes. Cette approche a été validée en l'appliquant à la visualisation de modèles graphiques.

1 Introduction

L'Ingéniérie Dirigée par les Modèles (IDM) [4] vise à mieux utiliser la notion de modèle pour améliorer le développement logiciel. Dans ce contexte, la transformation de modèles constitue un outil important pour la gestion et l'utilisation de ces modèles. Or lors de l'écriture d'une transformation, le développeur doit faire de nombreux choix.

La raison est que les approches classiques de transformation de modèle ne génèrent qu'un seul modèle cible à partir d'un modèle source et il y a des situations où tous les choix ne peuvent être intégrés. Par exemple, dans les systèmes d'aide à la décision, tels que la configuration de lignes de produits, l'utilisateur doit pouvoir explorer l'ensemble des solutions disponibles. Le problème de plannification et d'assignation de ressources (par exemple des enseignants) est un autre exemple. Résoudre entièrement ces problèmes est difficile, cependant des outils proposant à l'utilisateur d'explorer l'espace des solutions pourraient aider.

Nous pouvons aussi mentionner deux exemples plus proche du monde de l'IDM. Le premier, lors de la génération de code, de nombreuses implémentations existent pour un même modèle. Dans ces situations, de nombreux choix sont fixés et l'utilisateur ne peut pas explorer les autres solutions valides. La seconde est la visualisation de modèle, où on crée des diagrammes à partir de modèles. Une transformation de modèle peut être utilisée pour transformer un modèle source en une vue. Cependant, calculer un bon positionnement est difficile. Bien qu'il existe des algorithmes pour générer des solutions initiales correctes, l'utilisateur doit souvent améliorer le diagramme manuellement. Par conséquent, une transformation générant une vue ne devrait pas calculer les positions des éléments graphiques. Au lieu de cela, la transformation devrait spécifier quels diagrammes sont valides et laisser l'utilisateur explorer l'espace des positions valides.

L'approche présentée dans ce papier consiste à combiner la résolution de contraintes avec la transformation de modèles d'une façon qui peut être appliquée à ce genre de problème. Cette collaboration est possible grâce à un nouveau mécanisme appelé *variable pont*. Nous utilisons la transformation de modèle pour générer un modèle cible à partir d'un modèle source, nous générons aussi des contraintes qui s'appliquent au modèle cible. Ceci est résumé par la Figure 1. Au

lieu de totalement définir le modèle cible, nous définissons un modèle cible ainsi que des contraintes s'appliquant aux propriétés du modèle. Les utilisateurs peuvent ensuite manipuler un modèle qui satisfait ces contraintes. L'utilisateur peut casser le modèle en le modifiant, ce qui implique que le modèle ne valide plus les contraintes imposées. Un solveur de contraintes est alors utilisé pour réparer le modèle de façon à ce que le résultat soit le plus proche possible de ce que l'utilisateur voulait tout en validant les contraintes.

Nous validons notre approche en l'adaptant au cas d'étude de la visualisation de diagramme décrit en dessous. Nous pouvons définir des visualisateurs en utilisant une transformation de modèle accompagnée de contraintes. Les utilisateurs ne voient qu'une solution, mais ils peuvent la modifier. Un solveur de contraintes la répare au besoin, avec si possible une fonction à optimiser.

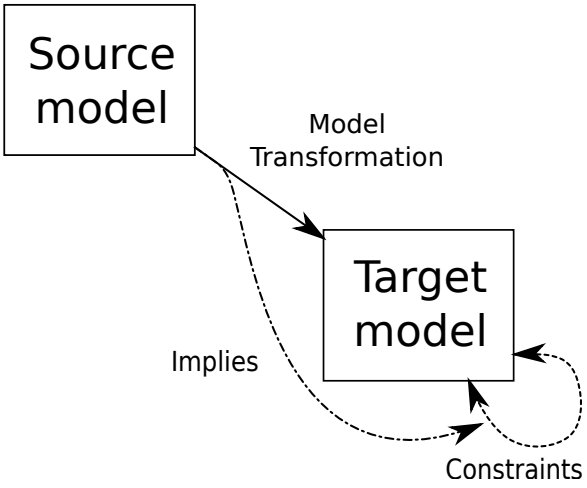


FIGURE 1 – Diagramme de l'approche

Le papier est organisé de la façon suivante. La Section 2 présente le cas d'étude de la visualisation de modèle ainsi qu'un exemple. L'approche est présentée dans la Section 3, ainsi que la notions d'exploration d'ensemble de modèles cibles. Dans la Section 4 nous décrivons comment fonctionne l'association du modèle cible au solveur de contraintes. La Section 5 adapte l'approche au cas détude. L'implémentation est décrite dans la Section 6. La Section 7 liste quelques travaux en rapport avec notre approche et la Section 8 conclut.

2 Cas d'étude et exemple

Dans l'introduction, nous avons listé plusieurs applications possible à notre approche. Cette section présente l'application que nous avons choisi : la visualisation de modèles, ainsi que l'exemple du diagramme de classe.

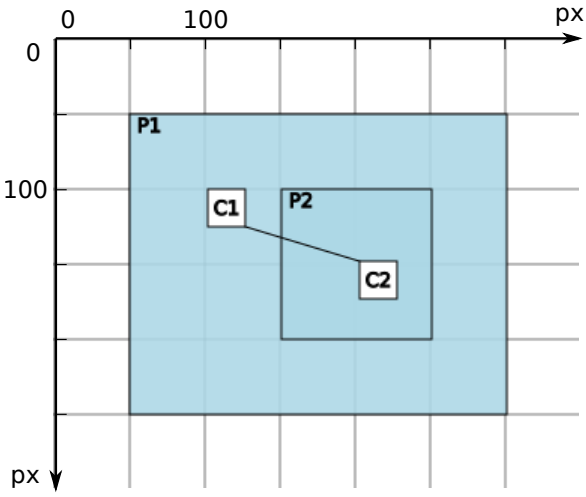


FIGURE 2 – Capture d'écran du résultat

2.1 Visualisation de modèle

Un outil de visualisation de modèle créer une représentation visuelle d'un modèle source. Comme exemple, nous utilisons une représentation (très) simplifiée du diagramme de classe, voir Figure 2, avec des rectangles et textes pour les classes et packages et des lignes pour les associations. Il y a de nombreuses façon d'implementer ce type d'outil. Dans ce papier, la visualisation de modèle n'est qu'un cas d'étude pour illustrer notre approche. Notre approche n'est en aucun cas la plus adaptée à ce genre de situation.

En gardant à l'esprit l'idée générale présentée dans la Figure 1, nous pouvons la spécialiser pour notre cas d'étude. Le modèle source est le modèle que l'on souhaite visualiser, tandis que le modèle cible correspond à la vue. Le modèle cible définit donc un ensemble de figures géométriques telles que le rectangle, la ligne et le texte. Ce modèle correspond à ce qui est appelé un *graphe de scène*, c'est une structure de données communément utilisée pour représenter des éléments graphiques. Dans notre cas, nous utilisons JavaFX pour afficher notre graphe de scène [1]. C'est donc notre modèle cible, il suffit alors d'instancier des figures et les ajouter au graphe de scène pour qu'elles soient affichées.

2.2 Définition de la transformation

Les Figures 3 et 4 définissent les métamodèles utilisés pour l'exemple de la Figure 2. Une version simplifiée du métamodèle du diagramme de classe est donné dans la Figure 3. Un `Package` peut contenir d'autres `Packages` et `Classes` via la classe abstraite `Packagea-`

1. `https://docs.oracle.com/javafx/2/scenegraph/jfxpub-scenegraph.htm`

bleElement. Une Association relie deux Classes : les propriétés first et second spécifient ces deux Classes. Tous les éléments ont un nom. La Figure 4 donne un métamodèle simplifié inspiré par JavaFX. Les Rectangles et Textes ont des coordonées (x et y), ainsi qu'une taille (width et height). Chaque Text possède un text, qui contient la String à afficher à l'écran. Enfin, les Lines possèdent un point de départ (startX, startY) et un point de fin (endX, endY).

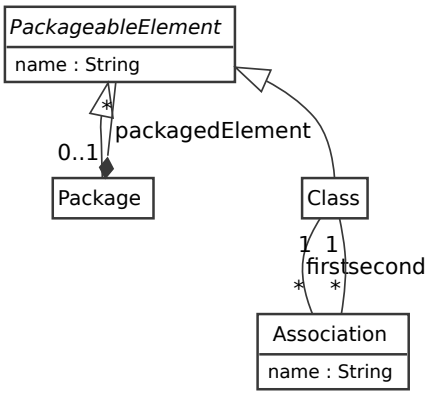


FIGURE 3 – Métamodèle source

Un exemple de source est donné dans la Figure 5. Il contient deux Packages nommés P1 et P2. P1 contient P2 ainsi qu'une Class nommée C1. P2 contient une autre Class C2. Une Association A relie C1 et C2. L'application de la transformation permet de créer le modèle montré dans la Figure 6. Chaque Package et Class est transformé en un Rectangle nommé en ajoutant .r au nom de l'élément (ex. P1.r pour P1) et un Text nommé en ajoutant .t (ex. P1.t pour P2). Une Line est créée pour l'Association.

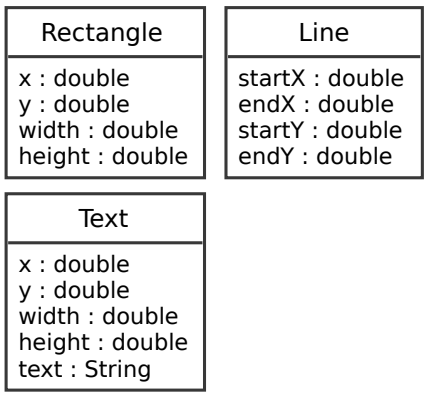


FIGURE 4 – Métamodèle cible

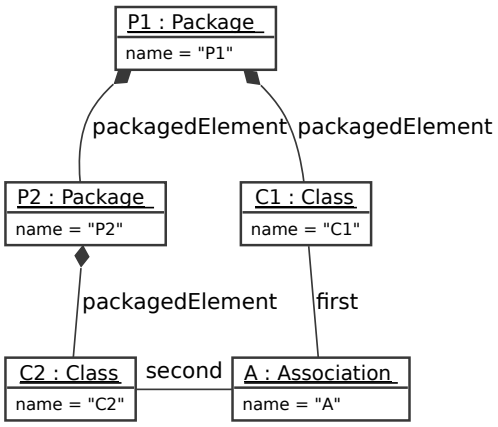


FIGURE 5 – Diagramme d'objet de la source

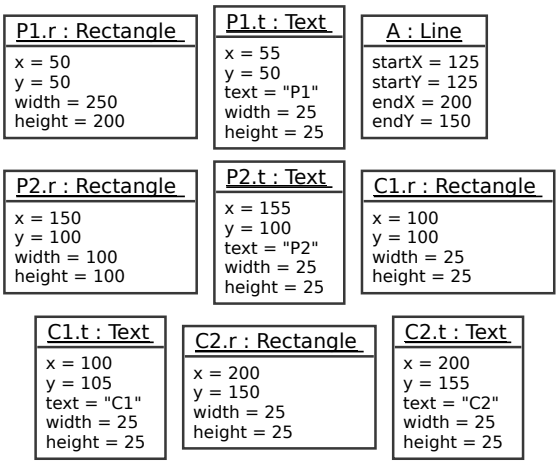


FIGURE 6 – Diagramme d'objet de la cible

2.3 Cohérence géométrique

Dans la Figure 6 nous voyons des coordonnées qui ne proviennent pas du modèle source. Elles correspondent à la position des éléments de la Figure 2, cependant elles ne sont pas calculées à partir d'informations contenues dans le modèle source (Figure 5). C'est ici que notre approche entre en jeu. Au lieu d'utiliser la transformation pour calculer ces coordonnées, nous l'utilisons pour définir des relations entre les propriétés du modèle cible. Notre approche permet à la transformation de le faire en spécifiant des contraintes entre ces propriétés, et ce tout en créer le modèle cible. C'est à cela que correspond la double flèche dans la Figure 1. Elle relie le modèle cible aux contraintes. Deux exemples de contraintes :

Constrainte 1 : Placement du texte Le point en haut à gauche de chaque Text doit coïncider avec le point en haut à gauche du Rectangle correspondant.

Constrainte 2 : Éléments contenus Chaque `Rectangle` correspondant à un *packaged element* doit rester à l'intérieur du `Rectangle` correspondant à son `Package`.

Les contraintes ont souvent été utilisées pour modéliser des problèmes de visualisation ou de géométrie. On peut penser aux travaux de reconstructions de scène de [27], de contrôle de caméra [2] ou encore à [8] et son raisonnement géométrique. De plus, le monde de blocs est souvent utilisé pour illustrer les contraintes sur domaine finis.

Ces contraintes géométriques assurent que la vue reste correcte quelle que soit la situation. L'utilisateur peut alors modifier la vue en déplaçant les formes. Les contraintes sont utilisées pour réparer le diagramme si l'utilisateur effectue un mouvement interdit. Par exemple, l'utilisateur peut déplacer le rectangle d'une `Class` contenu dans le rectangle d'un `Package` en dehors des limites de ce dernier. Il faut alors choisir entre déplacer le rectangle correspondant au `Package`, l'agrandir ou simplement ignorer le déplacement. Ces trois actions répareraient le modèle.

Il y a deux autres fonctionnalités que notre approche permet d'exprimer. La première est que le développeur peut définir finement quelle stratégie de réparation utiliser (ex. élargir un rectangle plutôt que de le déplacer). La seconde est que l'utilisation d'une transformation incrémentale permet de visualiser des modèles qui évoluent.

3 Ensemble de modèles cibles explorable

Dans cette section, nous présentons le concept d'exploration d'un ensemble de modèles cibles.

3.1 Définition en intention d'un ensemble de modèles cibles

Notre approche est basée sur la possibilité pour l'utilisateur d'explorer un ensemble de modèles cibles. Les approches classiques calculent un modèle cible unique t à partir d'un unique modèle source s. Cependant, dans certain cas, d'autres modèles cibles valides t' existent. Par exemple, considérons cette contrainte appliquée à un rectangle : `r.width >= 100`. Il y a ici une nombre infini de solutions valides pour `width`. Nous définissons l'*ensemble de modèles cibles* comme l'ensemble de tous les modèles cibles valides (par rapport à la transformation) correspondant à un modèle source donné.

Pour pouvoir mieux expliquer comment cet ensemble peut être exploré nous rappelons la définition d'*espace de modèles* présenté dans [5]. Un *espace de modèles* est un graphe dirigé $M = (M_\bullet, M_\Delta)$ avec $M_\bullet$ un ensemble de noeuds appelés modèles et M_Δ un ensemble d'arcs appelés deltas ou mise à jour. Si nous définissons $M_\bullet$ comme le set de tous les modèles conformes au métamodèle cible, nous remarquons qu'un ensemble de modèles cibles est un sous-set de $M_\bullet$. Nous définissons l'*exploration d'un espace de modèles* comme l'application d'un ou plusieurs deltas sur un modèle de $M_\bullet$. Par exemple, dans la visualisation de diagramme, une exploration correspond à suivre un delta qui déplace une forme.

L'ensemble de modèles cibles peut être arbitrairement grand. Il faut donc un mécanisme pour le définir en intention. Nous proposons d'utiliser la programmation par contrainte et d'ajouter des contraintes aux modèles générés par les approches classiques de transformation. La programmation par contrainte permet de décrire de façon déclarative le problème à résoudre et de laisser un solveur trouver une solution. Plus formellement, un CSP (Constraint Satisfaction Problem) est un triplet $P = (X, D, C)$ avec X un ensemble de variables de décision, D l'ensemble de leurs domaines et C l'ensemble de contraintes. Pour un ensemble de variables de décision $x_i \in X$ et leurs domaines respectifs $D_i \in D$, une contraintes est une relation d'arité k entre k variables de X. Une solution de P est un assignement des variables de X tel que toutes les contraintes soient satisfaites. Une contrainte est dite violée si elle n'est pas satisfaite. De plus, les solveurs de contraintes peuvent aussi optimiser la solution en minimisant (ou maximisant) une fonction objectif.

Dans notre approche, la transformation est augmentée en ajoutant des contraintes entre les propriétés des éléments du modèle cible ainsi qu'une fonction à optimiser. Un solveur de contraintes assure que pour un modèle source le modèle cible reste consistant et appartiennent donc à l'ensemble des modèles cibles.

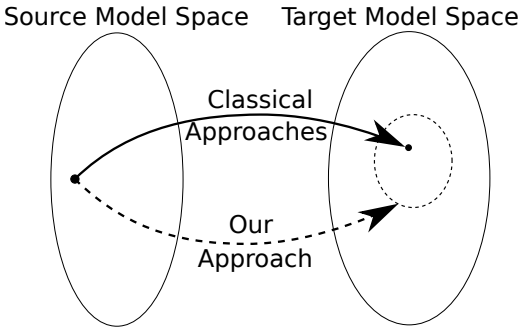


FIGURE 7 – D'un modèle à un ensemble de modèles

La Figure 7 résume la différence entre notre approche et les approches classiques. Les deux ellipses avec des bords pleins correspondent aux espaces de modèles sources et cibles. Les points représentent des modèles de $M_\bullet$. L'ellipse avec un concours en poin-

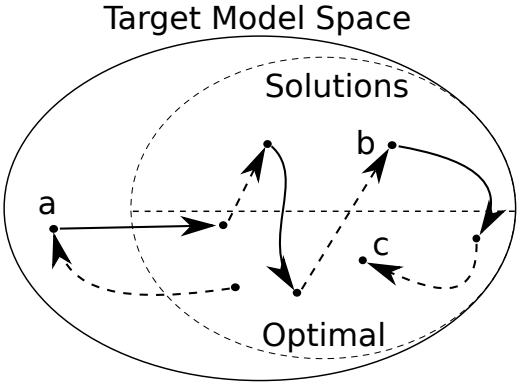


FIGURE 8 – Exploration de solutions

tillés représente l'ensemble des modèles cibles défini en intention par notre approche.

3.2 Stratégies d'exploration

Nous avons défini l'exploration d'un espace de modèles. Nous définissons maintenant l'*exploration d'un ensemble de modèles cibles* comme une séquence de deltas appliqués à un modèle de l'ensemble des modèles et qui finit sur un autre modèle de ce même ensemble. L'utilisateur ne voit qu'un seul modèle cible à la fois. Pour explorer, l'utilisateur peut modifier la vue. Après une mise à jour, le modèle cible peut être dans trois états différents comme illustré dans la Figure 8. Cette figure correspond à une version zoomée de la Figure 7 se concentrant sur l'espace des modèles cibles. Les deux ellipses ont la même signification qu'avant : celle avec le contour plain correspond à l'espace entier, tandis que l'autre correspond à l'ensemble des modèles cibles. De plus, l'ensemble des modèles est séparé en deux sous ensembles horizontalement, la partie du bas correspond aux modèles optimaux tandis que la partie au haut contient les modèles valides mais non optimaux. Chaque point correspond à un modèle totalement instancié et les flèches à des deltas. Les trois états possibles après une mise à jour sont :

1. **Invalide :** l'utilisateur a modifié la vue d'une façon qui met le modèle cible en dehors de l'ensemble (ex. le modèle `a` dans la Figure 8). Il y a deux solutions pour compléter le chemin vers un modèle valide. Le solveur reçoit une demande de résolution pour le modèle mis à jour. En fonction des contraintes, le solveur peut retourner une solution qui est utilisée pour *réparer* le modèle cible, ceci correspond à la flèche sortant du modèle `a`. Il peut aussi retourner l'ancienne solution, ceci est équivalent à ignorer la mise à jour de l'utilisateur.
2. **Non optimal :** l'utilisateur a modifié le modèle d'un façon qui le rend toujours valide mais non optimal pour le solveur (ex. modèle `b`). Dans ce cas, le solveur est utilisé pour mettre à jour la solution de façon à ce qu'elle redevienne optimale.
3. **Optimal :** l'utilisateur a modifié le modèle et celui-ci est toujours valide et optimal, dans ce cas le solveur ne fait rien (ex. modèle `c`).

Comme présenté dans cette section, la programmation par contrainte permet de facilement résoudre des situations complexes pour la transformation de modèles. Nous sommes capable, avec notre approche, de spécifier l'ensemble de modèles cibles, ainsi que de proposer un moyen de l'explorer.

4 Relier le modèle cible au solveur de contraintes

Pour écrire des contraintes entre les propriétés du modèle cible il est nécessaire de trouver une façon de combiner ces propriétés à des variables de décision. Pour cela, nous introduisons la *variable pont*. Avec celle-ci, nous proposons aussi une représentation intermédiaire adaptée.

4.1 Variable pont

Une variable pont peut être vue comme une relation bijective entre une propriété du modèle cible et une variable de décision.

Afin de pouvoir représenter des contraintes avec ces variables nous avons besoin d'introduire un modèle intermédiaire qui supporte le concept de variable pont. Il existe plusieurs langages de contraintes génériques tels que MiniZinc [19], mais ceux-ci ne permettent pas de facilement représenter nos contraintes. La même remarque tient pour d'autres solveurs tels qu'Alloy [9] ou Choco [21].

Ainsi, afin de pouvoir utiliser différents solveurs de contraintes nous présentons un modèle de contraintes pivot qui pourra ensuite être transformé en contraintes comprises par le solveur cible. Cette approche utilisant un modèle intermédiaire est similaire à celle de MiniZinc qui utilise FlatZinc comme langage de contraintes de bas niveau.

Le métamodèle proposé est construit autour de la notion de variable pont. Une version simplifiée est montrée dans la Figure 9. La `BridgeVariable` est responsable de maintenir la propriété et la variable qui lui correspont synchronisées. Lorsque l'une est changée, la variable pont transmet la modification à l'autre. Afin de rester le plus générique possible, les contraintes et expressions utilisent une notation sous forme de prédicats. Chaque prédicat (resp. opérateur) a une arité et une sémantique qui lui sont propre. Ainsi, il est possible d'utiliser les prédicats les plus adaptés au context.

Par exemple, pour notre cas d'étude, nous utilisons des prédicats géométriques tels que `contains`.

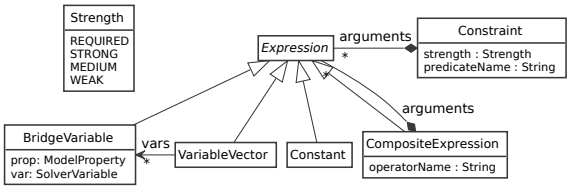


FIGURE 9 – Métamodèle des contraintes

Utiliser un métamodèle intermédiaire pour représenter les contraintes permet de réutiliser tous les outils déjà présents dans la transformation de modèle. En effet, les contraintes générées ne sont qu'un second modèle cible généré avec le modèle cible principal. Transformer ces contraintes nécessite cependant une étape de réécriture afin de les traduire en contraintes propres au solveur cible. Par exemple, réécrire les contraintes géométriques en contraintes linéaires.

4.2 Propagation des modifications au travers de la variable pont

Après que les liens entre les propriétés du modèle et les variables du solveurs aient été établis, chaque modification sur le modèle cible produit cette suite d'évènements.

Une modification est détectée sur une propriété du modèle cible. La variable pont qui lui est associé est notifiée de la mise à jour. Elle propage la nouvelle valeur au solveur qui recalcule une nouvelle solution. Chaque variable modifiée par le solveur notifie sa variable pont du changement, qui à son tour le transmet à la propriété du modèle cible. Après que tous les changements aient été propagés, le système est prêt à recevoir de nouvelles modifications.

4.3 Gestion des collections de propriétés

Comme nous l'avons vu dans la Section 4.1, chaque propriété du modèle cible est associé à une variable pont. Cependant, il est courant que les modèles possèdent des collections. Nous utilisons le `VariableVector` pour représenter ces collections. Un `VariableVector` est une liste de `BridgeVariable` de taille arbitraire. De plus, comme le contenu des propriétés peut évoluer, le nombre de variables de cette liste est voué à changer au cour de l'exécution.

Peu de solveurs supportent les collections de variables qui peuvent évoluer dans le temps. Par conséquent, une étape de réécriture est souvent nécessaire pour adapter ces listes au solveur. Chaque contrainte contenant des `VariableVector`s doit alors être développée en plusieurs contraintes ne contenant plus que des variables classiques. Par exemple, $\{x, y, z\} + \{a, b\} = 0$ doit être développée avant de pouvoir être utilisée par un solveur.

Il y a deux façon classiques de développer les contraintes. La première, le développement cartésien, qui a le même comportement que le développement d'accolades en Bash ou que les listes en compréhension de Python. Par exemple, pour un opérateur binaire, chaque élément de l'expression de gauche est combiné avec chaque élément de l'expression de droite. Avec ce développement, notre contrainte d'exemple est développée en six contraintes : $x + a = 0$, $x + b = 0$, $y + a = 0$, $y + b = 0$, $z + a = 0$ et $z + b = 0$. Le pseudo-code utilisé pour développer les `Constraint`s est donné dans Algorithm 1. Un pseudo-code similaire est utilisé pour les `CompositeExpression`s.

La seconde façon de faire, le développement scalaire, correspond à la fonction zip (ex. en Python ou en Haskell). Pour les opérations binaires, les expressions de gauche et de droites sont combinées par paires grâce à leurs indexes. Développer notre contrainte d'exemple en suivant cette façon de faire ne produit que deux contraintes : $x + a = 0$ et $y + b = 0$. La troisième variable de l'expression de gauche est ignorée car la liste de droite ne contient que deux éléments.

Nous pensons que cette syntaxe permet de gagner en efficacité lors de l'écriture de contraintes.

5 Adapter l'approche à la visualisation de modèles

Les deux sections précédentes présentent l'approche de façon indépendante de l'application. Il y a plusieurs façon d'implémenter cette approche. Cette section définit un ensemble de prérequis dans la Section 5.1 avant de justifier les décisions prises dans la Section 5.3.

5.1 Prérequis

À partir de la description du cas d'étude dans la Section 2 nous déduisons les quatre prérequis suivants.

Prérequis 1 : Définition des contraintes Peu importe l'implémentation de cette approche, il est nécessaire de fournir un moyen de définir les contraintes qui s'appliquent aux propriétés du modèle cible. Les contraintes nécessaires pour le cas d'étude se limitent aux contraintes géométriques (voir Contrainte 1 et 2 de la Section 2.3). De nombreux frameworks, comme JavaFX, utilisent un repère cartésien avec des doubles pour encoder les coordonnées. Ainsi, les contraintes géométriques sur les formes peuvent être exprimées en terme de contraintes arithmétiques sur les doubles.

Algorithm 1 Pseudo-code pour le développement d'une `Constraint`

1: **function** EXPANDCONSTRAINT(c)
2: $exps = [\text{expandExpression}(a) \mid a \in c.arguments]$
3: $pairs \leftarrow [\,]$
4: **if** isScalar($c.predicateName$) **then**
5: **for** $1 \leq i \leq \min_{\forall e \in exps} |e|$ **do**
6: $pairs \leftarrow pairs \cup (exps_{1,i}, ..., exps_{|exps|,i})$
7: **else if** isCartesian($c.predicateName$) **then**
8: $pairs \leftarrow [(e^1, ..., e^{|exps|}) \mid e^1 \in exps_1, ..., e^{|exps|} \in exps_{|exps|}]$
9: **return** [createConstraint($c.predicateName, p$) $\mid p \in pairs$]

Le langage doit donc fournir un moyen d'exprimer ces contraintes géométriques et arithmétiques.

Prérequis 2 : Préférences Pour pouvoir définir des comportements complexes, il est nécessaire de pouvoir donner une préférence dans les contraintes. De plus, quand toutes les contraintes ne peuvent être satisfaites, il faut que celles ayant le poids le plus faible soit violées en priorité.

Prérequis 3 : Le plus petit changement Pour rendre l'exploration de l'espace des solutions plus agréable, il faut ajouter un moyen de limiter les modifications du modèle lors du recalcul d'une nouvelle solution. Ceci est similaire au principe du plus petit changement dans la transformation [17], mais appliqué aux contraintes. Les changements visuels sont particulièrement visibles, il faut pouvoir préciser finement quelles valeurs doivent rester stables. Par exemple, faut-il déplacer ou agrandir un rectangle quand son contenu est déplacé au delà de ses limites ?

Prérequis 4 : Incrémentalité Par incrémentalité nous insistons sur le fait que si le modèle source est modifié, le modèle cible, ainsi que les contraintes, doivent être modifiés en conséquence. Ceci devrait se faire en recalculant le moins de chose possible, idéalement, en en propageant que le nécessaire. Ce prérequis rend possible l'édition des diagrammes.

Remarque : bien que ces prérequis soient guidés par notre cas d'étude, certains d'entre eux s'appliqueraient aussi à d'autres implémentation de l'approche, cependant cela dépasse l'étendu de ce papier.

5.2 Solveurs et langages de contraintes existants

Pour mieux justifier nos choix de la Section 5.3 nous discutons ici des solveurs et langages de contraintes existant par rapport aux prérequis que nous avec introduit dans la Section 5.1.

Langages de modélisation de contraintes

Dans la Section 4.1 nous avons présenté une nouvelle représentation intermédiaire pour les contraintes mais d'autres existent. Nous avons déjà mentionné MiniZinc [19] qui est un langage de contraintes de haut niveau qui est ensuite traduit en FlatZinc, un langage de contrainte de bas niveau. Nous pouvons aussi mentionner XCSP [22], une autre représentation basée sur le XML. Ces langages ont l'avantage d'être compris par de nombreux solveurs tels que Choco [21], JaCoP [14] ou Gecode [24].

Mais, comme nous l'avons dit, bien que puissants, ces langage ne correspondent pas tout à fait à nos besoins. Ils permettent d'écrire facilement des contraintes mais manquent de flexibilité pour exprimer les contraintes dont nous avons besoin. Avec la Figure 9 nous avons introduit un métamodèle minimal pour représenter les contraintes. Ce métamodèle peut représenter n'importe quel type de contraintes, qui peuvent ensuite être traduites en contraintes pour un solveur spécifique.

Solveurs existant

Dans la section précédente, nous avons vu qu'il existe de nombreux solveurs de contraintes. Ces solveurs pourraient être utilisés pour notre approche, cependant ils ne sont pas forcément adaptés à nos besoins. Ils répondent aux prérequis 1 et 2 mais ne peuvent pas efficacement être utilisés avec des problèmes qui changent au cour du temps. Sans support pour l'incrémentalité, il est nécessaire de recommencer la recherche de solution de zéro à chaque modification. Cependant, il existe quelques solveurs répondant à ce besoin, Cassowary [1] ou DeltaBlue [6] sont deux exemples de solveurs dynamiques.

5.3 Choix de design

Maintenant que nos prérequis ont été précisés, nous pouvons choisir comment les satisfaire.

Premièrement, nous choisissons d'utiliser une approche à base de règles déclaratives. Ceci permet de facilement remplir deux prérequis. Prérequis 4 à propos de l'incrémentalité, qui est plus facile à mettre en place avec une approche déclarative qui supporte

le recalcul automatique comme [25, 15, 10] qu'une approche impérative. De plus, les règles sont des abstractions qui encapsulent la source et la cible ainsi que les relations entre celles-ci. Ceci est particulièrement intéressant pour le prérequis 1 à propos des contraintes, ainsi il est aussi possible d'encapsuler les contraintes dans les règles de transformation.

Deuxièmement, nous décidons d'utiliser les contraintes hiérarchiques [28] pour gérer les priorités entres les contraintes imposées par le prérequis 2. Ceci permet d'attacher à chaque contrainte un niveau de préférence parmi : requis, fort, medium et faible. En plus de ce niveau, il est possible d'ajouter un poids qui permet encore d'affiner entre les contraintes au sein d'un même niveau. Il existe plusieurs façon de gérer les contraintes hiérarchiques, mais l'idée générale est que plus le niveau d'une contrainte est élevé plus l'erreur doit être faible. Ainsi, le solveur préfèrera violer une contrainte de niveau inférieur s'il n'existe pas de solution satisfiant toutes les contraintes. Un autre point notable est que les contraintes hiérarchiques ont été conçues pour être utilisées dans des contextes dynamiques [18] [2]. Dans de tels problèmes, il est possible d'ajouter ou de supprimer des contraintes sans avoir à recalculer la totalité de la solution. Ceci est particulièrement important pour notre approche car, dans notre cas d'étude, à chaque modification de l'utilisateur, le solveur doit être notifié et doit pouvoir recalculer une solution rapidement. De plus, à cause de l'incrémentalité (prérequis 4), il est possible d'ajouter de nouveaux éléments au modèle sources et ces nouveaux éléments entrainent l'ajout de nouvelles contraintes.

Enfin, le prérequis 3 peut être satisfait par l'utilisation d'une contrainte spécifique aux contraintes hiérarchiques appelée contrainte de *stay*[6]. Une contrainte de stay s'applique à une variable de décision et permet de minimiser la différence entre la valeur de la variable entre deux résolutions successives. Cette contraintes, comme n'importe quelle autre peut avoir un niveau et un poids. Ce mécanisme permet de finement définir quelles valeurs doivent rester stables d'une résolution à l'autre. Par exemple, sauf autre contrainte, les positions des rectangles doivent rester identiques d'une résolution à l'autre.

6 Implémentation

Dans la section précédente, nous avons expliqué comment nous pouvions adapter notre approche à la visualisation de modèle. Cette section s'attarde sur comment nous avons implémenté cette approche. Pour mieux séparer les idée, nous avons séparer cette implémentation en deux parties, le *front-end*, dans lequel sont exprimées les transformations ainsi que les contraintes et le *back-end* pour les exécuter. La Section 6.1 présente le langage de surface tandis que la Section 6.2 l'utilise sur notre exemple.

6.1 Langage de surface : ATLC

D'après les choix expliqués dans la section précédente, nous choisissons ici quel langage de transformation existant nous allons étendre. En pratique, notre implémentation actuelle abuse de la flexibilité de l'Xtend [3] pour définir un langage de transformation. Ce langage est inspiré d'ATL, et a initialement été créé que comme un prototype pour écrire des transformations incrémentales et par la suite étendu avec le support des contraintes. Cependant, ce langage est inutilement compliqué pour illustrer comment des contraintes peuvent être couplées à une transformation. Ainsi, nous présentons donc ATLC, qui est une extension d'ATL[11]. ATL est un langage de transformation déclaratif basé sur le concept de règle de transformation. Chaque règle est responsable de la génération d'une partie spécifique du modèle cible à partir de parties du modèle source.

Nous nommons notre extension ATLC pour *ATL with Constraints*. Le lecteur intéressé par notre implémentation actuelle, ou pour juger de la similarité avec ce que nous présentons ici peut trouver l'exemple complet et fonctionnel à l'adresse suivante `https://github.com/goldensuneur/demo-ATLC`.

Détaillons quelque contraintes utilisées en ATLC et présentes dans le Listing 1.

— **Contrainte simple** elles permettent d'écrire des contraintes arithmétiques en utilisant des opérateurs binaires classiques tels que $+$, $-$ ou $>$. On trouve deux exemples ligne 8.

— **Stay** (ligne 5), ces contraintes permettent d'ajouter une contrainte de stay sur des variables. Comme expliqué plus haut, les variables sur lesquelles s'appliquent cette contraintes tendent à conserver leurs anciennes valeurs. Il est possible de suggérer des valeurs initiales à ces variables.

— **Minimize**, cette contrainte (ligne 6) force la variable à être le plus proche possible de 0.

— **Contains**, (ligne 9), force une forme géométrique à être contenu dans une autre.

2. [18] utilise le terme *incrémental*. Cependant nous préférons utiliser le terme *dynamique*, qui est aussi accepté [26], et qui évite la confusion avec les transformations incrémentales.

```
1 rule Package2Rectangle {
2   from p : UML!Package
3   to   r : JFX!Rectangle (fill <- 'lightblue', stroke <- 'black'),
4        t : JFX!Text (stroke <- 'black', text <- p.name, origin <- #TOP)
5   with { medium stay r.topLeft suggest {90, 120},
6          weak minimize r.height, weak minimize r.width,
7          t.topLeft = r.topLeft.dx(5),
8          r.width >= 100, r.height >= 100,
9          r contains p.packagedElement }}
```

Listing 1 – Règle de transformation d'un Package UML en un Rectangle et un Text JavaFX

6.2 Exprimer l'exemple en ATLC

Comme mentionné au dessus, l'exemple complet est disponible sur GitHub. Le Listing 1 contient une des règles de transformation. Cette règle transforme un `Package` UML (ligne 2) en un `Rectangle` (ligne 3) et un `Text` (ligne 4) JavaFX. Des propriétés de l'élément source sont liées à des propriétés d'éléments cibles, par exemple le `fill`, le `stroke` et le `text` (lignes 3-4). Les autres propriétés sont utilisées dans le block `with` (lignes 5–9). On applique une contrainte de stay au point en haut à gauche du rectangle et on lui donne une valeur initiale (ligne 5). Sa longueur et sa largeur sont minimisées (lignes 6), mais avec des valeurs minimales (ligne 8). Le point en haut à gauche du texte doit coïncider avec le point en haut à gauche du rectangle avec un petit décalage en x (ligne 7). Enfin (ligne 9), le rectangle doit contenir tous les rectangles correspondant à ses packaged elements (c'est à dire, les autres packages et classes).

7 Travaux Similaires

Les techniques de transformation peuvent être utilisées pour synchroniser un modèle source avec un modèle cible et gérer un ensemble de contraintes. Les contraintes sont un bon moyen d'exprimer des relations complexes qui ne seraient pas exprimables avec des transformations classiques. Ces contraintes permettent d'obtenir un ensemble de modèles cibles. Des approches similaires sont utilisées dans [7] pour générer et explorer un espace de design. Cependant, contrairement à eux, notre approche propose un moyen générique d'intégrer des solveurs existants à des transformations. De même, [20] propose une extension à QVT-R pour gérer les contraintes. Cette extension permet à l'utilisateur d'écrire de façon déclarative des transformations mêlant approche traditionnelles et contraintes. Cependant, leur approche ne permet pas d'explorer l'ensemble des modèles cible. Dans [12], une approche permettant d'intégrer les contraintes dans les workflows de l'IDM est proposée. Mais, comme pour [20], les contraintes sont utilisées pour étendre l'expressivité de la transformation mais il n'y a aucune possibilité d'explorer les résultats. Par contre cette approche est utilisée dans [13] pour construire des transformations bidirectionnelles. La programmation par contrainte peut aussi être utilisée pour vérifier la validité d'une transformation. [3] propose un moyen de vérifier les transformations écrites en ATL. Il est aussi possible d'utiliser les contraintes pour générer des instances du métamodèle source afin de tester les transformations [23]. [16] donne un aperçu des approches de réparation de modèles.

3. `https://www.eclipse.org/xtend/`

8 Conclusion

Ce papier présente une approche permettant de générer un ensemble explorable de modèles cibles grâce à la transformation de modèle. La transformation génère un ensemble de modèle valides pour un modèle source en créant un modèle cible de base sur lequel s'appliquent des contraintes. Cet ensemble de modèles cible peut alors être exploré par l'utilisateur à l'aide d'un solveur.

L'approche peut être appliquée à la visualisation de modèle. Bien que les diagrammes que notre approche génère actuellement sont loin de la qualité de l'état de l'art. Cependant, cette implémentation démontre que l'approche peut fonctionner comme présenté.

Plusieurs aspects de notre approche peuvent être étendus. Par exemple, il serait possible de l'appliquer à d'autres domaines. Il serait aussi possible d'utiliser d'autres solveurs, plus puissant que Cassowary.

Références

[1] Greg J Badros, Alan Borning, and Peter J Stuckey. The Cassowary linear arithmetic constraint solving algorithm. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 8(4) :267–306, 2001.

[2] Frédéric Benhamou, Frédéric Goualard, Éric Languénou, and Marc Christie. Interval constraint

solving for camera control and motion planning. *ACM Trans. Comput. Log.*, 5(4) :732–767, 2004.

[3] Fabian Büttner, Marina Egea, Jordi Cabot, and Martin Gogolla. Verification of ATL Transformations Using Transformation Models and Model Finders. In *ICFEM 2012*, pages 198–213.

[4] Alberto Rodrigues da Silva. Model-driven engineering : A survey supported by the unified conceptual model. *Computer Languages, Systems and Structures*, 43 :139 – 155, 2015.

[5] Zinovy Diskin, Arif Wider, Hamid Gholizadeh, and Krzysztof Czarnecki. Towards a rational taxonomy for increasingly symmetric model synchronization. In *ICMT 2014*, pages 57–73.

[6] Bjorn N. Freeman-Benson, John Maloney, and Alan Borning. An incremental constraint solver. *Commun. ACM*, 33(1) :54–63, January 1990.

[7] Ákos Horváth and Dániel Varró. Dynamic constraint satisfaction problems over models. *Software & Systems Modeling*, 11(3) :385–408, 2012.

[8] Rémi Imbach, Pascal Mathis, and Pascal Schreck. A robust and efficient method for solving point distance problems by homotopy. *Math. Program.*, 163(1-2) :115–144, 2017.

[9] Daniel Jackson. Alloy : a lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(2) :256–290, 2002.

[10] Frédéric Jouault and Olivier Beaudoux. Efficient OCL-based Incremental Transformations. In *16th International Workshop in OCL and Textual Modeling*, pages 121–136, 2016.

[11] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, and Ivan Kurtev. Atl : A model transformation tool. *Science of Computer Programming*, 72(1) :31 – 39, 2008. Special Issue on Second issue of experimental software and toolkits (EST).

[12] Mathias Kleiner, Marcos Didonet Del Fabro, and Patrick Albert. Model search : Formalizing and automating constraint solving in mde platforms. In *ECMFA 2010*, pages 173–188.

[13] Mathias Kleiner, Marcos Didonet Del Fabro, and Davi De Queiroz Santos. Transformation as search. In *ECMFA 2013*, pages 54–69.

[14] Krzysztof Kuchcinski. Constraints-driven scheduling and resource assignment. *ACM Trans. Des. Autom. Electron. Syst.*, 8(3) :355–383, July 2003.

[15] Erhan Leblebici, Anthony Anjorin, Andy Schürr, Stephan Hildebrandt, Jan Rieke, and Joel Greenyer. A comparison of incremental triple graph grammar tools. *Electronic Communications of the EASST*, 67, 2014.

[16] N. Macedo, T. Jorge, and A. Cunha. A feature-based classification of model repair approaches. *IEEE Transactions on Software Engineering*, 43(7) :615–640, 2017.

[17] Nuno Macedo and Alcino Cunha. Least-change bidirectional model transformation with QVT-R and ATL. *Software & Systems Modeling*, 15(3) :783–810, Jul 2016.

[18] Francisco Menezes, Pedro Barahona, and Philippe Codognet. An incremental hierarchical constraint solver. In *PPCP*, volume 93, pages 190–199, 1993.

[19] Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck, and Guido Tack. MiniZinc : Towards a standard CP modelling language. In *International Conference on Principles and Practice of Constraint Programming*, pages 529–543. Springer, 2007.

[20] Andreas Petter, Alexander Behring, and Max Mühlhäuser. Solving constraints in model transformations. In *ICMT 2009*, pages 132–147.

[21] Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. *Choco Documentation.* TASC - LS2N CNRS UMR 6241, COSLING S.A.S., 2017.

[22] Olivier Roussel and Christophe Lecoutre. XML representation of constraint networks : Format XCSP 2.1. *CoRR*, abs/0902.2362, 2009.

[23] Sagar Sen, Benoit Baudry, and Jean-Marie Mottu. Automatic Model Generation Strategies for Model Transformation Testing. In Richard F. Paige, editor, *ICMT 2009*, pages 148–164.

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

[25] Dániel Varró, Gábor Bergmann, Ábel Hegedüs, Ákos Horváth, István Ráth, and Zoltán Ujhelyi. Road to a reactive and incremental model transformation platform : three generations of the VIATRA framework. *Software & Systems Modeling*, 15(3) :609–629, Jul 2016.

[26] Gérard Verfaillie and Narendra Jussien. Constraint solving in uncertain and dynamic environments : A survey. *Constraints*, 10(3) :253–281, Jul 2005.

[27] David Waltz. Understanding line drawings of scenes with shadows. In *The Psychology of Computer Vision*, page pages. McGraw-Hill, 1975.

[28] Molly Wilson and Alan Borning. Hierarchical constraint logic programming. *The Journal of Logic Programming*, 16(3) :277 – 318, 1993.

Étude probabiliste de la contrainte $alldifferent$: résultats préliminaires

Giovanni Lo Bianco[1] *__Xavier Lorca__[1] **Charlotte Truchet**[2] **Vlady Ravelomanana**[3]

[1] IMT Atlantique, 44300 Nantes, France
[2] Université de Nantes, 44000 Nantes, France
[3] Université Paris-Diderot, 75013 Paris, France
giovanni.lo-bianco@imt-atlantique.fr

Résumé

Cet article présente un modèle probabiliste pour la contrainte $alldifferent$. Une approche probabiliste permet d'étudier la structure combinatoire d'un problème. Plus particulièrement, nous nous intéressons à l'existence de solutions dans une instance de $alldifferent$ ainsi qu'à leur dénombrement. De telles informations sont utilisées dans certaines heuristiques de choix de variables et de valeurs. Cette approche probabiliste permet de calculer des indicateurs qui donnent des informations sur la structure combinatoire d'une instance.

1 Introduction

Dans cet article, nous voulons étudier la structure combinatoire de certains problèmes pour être plus efficace lors de l'exploration de l'espace de recherche ainsi que dans le déroulement de la propagation. Une approche probabiliste permet ce genre d'étude. Nous nous intéressons plus particulièrement aux contraintes de cardinalité. Les contraintes de cardinalité regroupent les contraintes qui portent sur le nombre d'occurrences des valeurs dans une solution. La contrainte *global_cardinality* [7] est la plus générale des contraintes de cardinalité. Elle permet de modéliser un problème de flot dans lequel nous cherchons à assigner toutes les variables à des valeurs de leur domaine, de sorte que, dans une solution, le nombre d'occurrences de chaque valeur est compris entre deux bornes, propres à chacune. Souvent, une contrainte de cardinalité peut être vue comme un cas particulier de *global_cardinality*. Nous proposons d'effectuer cette étude pour la contrainte $alldifferent$ [8], qui contraint chaque valeur à n'apparaître qu'au plus une fois. Le modèle proposé dans cette étude a pour but de pouvoir être appliqué de manière similaire sur d'autres contraintes de cardinalité, leur structure étant similaire.

Ce modèle probabiliste va nous permettre d'estimer le nombre de solutions pour une instance de $alldifferent$. Le dénombrement de solutions pour les contraintes a déjà été étudié, notamment par Pesant et al. [5], qui présentent une heuristique de recherche, nommée counting-based search, qui propose d'explorer en premier les branches de l'arbre de recherche dans lesquelles on estime qu'il y a le plus de solutions. Dénombrer les solutions d'un problème est au moins aussi dur que le problème lui-même. C'est pourquoi Pesant et al. [5] proposent une borne supérieure, moins coûteuse à calculer, du nombre de solutions pour une instance de $alldifferent$ et l'utilisent comme une heuristique pour guider la recherche.

Cette étude est similaire à [1], qui présente aussi un modèle probabiliste pour $alldifferent$: en probabilisant les domaines des variables, les auteurs calculent la probabilité et l'espérance qu'une contrainte $alldifferent$ soit bornes-consistante (BC) et exhibent deux régimes asymptotiques en fonction du ratio (nombre de variables) / (nombre de valeurs). La principale différence est que Boisberranger et al. [1] proposent de probabiliser le domaine des variables en se restreignant au cas où les domaines sont des intervalles, alors que dans notre étude nous ne posons aucune condition sur les domaines, ils peuvent être une union d'intervalles. Aussi Boisberranger et al. [1] s'intéressent à la probabilité qu'une instance de $alldifferent$ soit consistante aux bornes et démontrent qu'il n'est pas toujours nécessaire d'appliquer l'algorithme de filtrage. Quant à cette étude,

*Papier doctorant : Giovanni Lo Bianco[1] est auteur principal.

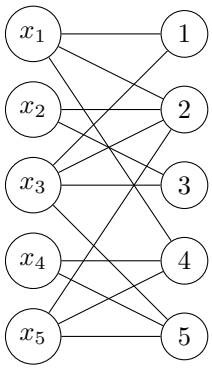


FIGURE 1 – Graphe des domaines de l'exemple 2.1

nous nous intéressons à l'existence et au nombre de solutions et n'allons pas étudier quand il est intéressant ou non d'appliquer les algorithmes de filtrage de $alldifferent$.

2 Préliminaires

Soit $X = \{x_1, \ldots, x_n\}$, l'ensemble des variables. Pour chaque variable $x_i \in X$, on note D_i son domaine et $Y = \bigcup_{i=1}^{n} D_i = \{y_1, \ldots, y_m\}$, l'union des domaines. Nous définissons maintenant la contrainte $alldifferent$ ainsi que le graphe qui la modélise.

2.1 La contrainte $alldifferent$

Définition 2.1 ($alldifferent$ [6])**.** On définit $alldifferent(X)$, le problème de satisfaction, dans lequel on cherche à instancier chaque variable $x_i \in X$ à une valeur de son domaine D_i tel que $\forall y_j \in Y$, y_j n'a été choisi qu'au plus une fois. On définit formellement l'ensemble des tuples admis :

$$\mathcal{S}_X = \{(d_1, \ldots, d_n) \in D_1 \times \ldots \times D_n | \forall i, j \in \{1, ..., n\}, i \neq j \Leftrightarrow d_i \neq d_j\}$$

Définition 2.2 (Graphe des domaines)**.** Soit $G_X = G(X \cup Y, E)$, le graphe des domaines de X avec $E = \{(x_i, y_j) | y_j \in D_i\}$. G_X est un graphe biparti représentant le domaine de chaque variable. Il y a une arête reliant une variable x_i à une valeur y_j ssi $y_j \in D_i$

Exemple 2.1. *Soit* $X = \{x_1, x_2, x_3, x_4, x_5\}$ *avec* $D_1 = \{1, 2, 4\}$, $D_2 = \{2, 3\}$, $D_3 = \{1, 2, 3, 5\}$, $D_4 = \{4, 5\}$ *et* $D_5 = \{2, 4, 5\}$. *On obtient alors le graphe des domaines* G_X *tel que sur la Figure 1. Les tuples* $\{1, 3, 5, 4, 2\}$ *et* $\{1, 2, 3, 5, 4\}$ *sont deux solutions de* $alldifferent(X)$.

2.2 Existence et dénombrement de solutions

Dans cette sous-section, nous nous intéressons à caractériser mathématiquement l'existence de solution pour une instance de $alldifferent$, puis de compter le nombre de solutions.

L'intérêt du graphe des domaines réside dans le fait qu'un couplage couvrant toutes les variables de X correspond à une solution de $alldifferent(X)$ [8].

Proposition 2.1. *L'ensemble des solutions de* $alldifferent(X)$ *est en bijection avec l'ensemble des couplages dans* G_X *couvrant toutes les variables de* X.

Pour dénombrer les solutions de $alldifferent(X)$, il faut alors dénombrer les couplages de G_X couvrant X. Nous introduisons maintenant la notion de matrice de biadjacence ainsi que le permanent d'une matrice. Les définitions et propriétés suivantes peuvent être trouvées dans le livre *Matching Theory*, de L. Lovàsz et M.D. Plummer [4].

Définition 2.3 (Matrice de biadjacence)**.** Soit $G(U \cup V, E)$, un graphe biparti. On définit $\mathcal{B}(G) = (b_{ij})$, sa matrice de biadjacence, tel que $\forall u_i \in U, \forall v_j \in V$, $b_{ij} = 1$, si $(u_i, v_j) \in E$ et $b_{ij} = 0$, sinon.

La matrice de biadjacence du graphe des domaines de la Figure 1 est :

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

Définition 2.4 (Permanent)**.** Soit $A = (a_{ij}) \in \mathbb{M}_{n,n}$, une matrice carrée et $\mathfrak{S}_n$, le groupe des permutations sur $\{1, \ldots, n\}$. Le permanent de A est défini comme suit :

$$Perm(A) = \sum_{\sigma \in \mathfrak{S}_n} \prod_{i=1}^{n} a_{i\sigma(i)} \quad (1)$$

Il est possible de calculer le permanent d'une matrice rectangulaire. Soit $A \in \mathbb{M}_{n,m}$ avec $m > n$, on construit $A' \in \mathbb{M}_{m,m}$, en rajoutant $m - n$ lignes remplies de 1. On calcule le permanent de A' avec la formule de la Définition 2.4, puis on prend en compte le biais créé par l'ajout des lignes remplies de 1 :

$$Perm(A) = \frac{Perm(A')}{(m-n)!}$$

On définit un graphe biparti équilibré comme étant un graphe biparti dont les deux parties possèdent le même nombre de nœuds et on définit un couplage parfait, un couplage qui couvre tous les nœuds d'un

graphe. Il est possible de trouver un couplage parfait dans un graphe biparti G, seulement si G est équilibré. Dans ce cas, la Proposition 2.2 donne une manière de compter le nombre de couplages parfaits, qu'on notera $\#PM(G)$ [5] :

Proposition 2.2. *Soit G, un graphe biparti équilibré et $\mathcal{B}(G)$, sa matrice de biadjacence. On note $\#PM(G)$, le nombre de couplages parfaits dans G, alors :*

$$\#PM(G) = Perm(\mathcal{B}(G)) \tag{2}$$

De la même manière qu'on peut calculer le permanent d'une matrice rectangulaire, la Proposition 2.2 reste vraie pour les graphes bipartis non-équilibrés, excepté que $\#PM(G)$ ne désigne plus le nombre de couplages parfaits, mais le nombre de couplages couvrant la plus petite partie de G. Soit $G(U \cup V, E)$, un graphe biparti non-équilibré avec, $|U| = n$, $|V| = m$ et $m > n$. On construit G', le graphe biparti équilibré en rajoutant $m - n$ nœuds dans U, chaque nouveau nœud étant relié à tous les nœuds de V. On peut alors calculer le nombre de couplages de G couvrant U :

$$\begin{aligned}\#PM(G) &= \frac{\#PM(G')}{(m-n)!}) \\ &= \frac{Perm(\mathcal{B}(G'))}{(m-n)!} = Perm(\mathcal{B}(G))\end{aligned}$$

On a vu qu'un couplage couvrant X dans G_X correspond à une solution de $alldifferent(X)$ et on sait maintenant comment compter le nombre de couplages de G_X couvrant X. On en déduit le Corollaire 2.1 :

Corollaire 2.1. *Le problème $alldifferent(X)$ admet $Perm(\mathcal{B}(G_X))$ solutions.*

Exemple 2.2. *L'instance de $alldifferent(X)$ de l'exemple 2.1 possède*

$$Perm\left(\begin{pmatrix}1&1&0&1&0\\0&1&1&0&0\\1&1&1&0&1\\0&0&0&1&1\\0&1&0&1&1\end{pmatrix}\right) = 8$$

solutions.

3 Un modèle probabiliste pour $alldifferent$

Dans cette partie, nous introduisons notre modèle probabiliste pour les contraintes de cardinalité, appliqué à la contrainte $alldifferent$.

Nous allons probabiliser les domaines des variables de la manière suivante : pour tout $x_i \in X$ et pour tout $y_j \in Y$, l'évènement $\{y_j \in D_i\}$ a une probabilité $p \in [0, 1]$ fixée de se produire et tous ces évènements sont **indépendants** entre eux :

$$P(\{y_j \in D_i\}) = p \in [0, 1] \tag{3}$$

Ce modèle est assez simple mais peut nous donner des informations sur le nombre de solutions, en se basant seulement sur la densité d'arcs dans le graphe des domaines.

3.1 Espérance du nombre de solutions

Nous proposons de calculer l'espérance du nombre de solutions comme indicateur de la structure combinatoire d'une instance :

Proposition 3.1. *Soit $X = \{x_1, \ldots, x_n\}$, l'ensemble des variables et $Y = \{y_1, \ldots, y_m\}$, l'ensemble des valeurs avec $m \geq n$, alors l'espérance du nombre de solutions de $alldifferent(X)$ est :*

$$E(|\mathcal{S}_X|) = \frac{m! \cdot p^n}{(m-n)!} \tag{4}$$

Démonstration. Pour simplifier les notations lors de cette preuve, nous noterons $B = \mathcal{B}(G_X)$. On construit $B' \in \mathbb{M}_{m,m}$, en rajoutant $m - n$ lignes remplies de 1 à B. On a alors :

$$\begin{aligned}E(Perm(B)) &= E\left(\frac{Perm(B')}{(m-n)!}\right) \\ &= E\left(\frac{1}{(m-n)!}\sum_{\sigma\in\mathfrak{S}_m}\prod_{i=1}^{m} B'_{i\sigma(i)}\right) \\ &= E\left(\frac{1}{(m-n)!}\sum_{\sigma\in\mathfrak{S}_m}\prod_{i=1}^{n} B_{i\sigma(i)}\right)\end{aligned}$$

car $\forall i > n, B'_{ij} = 1$ et $\forall i \leq n, B'_{ij} = B_{ij}$. L'espérance est un opérateur linéaire et $\forall \sigma \in \mathfrak{S}_m, \forall i \in \{1, \ldots, n\}$ les variables aléatoires $B_{i\sigma(i)}$ sont deux à deux indépendantes, donc :

$$\begin{aligned}E(Perm(B)) &= \frac{1}{(m-n)!}\sum_{\sigma\in\mathfrak{S}_m}\prod_{i=1}^{n} E\left(B_{i\sigma(i)}\right) \\ &= \frac{1}{(m-n)!}\sum_{\sigma\in\mathfrak{S}_m}\prod_{i=1}^{n} p \\ &= \frac{m! \cdot p^n}{(m-n)!}\end{aligned}$$

□

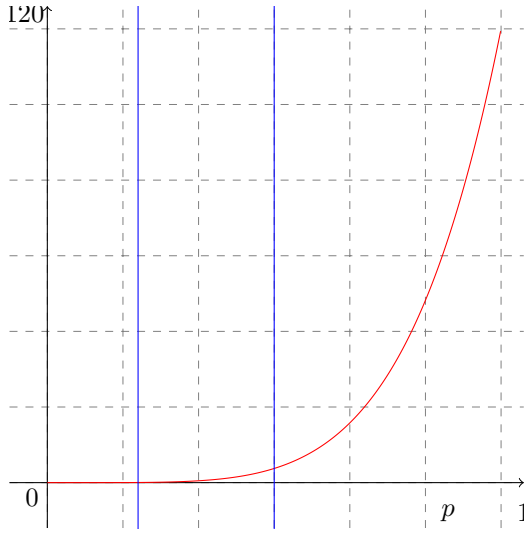


FIGURE 2 – Esperance du nombre de solutions en fonction de p avec $n = m = 5$

Lorsque $n = m$, l'espérance du nombre de couplages parfaits est $n! \cdot p^n$. Ce résultat est connu est est démontré dans [2]. La Proposition 3.1 étend ce résultat au cas des graphes bipartis non-équilibrés.

Dans l'exemple 2.1, nous sommes dans le cas $n = m = 5$ et il y a 14 arcs dans le graphe sur un maximum possible de 25, la densité d'arcs est donc de $\frac{14}{25} = 0,56$. On espère donc avoir $5! \cdot 0,56^5 = 6,61$ solutions. En effet la densité d'arc sert ici d'estimateur du paramètre p.

Nous nous intéressons à caractériser les instances de $alldifferent$ selon l'espérance de leur nombre de solutions.

3.2 Comportement du nombre de solutions

La Figure 2 montre l'évolution du nombre de solutions en fonction de p pour $n = m = 5$.

De manière attendue, lorsque p est petit, on espère avoir aucune ou très peu de solutions. L'espérance du nombre de solutions croît exponentiellement jusqu'à atteindre le nombre maximale de combinaisons. On peut caractériser trois types d'instances différentes (représenté par les trois zones dans la Figure 2). Il y a les instances pour lesquelles l'espérance du nombre de solutions est très inférieure à 1 (à gauche), et donc pour lesquelles, on est à peu près sur qu'il n'existe pas de solutions, il y a les instances pour lesquelles on est à peu près sûr qu'il existe une solution (à droite) puis les instances où il est très difficile de décider s'il en existe ou pas (au milieu). Autrement dit, dans la zone de gauche et de droite, on est capable de décider s'il existe une solution ou non.

Ainsi, nous avons un indicateur pour l'existence d'une solution pour une instance de $alldifferent$. Cet indicateur peut être utilisé pour guider la recherche. Par exemple, sur le même principe que le $first - fail$ [3], on peut d'abord explorer les branches de l'arbre de recherche pour lesquelles, on est presque sûr qu'il n'existe pas de solutions.

4 Conclusion

Cette étude propose un modèle probabiliste pour la contrainte $alldifferent$, à partir duquel on peut calculer certains indicateurs. Ces indicateurs nous donnent des informations sur la structure combinatoire de l'instance que l'on veut résoudre. Nous proposons d'utiliser l'espérance du nombre de solutions comme indicateur. L'intuition est similaire à ce que propose Pesant et al. [5]. Il serait intéressant de comparer ces deux approches. La poursuite de cette étude consistera à développer d'autres indicateurs et à les mettre en place pour résoudre des instances de $alldifferent$. Aussi, ce modèle peut être adapté pour d'autres contraintes de cardinalité.

Références

[1] Jérémie Du Boisberranger, Danièle Gardy, Xavier Lorca, and Charlotte Truchet. When is it worthwhile to propagate a constraint ? A probabilistic analysis of alldifferent. 2013.

[2] P. Erdos and A. Renyi. On random matrices. *Publication of the Mathematical Institute of the Hungarian Academy of Science*, 1963.

[3] Robert M. Haralick and Gordon L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artif. Intell.*, 14(3) :263–313, 1980.

[4] Lászlo Lovasz and Michael D. Plummer. *Matching Theory*. American Mathematical Society, 2009.

[5] Gilles Pesant, Claude-Guy Quimper, and Alessandro Zanarini. Counting-based search : Branching heuristics for constraint satisfaction problems. *J. Artif. Intell. Res.*, 43 :173–210, 2012.

[6] Jean-Charles Régin. A filtering algorithm for constraints of difference in csps. pages 362–367, 1994.

[7] Jean-Charles Régin. Generalized arc consistency for global cardinality constraint. 1996.

[8] Willem Jan van Hoeve. The alldifferent constraint : A survey. *CoRR*, cs.PL/0105015, 2001.

Apprentissage de Top-k clauses par dominance pour les Solveurs SAT Modernes

Jerry Lonlac[1,2,3] **Engelbert Mephu Nguifo**[1]

[1] Université Clermont Auvergne, CNRS, LIMOS, F-63000 Clermont-Ferrand, France
[2] CRIL CNRS, UMR 8188, Université Lille-Nord de France, Lens, France
[3] Université de Douala, ENSET, Département de Génie Informatique, Cameroun
lonlac@cril.fr, engelbert.mephu_nguifo@uca.fr

Résumé

Dans ce papier, nous proposons une nouvelle stratégie pour identifier les clauses apprises les plus pertinentes à maintenir par les solveurs SAT modernes CDCL durant la recherche, sans favoriser ou exclure aucune des mesures de pertinence des clauses apprises proposées dans la littérature. Nous adoptons la notion de relation de dominance entre les différentes clauses apprises pour sélectionner judicieusement à chaque étape de réduction de la base des clauses apprises k clauses apprises non dominées (appelées top-k) et nous supprimons ensuite toutes les clauses apprises dominées par au moins l'une des top-k clauses apprises. Les top-k clauses apprises correspondent aux k clauses apprises préférées par rapport à la relation de dominance avec l'ensemble des mesures de pertinence considérées. Notre approche s'affranchit du problème de la diversité des résultats et cherche un compromis entre les performances de ces mesures. De plus, l'approche proposée évite un autre problème non trivial qui est celui de déterminer le nombre de clauses à supprimer à chaque étape de réduction de la base des clauses apprises.

1 Introduction

Le problème SAT, i.e., le problème de décider si une formule booléenne sous forme normale conjonctive (CNF) est satisfiable ou non est central en Informatique et en Intelligence Artificielle incluant les problèmes de satisfaction de contraintes (CSP), planification, raisonnement non-monotone, vérification de logiciels, etc. Aujourd'hui, le problème SAT a gagné une audience considérable avec l'avénement d'une nouvelle génération de solveurs capables de résoudre des grandes instances issues du codage des applications du monde réel. Ces solveurs, communément appelés *solveurs SAT modernes* [27, 13] ou solveurs SAT CDCL (Conflict Driven Clause Learning) se sont avérés être très efficaces pour résoudre les instances SAT issues des applications du monde réel. Ils sont construits en intégrant quatre composants majeurs à la procédure *DPLL (Davis, Putnam, Logemann and Loveland)* classique [11] : les structures de données paresseuses [27], les heuristiques de choix de variables basées sur les activités (comme VSIDS), [27], les stratégies de redémarrage [17], et l'apprentissage de clauses [31, 27]. Bien qu'une combinaison astucieuse de ces composants contribue à améliorer l'efficacité des solveurs SAT modernes [23], l'apprentissage de clauses reste connu comme le composant le plus important

L'idée principale de l'apprentissage de clauses est qu'au cours du processus de propagation unitaire, quand une branche courante de l'arbre de recherche mène à un conflit, les solveurs SAT modernes apprennent une clause conflit qui aide la propagation unitaire à découvrir l'une des implications manquées à un niveau plus haut de l'arbre de recherche. Cette clause conflit exprime les causes du conflit et est utilisée pour élaguer l'espace de recherche. L'apprentissage de clauses, aussi connu dans la littérature comme *"Conflict Driven Clause Learning" (CDCL)* se réfère maintenant au schéma d'apprentissage le plus connu et utilisé premier "UIP" ("Unique Implication Point") [37], initialement intégré dans le solveur SAT *Grasp* [31] et efficacement mis en oeuvre dans *zChaff* [27]. La plupart des solveurs SAT intégrent ce solide schéma d'apprentissage. Comme à chaque conflit, les solveurs CDCL apprennent une nouvelle clause qui est ajoutée à la base des clauses apprises, et que le nombre de clauses apprises se rèvèle être exponentiel dans le pire des cas, il est nécessaire de supprimer certaines

clauses apprises afin de maintenir une base de clauses de taille polynomiale. Cependant, supprimer trop de clauses peut rendre l'apprentissage inefficace, et garder trop de clauses également peut altérer l'efficacité de la propagation unitaire. Il est nécessaire d'identifier les clauses apprises les plus pertinentes à maintenir pour la suite de la recherche.

La gestion de la base des clauses apprises a fait l'objet de plusieurs études et plusieurs stratégies de gestion de la base des clauses apprises furent proposées dans la littérature [27, 31, 13, 5, 4, 18].

Considérant l'impact de l'apprentissage de clauses sur la résolution pratique des instances SAT, notre motivation pour ce travail vient de l'observation que l'utilisation de différentes stratégies de suppression fondées sur la pertinence donne des performances différentes. Notre but est de tirer profit de plusieurs stratégies de suppression de clauses apprises en cherchant un compromis entre ces stratégies à travers une relation de dominance.

Dans ce papier, nous intégrons un point de vue basé sur les préférences de l'utilisateur dans le processus de résolution SAT. A cet effet, afin d'apprendre les clauses de manière à ne pas se fixer un seuil pour le nombre de clauses à apprendre, nous intégrons dans le processus SAT l'idée de maximal d'un ensemble de vecteurs [30], aussi appelée requêtes *skyline* [9] dans le contexte des bases de données, *motifs dominants* [33, 34] en fouille de motifs, ou *règles d'association non dominées* [10] pour la fouille des règles d'association. De telles idées ont aussi attirées une attention considérable du fait de leur importance dans la prise de décision multi-critères. Etant donné un ensemble de clauses, l'ensemble `skyline` contient les clauses qui ne sont dominées par aucune autre clause. A cet effet, étant donné un ensemble de mesures de pertinence de clauses apprises, nous cherchons à trouver un compromis entre les différentes mesures.

Le traitement `skyline` ne réquiert aucune fonction de sélection de seuil, et la propriété formelle de dominance satisfait par les clauses `skyline` donne aux clauses un intérêt global avec des sémantiques facilement comprehensibles par l'utilisateur. Cette notion de `skyline` a été développée pour des applications de base de données et de fouille de données, cependant elle n'a jamais été utilisée à de fins de SAT. Dans ce papier, nous adaptons cette notion au processus de gestion des clauses apprises pour proposer une solution au problème de diversité entre les mesures de pertinence des clauses apprises.

Dans notre travail précédent [25], nous proposons de chercher à chaque étape de réduction de la base des clauses apprises, la clause apprise de référence courante [25] (top-1 clause apprises non dominées dans ce papier) par rapport à un ensemble de mesures de pertinence de clauses et de supprimer toutes les clauses apprises dominées par cette clause apprise de référence. Dans ce papier, nous étendons cette idée en considérant plutôt k clauses apprises de référence courantes, c'est à dire les top-k clauses apprises non dominées à chaque étape de nettoyage de la base des clauses apprises et supprimer toutes les clauses apprises dominées par au moins l'une des top-k clauses apprises non dominées.

Le papier est organisé comme suit : nous présentons premièrement quelques stratégies efficaces de suppression de clauses apprises fonées sur la pertinence utiliséss dans la littérature. Ensuite, notre stratégie de suppression de clauses apprises fondée sur la relation de dominance entre différentes stratégies est présentée dans la section 4. Finalement, avant de conclure, les résultats expérimentaux démontrant l'efficacité de notre approche sont présentés.

2 Autour du problème SAT et travaux connexes

SAT est un problème de décision NP-complet pour lequel le plus grand éffort de recherche a été consenti pour le développement des algorithmes sophistiqués avec des implémentations hautement optimisées. La pluspart des algorithmes de résolution SAT sont hautement complexes et les études empiriques permettent d'évaluer et de comparer leur performance. Une compétition SAT organisée chaque année permet d'une part une évaluation objective de ces algorithmes SAT, et d'autre part une promotion de nouveaux solveurs SAT. Cependant, un solveur peut être meilleur que d'autres solveurs sur la résolution de certaines instances d'une classe donnée, mais dramatiquement pire sur d'autres instances. Il n'existe de nos jours aucun solveur efficace sur toutes les classes d'instances SAT, différents solveurs sont meilleurs sur des instances différentes. Ainsi, de cette observation, plutôt que de suivre les approches traditionnelles choisissant le meilleur solveur pour une classe d'instances donnée, certains travaux sur SAT construisent des algorithmes portfolio [16, 24, 36] qui sélectionnent les solveurs par instance en utilisant des modèles dures empiriques pour la prédiction du temps d'éxecution [20]. Ces modèles dures empiriques sont généralement construits en utilisant les techniques d'apprentissage automatique. Un prédicteur d'un temps d'exécution d'algorithme sur une instance de problème donnée utilise un ensemble de caractéristiques de l'instance et les performances passées de l'algorithme [29]. Pour ce faire, les instances sont regroupées en classes en fonction de leurs caractéristiques, et le meilleur algorithme

est trouvé pour chaque instance. Par la suite, étant donnée une instance en entrée, ces caractéristiques sont calculées et elle est assignée à une classe (en utilisant le modèle de prédiction précédement construit), et elle est résolue par le solveur correspondant assigné à cette classe. Au cours de ces récentes années, certains travaux ont été effectués sur une meilleure caractérisation des instances SAT afin de les classifier efficacement[2, 28]. En effet, l'ensemble des caractéristiques utilisées pour construire les classifieurs des instances SAT joue un rôle crucial. Dans [1], les auteurs utilisent certaines caractéristiques de structures d'instances SAT industrielles pour construire certains classifieurs de familles d'instances SAT industrielles. L'efficacité de ces classifieurs est mesurée en les comparant à d'autres ensembles de caractéristiques d'instances SAT communément utilisées dans les approches de résolution SAT de type portfolio. Dans [14], il est prouvé que le ratio entre les temps d'exécution requis par un solveur SAT CDCL et par un solveur spécialisé aléatoire est lié à la structure sans échelle de la formule booléenne. Les séries des compétitions SAT internationales stimulent le développement des implḿentations efficaces conduisant à prendre les solveurs plus complexes avec plusieurs paramètres. Ces paramètres permettent aux solveurs d'être personnalisés pour une famille particulière d'instances SAT. Cependant, dans les compétitions SAT, les solveurs sont exécutés en utilisant une simple configuration de paramètres par défaut fourni par les auteurs pour toutes les instances du benchmark dans une compétition donnée. Cela pose un problème pour une utilisation pratique de SAT par les utilisateurs qui ne se soucient que des performances sur une application particulière et peuvent investir un certain temps dans le réglage des paramètres du solveur pour cette application. Ainsi, une nouvelle compétition des solveurs SAT configurables (CSSC) [19] a été concu afin d'évaluer les performances du solveur pour chaque instance SAT après avoir configuré ses paramètres. La compétition CSSC prend en compte le fait que les procédures efficaces de configuration d'algorithme peuvent personnaliser automatiquement les solveurs pour une distribution donnée d'instances benchmak. Plus précisement, pour chaque type d'instances SAT et pour chaque solveur SAT, une phase de configuration hors ligne fixée dans le temps, automatisée détermine les réglages des paramètres du solveur optimisé pour des performances élevées sur ce type d'instances SAT. Ensuite, la performance du solveur sur le type d'instance est évaluée avec ces paramètres, et le solveur avec la meilleure performance est vainqueur.

D'autres travaux de la littérature portant sur le composant apprentissage de clauses se sont plutôt focalisés sur la minimisation de la clause apprise. Ces travaux visent le plus souvent à réduire le nombre de litéraux dans les clauses apprises [7, 32, 6, 21]. Dans cette direction, plus récemment, dans [26], les auteurs proposent pour les solveurs SAT CDCL, une nouvelle approche in-processing de minimisation de clause apprise capable de supprimer les litŕaux rédondants des clauses apprises. Cette approche est basée sur la propagation des contraintes booléenne, ou plus précisément propagation unitaire qui consomme trop de temps sur des larges instances. L'intgration de cette approche dans les meilleures solveurs SAT de l'état de l'art permet de résoudre un plus grand nombre d'instances prises des catégories application et combinatoires difficiles des compétitions SAT 2014 et 2016. Le solveur SAT `Maple_LCM_Dist` vainqueur de la dernière compétition SAT 2017 sur la catégorie d'instances industrielles implémente cette approche de minimisation.

Bien que les approches proposées pour la minimisation des clauses apprises permettent d'améliorer les performances des solveurs SAT CDCL, l'ensemble des clauses apprises reste de taille exponentielle dans le pire des cas. Ainsi, plusieurs stratégies de gestion de la base des clauses apprises ont été conçues pour faire face à ce problème d'explosion combinatoire.

3 Sur les stratégies de gestion de la base des clauses apprises

Dans cette section, nous présentons quelques mesures de pertinence des clauses apprises exploitées par la plupart des solveurs SAT de la littérature.

Le solveur SAT CDCL le plus populaire *Minisat* [13] considère comme pertinentes les clauses les plus impliquées dans les récentes analyses de conflits et élimine les clauses apprises dont l'implication dans les récentes analyses de conflits est marginale. Une autre stratégie appelée *LBD* pour *Literal Block Distance* fut proposée dans [5]. La mesure *LBD* dont l'efficacité a été prouvée empiriquement est aussi exploitée par la plupart des meilleurs solveurs SAT de l'état de l'art (*Glucose, Lingeling [8]*). Cette mesure utilise le nombre de niveaux différents impliqués dans une clause apprise donnée pour quantifier la qualité des clauses apprises. Ainsi, les clauses apprises avec les plus petites valeurs de *LBD* sont considérées comme les plus pertinentes. Dans [4], une nouvelle police de gestion dynamique de la base des clauses apprises est proposée. Elle est fondée sur un principe d'activation et de gel dynamique des clauses apprises. A un état de recherche donné, en utilisant une fonction de sélection fondée sur une sauvegarde progréssive des polarités des littéraux appelée *PSM (Progress Saving Measure)*, elle active les clauses apprises les plus prometteuses tout en gélant celles ju-

gées non pertinentes. L'idée est alors de geler certaines clauses au lieu de les supprimer définitivement afin de les réactiver plus tard. La valeur *PSM* d'une clause est le nombre de littéraux ayant la même polarité que la *phase saving* (dernières polarités choisies pendant la recherche par l'heuristique de polarité) apparaissant dans cette clause.

Dans [18], un nouveau critère pour quantifier la pertinence d'une clause apprise en utilisant son niveau de retour-arrière appelé *BTL* pour `BackTrack Level` est proposé. A partir des expérimentations, les auteurs observent que les clauses apprises avec les petites valeurs de *BTL* sont plus souvent utilisées dans le processus de propagation unitaire que celles ayant des grandes valeurs *BTL*. Plus précisément, les auteurs observent que les clauses apprises ayant les valeurs *BTL* inférieures à 3 sont toujours plus utilisées dans la propagation unitaire que le reste des clauses. A partir de cette observation, et motivés par le fait qu'une clause apprise avec une petite valeur *BTL* contient plus de littéraux du sommet de l'arbre de recherche, les auteurs déduisent que les clauses apprises les plus pertinentes sont celles permettant un retour-arrière plus haut dans l'arbre de recherche (clauses ayant de petites valeurs *BTL*). Plus récemment, plusieurs autres stratégies de gestion de la base des clauses apprises furent proposées dans [22, 3]. Dans [22], les auteurs explorent un nombre varié de stratégies de réduction de la base des clauses apprises, et les performances des différentes extensions du solveur *Minisat* intégrant leurs stratégies sont evaluées sur les instances SAT prises des compétitions SAT 2013 et 2014, et comparées aux autres solveurs SAT de la littérature (*Glucose, Lingeling*) ainsi qu'au solveur original *Minisat*.

A partir des performances obtenues avec leurs stratégies, les auteurs de [22] ont montré que les stratégies d'apprentissage fondées sur la mise en place d'une borne supérieure sur la taille des clauses proposées il y a plus d'une quinzaine d'années restent également de bonnes mesures pour prédire la qualité des clauses apprises. Ils montrent que l'ajout de la randomisation à l'apprentissage fondée sur la mise en place d'une borne supérieure sur la taille des clauses est un bon moyen de parvenir à une diversification contrôlée, permet de favoriser les clauses de petites tailles tout en maintenant une petite fraction de clauses de grande taille nécessaires pour la dérivation des preuves par résolution sur certaines instances difficiles.

Dans [3], les auteurs utilisent la structure de communauté des instances SAT industrielles pour identifier un ensemble de clauses apprises fortement utiles. Ils montrent que l'augmentation d'une instance SAT avec les clauses apprises par le solveur pendant son exécution n'est pas toujours un moyen de rendre facile la résolution de l'instance. Les auteurs montrent cependant qu'en ajoutant à la formule originale un ensemble de clauses fondé sur la structure de communauté de la formule améliore les performances du solveur dans beaucoup de cas. Les différentes performances obtenues par chaque stratégie suggèrent que la question de prédire efficacemment les meilleures clauses apprises à maintenir pour la suite de la recherche est toujours ouverte et mérite de fortes investigations.

D'autre part, il est important de noter que l'efficacité de la plupart de ces stratégies de gestion des clauses apprises de létat de l'art depend fortement de la fréquence de nettoyage et de la quantité de clauses à supprimer à chaque nettoyage. Généralement, tous les solveurs SAT CDCL utilisant ces stratégies suppriment exactement la moitié des clauses apprises à chaque étape de nettoyage de la base des clauses apprises. Par exemple, les solveurs SAT CDCL *Minisat* [13] et *Glucose* [5] suppriment la moitié des clauses apprises à chaque étape de nettoyage. Cependant, l'efficacité de cette quantité de clauses apprises à supprimer (c'est à dire la moitié) à chaque étape de nettoyage de la base de clauses apprises n'a pas été démontrée théoriquement, mais plutôt expérimentalement. A notre connaissance, il n'existe pas beaucoup d'études dans la littérature sur comment déterminer la quantité de clauses à supprimer à chaque nettoyage. Ce papier propose une approche pour identifier les clauses apprises les plus pertinentes au cours du processus de résolution sans favoriser auncune des mesures de qualité de la littérature, de plus, cette approche s'affranchit du problème de la determination de la quantité de clauses à supprimer à chaque nettoyage de la base des clauses apprises : la quantité de clauses apprises à supprimer correspond à chaque fois au nombre de clauses apprises dominées par au moins l'une des top-k clauses apprises (ensemble des clauses apprises de référence courantes). Les top-k clauses apprises seront appelées dans les sections suivantes clauses apprises de référence.

4 Détection des clauses apprises non dominées

Nous présentons maintenant notre mesure de pertinence des clauses apprises fondée sur la relation de dominance. Nous motivons tout d'abord cette approche à partir d'un simple exemple, et proposons ensuite un algorithme permettant d'identifier les clauses apprises les plus pertinentes avec quelques détails techniques.

4.1 Motivation

Considérons les stratégies de pertinence suivantes : *LBD* [5], *SIZE* (stratégie qui considère comme pertinentes les clauses de petites tailles) et la mesure de

pertinence utilisée par *minisat* [13] que nous appelons ici *CVSIDS*. Supposons que nous ayons dans la base des clauses apprises, les clauses c_1, c_2 et c_3 avec :

— $SIZE(c_1) = 8$, $LBD(c_1) = 3$, $CVSIDS(c_1) = 1e^{100}$;
— $SIZE(c_2) = 6$, $LBD(c_2) = 5$, $CVSIDS(c_2) = 1e^{200}$;
— $SIZE(c_3) = 5$, $LBD(c_3) = 4$, $CVSIDS(c_3) = 1e^{300}$.

La question que nous posons est la suivante : laquelle des clauses est la plus pertinente ? Dans [5], les auteurs considèrent la clause c_1 qui a le plus petit *LBD* comme la plus pertinente. Par contre, les auteurs de [22] et [15] préfèrent la clause c_3 tandis que la préférence des auteurs de *Minisat* [13] va à la clause c_3. Notre approche s'affranchit de cette préférence particulière à une mesure en cherchant un compromis entre les différentes mesures de pertinence à travers une relation de dominance. Ainsi, pour la situation décrite plus haut, seule la clause c_2 est non pertinente, elle est dominée par la clause c_3 sur les trois mesures données.

4.2 Formalisation

Au cours de la recherche, les solveurs SAT CDCL apprennent un ensemble de clauses qui sont stockées dans une base des clauses apprises Δ, $\Delta = \{c_1, c_2, ..., c_n\}$. A chaque étape de réduction de la base des clauses apprises, nous évaluons ces clauses sur un ensemble de mesures de pertinence $\mathcal{M} = \{m_1, m_2, ..., m_k\}$. Nous notons $m(c)$ la valeur de la mesure m pour la clause c, $c \in \Delta$, $m \in \mathcal{M}$.

Comme l'évaluation des clauses apprises varie d'une mesure à une autre, l'utilisation de plusieurs mesures pourrait conduire à différents résultats (clauses pertinentes par rapport à une mesure de pertinence). Par exemple, si nous considérons l'exemple ci-dessus, c_1 est la meilleure clause par rapport à la mesure *LBD*, ce qui n'est pas le cas par rapport à la mesure d'évaluation $SIZE$ qui favorise c_3. Cette différence d'évaluation rend difficile le choix d'une mesure lors du processus de selection des clauses apprises. Ainsi, nous pouvons utiliser la notion de dominance entre clauses apprises afin de selectionner les clauses les plus pertinentes. Avant la formulation de la relation de dominance entre clauses apprises, il est nécessaire de la définir au niveau des valeurs des mesures de pertinence. Pour ce faire, nous définissons la `valeur de dominance` comme suit :

Definition 1 (Valeur de dominance) *Etant donnée une mesure de pertinence de clauses apprises m et deux clauses apprises c et c', nous disons que $m(c)$ domine $m(c')$, et notons $m(c) \succeq m(c')$, si et seulement si $m(c)$ est préférée à $m(c')$. Si $m(c) \succeq m(c')$ et $m(c) \neq m(c')$ alors nous disons que $m(c)$ domine strictement $m(c)'$ et notons $m(c) \succ m(c')$.*

Definition 2 (Dominance entre clauses) *Etant données deux clauses apprises c et c', la relation de dominance entre c et c' par rapport à un ensemble de mesures de pertinence $\mathcal{M}$ est définie comme suit :*

— *c domine c', noté $c \succeq c'$, si et seulement si $m(c) \succeq m(c')$, $\forall m \in \mathcal{M}$.*
— *si c domine c' et $\exists m \in \mathcal{M}$ tel que $m(c) \succ m(c')$, alors c domine strictement c' et nous notons $c \succ c'$.*

Pour découvrir les clauses apprises pertinentes, une approche naïve consiste à comparer chaque clause apprise avec toutes les autres clauses apprises. Cependant, le nombre de clauses apprises est prouvé être exponentiel, ce qui rend les comparaisons par paire très coûteuses. Dans la suite, nous montrons comment rsoudre ce problème en définissant à chaque étape de réduction de la base de clauses apprises, un ensemble de clauses apprises particulier (*top-k* clauses apprises) que nous appelons dans ce papier *clauses apprises de référence (en abrégé RLC)* qui est un ensemble de clauses apprises non dominées de Δ par rapport à l'ensemble des mesures de pertinence de clauses apprises $\mathcal{M}$

A chaque étape de nettoyage de la base des clauses apprises, toutes les clauses apprises dominées par au moins l'une des k *clauses apprises de référence (top-k clauses apprises)* seront considérées comme étant ds clauses apprises non pertinentes et ainsi supprimées de la base des clauses apprises.

Pour définir les *clauses apprises de référence*, nous avons besoin de définir une nouvelle mesure de qualité de clauses fondée sur toutes les mesures de qualité des clauses apprises de $\mathcal{M}$. Nous appelons cette nouvelle mesure *"Dégré de compromis"*, en court *DegComp*. Elle est définie comme suit :

Definition 3 (Degré de compromis) *Etant donnée une clause apprise c, le degré de compromis de c par rapport à l'ensemble des mesures de pertinence de clauses apprises $\mathcal{M}$ est défini par $DegComp(c) = \frac{\sum_{i=1}^{n} \widehat{m_i(c)}}{|M|}$, où $\widehat{m_i(c)}$ correspond à la valeur normalisée de la mesure m_i pour la clause c.*

Dans le même ordre d'idée, et afin d'éviter de conserver un grand nombre de clauses apprises dans la base de clauses apprises à chaque étape de réduction, nous proposons une stratégie de dominance qui considère k *Clauses Apprises de Référence (en court k-RLC)* au lieu d'une seule *clause clause apprise de référence* appelée ici 1-*RLC*. Plus précisément, la stratégie *k-RLC* est définie comme suit : à chaque étape de réduction, nous supprimons de la base des clauses apprises toutes les clauses dominées par au moins l'une des k prémières clauses apprises non dominées (*top-k* clauses apprises) par rapport à l'ensemble de mesures de pertinence de clauses appises.

En effet, en pratique, les mesures sont hétérogènes et sont définies à différentes échelles. Par exemple, les valeurs de la mesure de qualité des clauses apprises dans [13] sont des très grandes valeurs, alors que les valeurs de la mesure de pertinence dans [5] sont des petites valeurs. Ainsi, afin d'éviter que des mesures avec des grandes valeurs rendent marginales les mesures avec de petites valeurs dans le calcul du dégré de compromis d'une clause apprise donnée, il est nécessaire de normaliser les valeurs des différentes mesures de pertinence. Dans notre cas, nous choisissons de normaliser toutes les mesures dans l'intervalle $[0, 1]$.

Plus précisément, chaque valeur de mesure $m(c)$ de toute clause apprise c doit être normalisée dans $[0, 1]$. La normalisation d'une mesure donnée m est effectuée en fonction du domaine de ces valeurs et de la distribution statistique de son domaine actif. Nous rappelons que le domaine actif d'une mesure m est l'ensemble de ses valeurs possibles. Il convient de mentionner que *la normalisation d'une mesure ne modifie pas la relation de dominance* entre deux clauses données. Considérons la clause apprise c_1 donnée dans l'exemple de motivation à la section 4.1, avec ses valeurs sur les trois mesures considérées : $DegComp(c_1) = \frac{\widetilde{CVSIDS}(c_1)+\widetilde{LBD}(c_1)+\widetilde{SIZE}(c_1)}{3}$, then, we have, $DegComp(c_1) = \frac{\frac{1}{1e^{100}}+\frac{3}{nVars()}+\frac{8}{nVars()}}{3}$, avec $nVars()$ le nombre de variables de la formule booléenne.

Il est important de préciser que, le domaine des valeurs de toutes les mesures de pertinence devrait être conforme, c'est-à-dire, soient les plus petites valeurs sont meilleures ou soient les plus grandes valeurs sont meilleures. Par exemple, si nous considérons la clause apprise c_1, et les 3 mesures de pertinence LBD, $SIZE$ et $CVSIDS$, nous prenons la valeur $\frac{1}{CVSIDS(c_1)}$ comme valeur normalisée de la clause c_1 sur la mesure $CVSDIS$. En effet, les plus grandes valeurs $CVSIDS$ sont meilleures alors que les plus petites valeurs LBD et $SIZE$ sont meilleures. De cette manière, les mesures deviennent comparables sur les clauses.

Apès avoir donné les définitions néecessaires (*clause apprise de référence et dégré de compromis*), le lemme suivant offre une solution plus rapide que des comparaisons de clauses par paires, pour rechercher les clauses apprises pertinentes fondées sur la relation de dominance.

Lemme 1 *Soit c une clause apprise ayant le dégré de compromis minimalpar rapport à l'ensemble des mesures de pertinence des clauses apprises $\mathcal{M}$, alors c est une clause apprise non dominée.*

Dans notre stratgie k-RLC, les k prémières clauses apprises non dominées sont les k prémières clauses apprises classées dans l'ordre croissant de leur dégré de compromis. Il est important de souligner qu'à chaque étape de réduction, les k prémières clauses apprises non dominées ne sont pas toujours les k prémières clauses apprises classées dans l'ordre croissant de leur dégré de compromis.

Propriété 1 *Soit $\mathcal{M}$ l'ensemble des mesures de pertinence des clauses apprises, $\forall c, c', c"$ trois clauses apprises, si $c \succ c'$ et $c' \succ c"$ alors $c \succ c"$.*

Rechercher toutes les clauses apprises non dominées à chaque étape de réduction peut être tr'es couteux en temps, de ce fait, nous cherchons uniquement les clauses apprises non dominées par rapport aux k clause apprises de référence (*top-k* clauses apprises) à chaque étape de réduction de la base des clauses apprises.

Au cours de la recherche, à chaque étape de nettoyage de la base des clauses apprises, nous cherchons d'abord la clause apprise $cMin$ ayant le dégré de compromis minimal par rapport à l'ensemble des mesures de pertinence consodérées $\mathcal{M}$. Nous supprimons ensuite de la base des clauses apprises toutes les clauses dominées par $cMin$.

Au cours de la recherche, à chaque étape de nettoyage de la base des clauses apprises, nous comparons toutes les autres clauses apprises avec chacune des *top-k* clauses apprises. Ainsi, pour chaque clause apprise, nous éffectuons au moins une comparaison et au plus k comparaisons avec les *top-k* clauses apprises pour savoir si la clause est dominée et ensuite supprimée ou si la clause n'est pas dominée et donc conservée.

Propriété 2 *Soient $\Delta = \{c_1, c_2, ..., c_n\}$, une base de clauses apprises, k le nombre de clauses apprises de réfrence (top-k clauses apprises), la complexité de notre approche de dominance k-RLC est linéaire dans le pire des cas.*

4.3 Algorithme

Dans cette section, après avoir présenté le schéma général d'une stratégie de suppression des clauses apprises ($reduceDB(\Delta)$) adoptée par la plupart des solveurs SAT de la littérature, nous proposons un algoritme permettant de découvrir les clauses apprises pertinentes en utilisant la relation de dominance.

L'algorithme 1 présente le schéma général d'une stratégie de suppression des clauses apprises ($reduceDB(\Delta)$). Cet algorithme trie dans un premier temps l'ensemble des clauses apprises sur le critère défini et supprime ensuite la moitié des clauses apprises. En effet, cet algorithme prend en entrée une base de clauses apprises de taille n et retourne en sortie une base de clauses apprises de taille $\frac{n}{2}$. Ceci est

différent de notre approche qui trie dans un premier temps l'ensemble des clauses apprises par rapport à leur dégré de compromis et supprime ensuite toutes les clauses qui sont dominées par au moins l'une des k prémières clauses apprises non dominées (*top-k* clauses apprises).

L'algorithme 2 présente notre stratégie de suppression des clauses apprises. Il faut noter que les clauses apprises dont la taille (en nombre de littéraux) ou le LBD est plus petit ou égal à 2 ne sont pas concernées par la relation de dominance. Ces clauses apprises sont considérées comme pertinentes et sont maintenues dans la base des clauses apprises.

Algorithm 1: Stratégie de suppression : fonction `reduceDB`

Données: Δ : La base des clauses apprises de taille n
Résultat: Δ : La nouvelle base des clauses apprises de taille $n/2$

```
1  sortLearntClauses() ;      /* sur le critère défini */
2  limit = n/2;
3  ind = 0;
4  tant que ind < limit faire
5      clause = Δ[ind] ;
6      si clause.size() > 2 && clause.lbd() > 2 alors
7          removeClause() ;
8      sinon
9          saveClause() ;
10     ind++;
11 retourner Δ ;
```

Proposition 1 *Les k clauses apprises de référence considérées par l'algorithme 2 correspondent aux k prémières clauses apprises non dominées classées dans l'ordre croissant de leur dégré de compromis.*

Propriété 3 *Soient $\Delta = \{c_1, c_2, ..., c_n\}$ une base de clauses apprises et k le nombre de clauses apprises de référence considérées. Soit u le nombre de clauses apprises non dominées après avoir appliqué l'algorithme 2. Nous avons $k \leq u \leq |\Delta|$.*

5 Expérimentationss

Pour nos expérimentations, nous utilisons trois mesures de pertinence pour la relation de dominance afin d'évaluer l'efficacité de notre approche. Il faut noter que l'utilisateur peut choisir de combiner différentes autres mesures de pertinence. Nous utilisons pour notre étude, les mesures SIZE [15], LBD [5] et

Algorithm 2: `reduceDB-Dominance_Relationship`

Données: Δ : La base des clauses apprises ; $\mathcal{M}$: un ensemble de mesure de pertinence de clauses apprises ; k : le nombre de clauses apprises de réffrence
Résultat: Δ : La nouvelle base des clauses apprises

```
1  sortLearntClauses() ;      /* par rapport au dégré de compromis */
2  ind = 1;
3  j = 1;
4  undoC = 1 ;      /* le nombre de clauses non dominées courant */
5  tant que ind < |Δ| faire
6      c = Δ[ind] ;        /* une clause apprise */
7      si c.size() > 2 && c.lbd() > 2 alors
8          cpt = 0 ;
9          tant que cpt < undoC &&
           ¬dominates(Δ[cpt], Δ[ind], M) faire
10             cpt++ ;
11         si cpt >= undoC alors
12             saveClause() ;
13             j++ ;
14             undoC = min(k, j) ;    /* minimum entre
               j et k */
15         sinon
16             removeClause() ;
17     sinon
18         saveClause() ;
19         j++ ;
20         undoC = min(k, j) ;    /* minimum entre j
           et k */
21     ind++;
22 retourner Δ ;

23 Fonction dominates(cMin : une clause, c : une clause, M)
24 i = 0;
25 tant que i < |M| faire
26     m = M[i] ; /* une mesure de pertinence */
27     si m(c) ⪰ m(cMin) alors
28         retourner FAUX ;
29     i++;
30 retourner VRAI ;
```

CVSIDS [13]. L'efficacité de toutes ces mesures a été prouvée dans la littérature [13, 5, 22].

Nous exécutons les différents solveurs SAT sur les 300 instances SAT prises de la dernière SAT-RACE

2015 et sur les 300 instances SAT prises de la dernière compétition SAT 2016. Toutes les instances sont prétraitées par `SatElite` [12] avant l'exécution du solveur SAT. Les expérimentations sont ménées sur une machine Quad-Core Intel XEON avec 32GB de mémoire fonctionnant à 2.66 Ghz. Pour chaque instance, nous utilisons un temps limite égal à 3600 secondes du temps CPU pour les instances de la SAT-RACE, et 10000 secondes pour celles de la compétition SAT 2016. Nous implémentons notre approche dans le solveur SAT *Glucose* et faisons une comparaison entre le solveur original et celui amélioré avec la nouvelle stratégie de suppression des clauses apprises qui utilise la relation de dominance et que nous appelons ici *k−RLC−Glucose*, avec k le nombre de *clauses apprises de rfŕence*. Pour déterminer la meilleure valeur de k, nous exécutons *k-RLC-Glucose* avec $k = 1$, 3, 5 et 6.

5.1 Nombre d'instances résolues et temps CPU

Le tableau 1 présente les résultats obtenus sur les instances de la SAT-RACE 2015. Nous utilisons le code source de *Glucose* 3.0 avec la mesure *LBD* (appelé *LBD-Glucose* ou *Glucose* dans la suite). Nous remplaçons ensuite la mesure *LBD* par chacune des autres mesures de pertinence considérées ici, à savoir : *SIZE-Glucose* qui considère les clauses apprises de petites tailles comme les plus pertinentes, *CVSIDS-Glucose* qui maintient les clauses apprises les plus impliquées dans les récentes analyses de conflits, *RAND-Glucose* qui supprime de manière aléatoire les clauses apprises, et finalement notre approche *k-RLC-Glucose* qui supprime de la base des clauses apprises toutes les clauses dominées par les k prémières clauses apprises non dominées classées dans l'ordre croissant de leur dégré de compromis. Le tableau 1 montre une évaluation expérimentale comparative des quatre mesures de pertinence, ainsi que du solveur *Minisat* 2.2.

Dans la deuxième colonne du tableau 1, nous donnons le nombre total d'instances résolues (#Solved). Nous mentionnons aussi, le nombre d'instances prouvées satisfiables (#SAT) et le nombre d'instances prouvées insatisfiables (#UNSAT) entre parenthèses. La troisième colonne indique le temps CPU moyen en secondes (le temps total sur les instances résolues divisé par le nombre d'instances résolues). Sur la SAT-RACE 2015, notre approche *k-RLC-Glucose* est plus efficace que toutes les autres approches en terme de nombre d'instances résolues (voir aussi la figure 1).

En effet, le solveur original *Glucose* résout 236 instances, alors que le solveur amélioré avec notre approche de dominance *k-RLC-Glucose* résout 10 (respectivement 12) instances de plus pour $k = 3$ (respectivement $k = 6$). En effet, résoudre un tel nombre d'instances en plus est clairement significatif pour la résolution pratique de SAT. Le solveur *CVSIDS-Glucose* résout 4 instances de plus que *Glucose* 3.0. *Minisat* 2.2 résout seulement 209 instances.

Comme la stratégie de suppression aléatoire de clause est quelque fois efficace pour résoudre certaines instances SAT [22], nous quantifions le gap de performance entre notre stratégie de suppression et celle qui supprime aléatoirement les clauses appelée *RAND-Glucose*. Le solveur *RAND-Glucose* est obtenu comme suit : à chaque conflit, l'activité de la clause apprise c est mise à la valeur aléatoire $irand(random\_seed, |V_{\mathcal{F}}|)$, où $irand(random\_seed, |V_{\mathcal{F}}|)$ retourne un nombre entre 0 et $|V_{\mathcal{F}}|$ et $|V_{\mathcal{F}}|$ dénote l'ensemble de variables apparaissant dans la formule booléenne $\mathcal{F}$. Nous utilisons exactement la fonction aléatoire du solveur *Glucose* avec le même *random_seed* pour permettre la réproduction des résultats réportés dans ce papier. Nous pouvons voir comme réporté dans la table 1 que *RAND-Glucose* est le plus pire des solveurs, il résout seulement 174 instances, 72 (respectivement 74) instances de moins que notre solveur *k-RLC-Glucose* pour $k = 3$ (respectivement $k = 6$). Ceci montre l'interêt de notre approche de dominance sur les instances de la SAT-RACE 2015.

Solveurs	#Solved (#SAT - #UNSAT)	Temps Moyen
Minisat 2.2	209 (134 - 75)	585.19 s
RAND-Glucose	174 (99 - 75)	608.82 s
SIZE-Glucose	230 (131 - 99)	533.86 s
CVSIDS-Glucose	240 (140 - 100)	622.23 s
LBD-Glucose	236(136 - 100)	**481.66 s**
1-*RLC-Glucose*	238 (138 - 100)	**481.72 s**
3-*RLC-Glucose*	**246 (144 - 102)**	523.46 s
5-*RLC-Glucose*	245 (144 - 101)	542.29 s
6-*RLC-Glucose*	**248 (145 - 103)**	532.05 s

TABLE 1 – Evaluation comparative des solveurs sur la SAT-RACE-2015.

La figure 1 montre les résultats des temps cumulés i.e le nombre d'instances (axe-x) résolues en un temps donné en secondes (axe-y). Cette figure donne pour chaque technique, le nombre d'instances (*#instances*) en t secondes. Elle confirme l'efficacité de notre approche fondée sur la relation de dominance. Nous observons sur cette figure que le solveur *k-RLC-Glucose* (pour les valeurs de $k = 3, 6$) est généralement plus rapide que tous les autres solveurs, même si le temps de résolution moyen du solveur *LBD-Glucose* est légèrement meilleur (voir Tableau 1). Bien que 3-*RLC-Glucose* ait besoin d'un temps supplémentaire pour appliquer la relation de dominance, la qualité des clauses apprises restantes (sur la SAT-RACE) contribue à améliorer le temps nécessaire pour résoudre les instances.

Le tableau 2 montre 6 instances parmi les instances de la SAT-RACE 2015 résolues par notre meilleure version 6-*RLC-Glucose* en moins de 2200*secondes* et

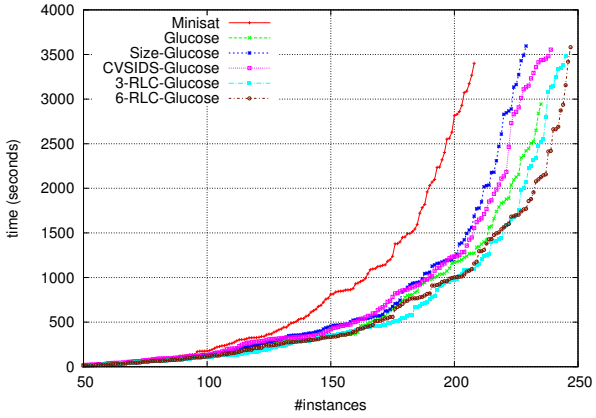


FIGURE 1 – **Evaluation des solveurs sur la SAT-RACE-2015**

non résolues par *LBD-Glucose*, *SIZE-Glucose*, ni par *CVSIDS-Glucose* en 3600*secondes*. Le temps utilisé pour résoudre ces instances peut expliquer en partie l'augmentation du temps moyen de résolution de 6-*RLC-Glucose*. En plus, nous constatons également qu'il n'y a aucune instance résolue par tous les autres solveurs et non résolue par 6-*RLC-Glucose* (comme détaillé plus loin). Cela montre d'une part que l'application de la domination entre différentes mesures de pertinence ne dégrade pas les performances de ces mesures, mais tire profit de la performance de chacune des mesures, en considérant l'ensemble d'instances de la SAT-RACE 2015.

Instances	LBD	SIZE	CVSIDS	6-RLC
jgiraldezlevy.2200.9086.08.40.8	-	-	-	**80.57 s**
jgiraldezlevy.2200.9086.08.40.149	-	-	-	**393.51 s**
jgiraldezlevy.2200.9086.08.40.2	-	-	-	**1599.86 s**
manthey_DimacsSorterHalf_37_3	-	-	-	**1763.42 s**
manthey_DimacsSorter_37_3	-	-	-	**1770.69 s**
hwmcc10-*-pdtvisns3p00-tseitin	-	-	-	**2146.03 s**

TABLE 2 – Instances résolues par 6-*RLC-Glucose* et non résolues par les autres solveurs sur la SAT-RACE 2015.

Le tableau 3 présente les résultats sur les instances de la compétition SAT 2016. Ici, *LBD-Glucose* et *CVSIDS-Glucose* résolvent le même nombre d'instances. Notre approche *k*-*RLC-Glucose* reste compétitive et résout plus d'instances que le solveur original *Glucose* pour $k = 3, 6$. La figure 2 présente les résultats des temps cumulés sur les instances de la compétition SAT 2016. Il ressort de ce deuxième jeu de données que 3-*RLC-Glucose* est légèrement plus rapide que *LBD-Glucose*. *LBD-Glucose* est plus efficace que les autres solveurs. Une analyse plus fine de la figure 2 montre que 3-*RLC-Glucose* est généralement plus rapide sur les instances résolues entre 3000*secondes* and 5000*secondes*. Ceci peut être expliqué par le fait que quelque fois au cours d'une étape de résolution, très peu de clauses apprises sont dominées par les $top - k$ clauses apprises, et notre solveur conserve un grand nombre de clauses apprises dans la base.

Ce résultat donne du crédit au théorème du "No Free Lunch" [35]. Nous pensons également que la fonction d'aggrégation ne peut pas être unique pour tous les jeux de données, de ce fait qu'il est nécessaire d'explorer la combinaison efficace des mesures préférées.

RAND-Glucose résout seulement 121 instances, 48 instances de moins que 3-*RLC-Glucose*.

Solvers	#Solved (#SAT - #UNSAT)	Temps Moyen
Minisat 2.2	138 (65 - 73)	1194.85 s
RAND-Glucose	121 (56 - 65)	1120.99 s
SIZE-Glucose	156 (67 - 89)	1396.73 s
CVSIDS-Glucose	165 (67 - **98**)	1368.99 s
LBD-Glucose	165 (68 - 97)	**1142.33 s**
1-*RLC-Glucose*	156 (64 - 92)	1227.62 s
3-*RLC-Glucose*	**169 (71 - 98)**	1297.36 s
5-*RLC-Glucose*	165 (68 - 97)	1352.46 s
6-*RLC-Glucose*	167 (70 - 97)	1439.21 s

TABLE 3 – Evaluation comparative des solveurs sur la Compétition SAT 2016.

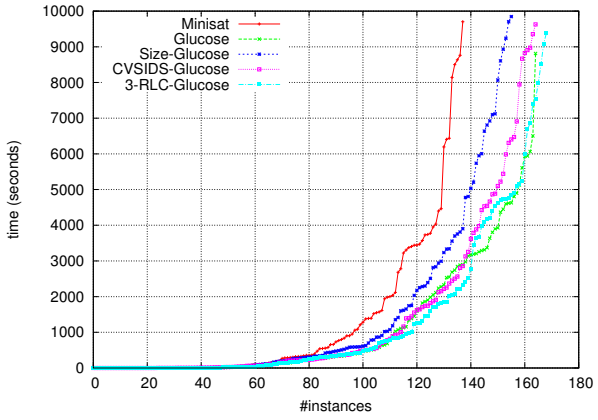


FIGURE 2 – **Evaluation des solveurs sur la compétition SAT 2016**

Le tableau 4 montre6 instances de la compétition SAT 2016 résolues par notre approche avec la meilleure version 3-*RLC-Glucose* parmi lesquelles 5 instances sont résolues en moins de 5000 secondes mais aucune instance n'est résolue par les solveurs *LBD-Glucose*, *SIZE-Glucose*, ni par *CVSIDS-Glucose*. Ceci confirme comme sur les instance de la SAT-RACE 2015 que notre approche de dominance ne dégrade par les performances des solveurs mais prend plutôt avantage de toutes les performances de chaque mesure de pertinence.

Dans les expériementations, nous nous focalisons sur la meilleure version du solveur 6-*RLC-Glucose* (respectivement 3-*RLC-Glucose*) pour les instances de la SAT-RACE 2015 (respectivement SAT competition

Instances	LBD	SIZE	CVSIDS	3-RLC
barman-pfile08-032.sas.ex.15	-	-	-	**5.01 s**
partial-10-19-s	-	-	-	**1248.47 s**
ak128modasbg2asisc	-	-	-	**1021.06 s**
par32-3-c	-	-	-	**2206.07 s**
gss-24-s100	-	-	-	**4753.82 s**
eq.atree.braun.13.unsat	-	-	-	**6683.69 s**

TABLE 4 – Instances résolues par 3-*RLC-Glucose* et non résolues par les autres solveurs sur la SAT compétition 2016.

2016). En effet, les deux solveurs (6-*RLC-Glucose*, 3-*RLC-Glucose*) sont les plus efficaces et ont approximativement les même performances sur les deux jeux de données (SAT-RACE 2015, SAT-Competition 2016), même si le solveur 6-*RLC-Glucose* (respectivement 3-*RLC-Glucose*) semble lgèrement plus compétitif que 3-*RLC-Glucose* (respectivement 6-*RLC-Glucose*) sur les instances de la SAT-RACE 2015 (respectivement SAT compétition 2016).

5.2 Instances communes résolues

Dans le tableau 5, l'intersection entre deux mesures de pertinece donne le nombre d'instances communes résolues par chaque mesure. Par exemple, LBD et $SIZE$ résolvent 219 instances en commun, tandis que 235 instances sont résolues par LBD et 6-RLC. Nous pouvons voir que notre approche résout en commun le plus grand nombre d'instances avec chacune des mesures aggrégées. Plus précisément, pour chacune des mesures, le nombre d'instances résolues en commun avec une autre mesure est plus petit que le nombre d'instances résolues en commun avec notre approche.

Measures	LBD	SIZE	CVSIDS	6-RLC
LBD	236			**235**
SIZE	219	230		**224**
CVSIDS	233	221	240	**234**
6-RLC				**248**

TABLE 5 – Instances résolues communément sur la SAT-RACE 2015.

Le tableau 6 donne l'intersection des résultats entre deux mesures de pertinence sur les instances de la SAT compétition 2016. Pour une mesure de pertinence donnée, nous pouvons voir que le nombre d'instances communes résolues avec une autre mesure est plus petit ou égal au nombre d'instances communes résolues avec notre approche k-*RLC-Glucose* pour $k = 3$.

Pour plus de détails, le tableau 7 donne le nombre d'instances communes résolues par les mesures de pertinence sur les instances de la SAT-RACE-2015. Ce tableau permet de voir le nombre d'instances communes résolues par 1, 2, 3 ou 4 mesures. Par exemple, les 4 stratégies utilisées dans l'expérimentation de notre approche résolvent en commun 218 instances, tandis que

Measures	LBD	SIZE	CVSIDS	3-RLC
LBD	165			**161**
SIZE	153	156		**153**
CVSIDS	161	151	165	**161**
3-RLC				**169**

TABLE 6 – Instances résolues communément sur la SAT compétition 2016.

43 instances ne sont résolues par aucune des stratégies. Nous pouvons observer aussi que 0, 3, 4, et 6 sont respectivement les nombres d'instances résolues uniquement respectivement par LBD, $CVSIDS$, $SIZE$ et 6-RLC. De plus, il n'existe aucune instance résolue par les trois stratégies (LBD, $SIZE$ et $CVSIDS$) et non résolue par notre approche 6-RLC.

Measures		6-RLC		¬6-RLC	
		CVSIDS	¬CVSIDS	CVSIDS	¬CVSIDS
LBD	SIZE	218	1	**0**	0
	¬SIZE	14	2	1	0
¬ LBD	SIZE	1	4	2	4
	¬SIZE	1	**6**	3	43

TABLE 7 – Détails des instances communes résolues sur la SAT-RACE 2015.

Le tableau 8 donne plus de détails sur le nombre d'instances communes résolues par les mesures de pertinence sur la compétition SAT 2016. De ce tableau, nous observons que les 4 stratgies de suppression de clauses apprises résolvent en commun 150 instances et 123 instances ne sont résolues par aucune des stratégies. Il est important de noter sur ces instances de la compétition SAT 2016 qu'aucune instance n'est résolue par les trois stratégies (LBD, $SIZE$ and $CVSIDS$) et non résolue par notre approche 3-RLC alors que 6 instances sont résolues uniquement par 3-RLC.

Measures		3-RLC		¬3-RLC	
		CVSIDS	¬CVSIDS	CVSIDS	¬CVSIDS
LBD	SIZE	150	1	**0**	2
	¬SIZE	10	0	1	**1**
¬ LBD	SIZE	1	1	0	**1**
	¬SIZE	0	**6**	**3**	123

TABLE 8 – Détails des instances communes résolues sur la compétition SAT 2016.

6 Conclusion et Perspectives

Dans ce papier, nous proposons une approche qui traite le problème de la gestion de la base des clauses apprises. Nous avons montré que l'idée de relation de dominance entre mesures de qualité des clauses apprises est un bon moyen pour tirer profit de chaque mesure. A chaque étape de réduction de la base des clauses apprises, nous sélectionnons judicieusement les

top-k clauses apprises (les k prémières clauses apprises non dominées courant) par rélativement à un ensemble de mesures de pertinence de clauses de clauses, et supprimons toutes les clauses dominées par au moins l'une des top-k clauses apprises non dominées. Cette approche n'est pas altérée par l'abondance des mesures de pertinence qui ont fait l'objet de plusieurs travaux. L'approche proposée évite un autre problème non trivial qui est celui de déterminer la quantité de clauses apprises à supprimer à chaque étape de réduction de la base des clauses apprises en déterminant dynamiquement le nombre de clauses à supprimer à chaque étape de réduction. Un algorithme général pour notre approche est proposé et évalué sur les instances de la SAT-RACE 2015 et de la compétition SAT 2016. Les résultats expérimentaux montrent que l'exploitation de la relation de dominance améliore les performances des solveurs SAT CDCL sur ces instances. Les performances sont plus significatives sur les instances de la SAT-RACE 2015. Pour le cas de la compétition SAT 2016 où les performances sont légèrement diminuées, nous devons explorer les effets des autres composants clés des solveurs SAT CDCL sur notre approche de dominance. Les catégories d'instances sont également un problème qui devrait être exploré.

A notre connaissance, c'est la première fois que la relation de dominance a été utilisée dans le domaine de la satisfiabilité pour améliorer les performances d'un solveur SAT CDCL. Notre approche ouvre des perspectives intéressantes. En effet, toute nouvelle mesure de pertinence de clauses apprises peut être intégrée dans la relation de dominance. Cependant, il faudrait noter que l'ajout d'une nouvelle mesure de pertinence à la relation de dominance ne contribue pas toujours à améliorer les performances du solveur. Il est important d'avoir une diversité entre les différents mesures de pertinence utilisées dans la relation de dominance. Par exemple, l'utilisation de la mesure `BTL` [18] avec les mesures `LBD, SIZE, CVSIDS` dans la relation de dominance n'améliore pas les performances du solveur `Glucose` sur les instances de la SAT-RACE 2015 et sur les instances de la compétition SAT 2016. Une perspective immédiate serait de proposer une direction pour ajuster dynamiquement le nombre de top-k clauses apprises à considérer à chaque étape de réduction en fonction de l'état de la recherche.

Remerciements

Nous remercions les auteurs du solveur SAT *Glucose* pour la disponibilité du code source de leur solveur. Nous souhaitons également remercier la région "Auvergne-Rhône-Alpes" et l'Union européenne pour leur soutien financier par le biais du Fonds européen de développement régional (FEDER) dans le cadre du projet "MobiPaleo" ainsi que la région "Hauts-de-France" pour son soutien financier à travers le projet CPER Data.

Références

[1] C. Ansótegui, M. L. Bonet, J. Giráldez-Cru, and J. Levy. Structure features for SAT instances classification. *J. Applied Logic*, 23 :27–39, 2017.

[2] C. Ansótegui, J. Giráldez-Cru, and J. Levy. The community structure of SAT formulas. In *SAT, Trento, Italy*, pages 410–423, 2012.

[3] C. Ansótegui, J.s Giráldez-Cru, J. Levy, and L. Simon. Using community structure to detect relevant learnt clauses. In *SAT*, pages 238–254, 2015.

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

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

[6] P. Beame, H. A. Kautz, and A. Sabharwal. Towards understanding and harnessing the potential of clause learning. *JAIR*, 22 :319–351, 2004.

[7] A. Biere. Preprocessing and inprocessing techniques in SAT. In *Hardware and Software : Verification and Testing, HVC, Haifa, Israel, Revised Selected Papers*, page 1, 2011.

[8] A. Biere. Lingeling and friends entering the sat challenge 2012. In *Proceedings of SAT Challenge 2012 : Solver and Benchmark Descriptions*, pages 33–34, 2012.

[9] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE, Heidelberg, Germany*, pages 421–430, 2001.

[10] S. Bouker, R. Saidi, S. Ben Yahia, and E. Mephu Nguifo. Mining undominated association rules through interestingness measures. *IJAIT*, 23(04) :1460011, 2014.

[11] M. Davis, G. Logemann, and D. W. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7) :394–397, 1962.

[12] N. Eén and A. Biere. Effective preprocessing in sat through variable and clause elimination. In *SAT*, pages 61–75, 2005.

[13] N. Eén and N. Sörensson. An extensible sat-solver. In *SAT*, pages 502–518, 2003.

[14] J. Giráldez-Cru and J. Levy. A modularity-based random SAT instances generator. In *IJCAI, Buenos Aires, Argentina*, pages 1952–1958, 2015.

[15] E. Goldberg and Y. Novikov. Berkmin : A fast and robust sat-solver. *Discrete Applied Mathematics*, 155(12) :1549 – 1561, 2007.

[16] C. P. Gomes and B. Selman. Algorithm portfolios. *AIJ*, 126(1-2) :43–62, 2001.

[17] C. P. Gomes, B. Selman, and H. A. Kautz. Boosting combinatorial search through randomization. In *AAAI/IAAI*, pages 431–437, 1998.

[18] L. Guo, S. Jabbour, J. Lonlac, and L. Sais. Diversification by clauses deletion strategies in portfolio parallel SAT solving. In *ICTAI*, pages 701–708, 2014.

[19] F. Hutter, M. Lindauer, A. Balint, S. Bayless, H. H. Hoos, and K. Leyton-Brown. The configurable SAT solver challenge (CSSC). *AIJ*, 243 :1–25, 2017.

[20] F. Hutter, Lin Xu, H. H. Hoos, and K. Leyton-Brown. Algorithm runtime prediction : Methods & evaluation. *AIJ*, 206 :79–111, 2014.

[21] S. Jabbour, J. Lonlac, and L. Sais. Extending resolution by dynamic substitution of boolean functions. In *ICTAI, Athens, Greece*, pages 1029–1034, 2012.

[22] S. Jabbour, J. Lonlac, L. Sais, and Y. Salhi. Revisiting the learned clauses database reduction strategies. *CoRR*, abs/1402.1956, 2014.

[23] H. Katebi, K. A. Sakallah, and J. P. Marques Silva. Empirical study of the anatomy of modern sat solvers. In *SAT*, pages 343–356, 2011.

[24] K. Leyton-Brown, E. Nudelman, G. Andrew, J. McFadden, and Y. Shoham. A portfolio approach to algorithm selection. In *IJCAI-03, Acapulco, Mexico*, page 1542, 2003.

[25] J. Lonlac and E. Mephu Nguifo. Towards learned clauses database reduction strategies based on dominance relationship. *CoRR*, abs/1705.10898, 2017.

[26] M. Luo, Chu-Min Li, F. Xiao, F. Manyà, and Z. Lü. An effective learnt clause minimization approach for CDCL SAT solvers. In *IJCAI, Melbourne, Australia*, pages 703–711, 2017.

[27] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff : Engineering an efficient sat solver. In *DAC*, pages 530–535, 2001.

[28] Z. Newsham, V. Ganesh, S. Fischmeister, G. Audemard, and L. Simon. Impact of community structure on SAT solver performance. In *SAT, Vienna, Austria*, pages 252–268, 2014.

[29] E. Nudelman, K. Leyton-Brown, H. H. Hoos, A. Devkar, and Y. Shoham. Understanding random SAT : beyond the clauses-to-variables ratio. In *CP, Toronto, Canada*, pages 438–452, 2004.

[30] F. P. Preparata and M. I. Shamos. *Computational Geometry : An Introduction*. Springer-Verlag, Berlin, 1985.

[31] J. P. Marques Silva and K. A. Sakallah. GRASP : A search algorithm for propositional satisfiability. *IEEE Trans. Computers*, 48(5) :506–521, 1999.

[32] N. Sörensson and A. Biere. Minimizing learned clauses. In *SAT, Swansea, UK, June 30 - July 3*, pages 237–243, 2009.

[33] A. Soulet, C. Raïssi, M. Plantevit, and B. Crémilleux. Mining dominant patterns in the sky. In *ICDM*, pages 655–664, 2011.

[34] M. van Leeuwen and A. Ukkonen. Discovering skylines of subgroup sets. In *ECML PKDD, Prague, Czech Republic*, pages 272–287, 2013.

[35] D. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Trans. Evolutionary Computation*, 1(1) :67–82, 1997.

[36] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Satzilla : Portfolio-based algorithm selection for SAT. *JAIR*, 32 :565–606, 2008.

[37] L. Zhang, C. F. Madigan, M. W. Moskewicz, and S. Malik. Efficient conflict driven learning in boolean satisfiability solver. In *ICCAD, San Jose, CA, USA*, pages 279–285, 2001.

Echantillonnage sous contraintes en viticulture de précision

Baptiste OGER[1,2] * **Bruno TISSEYRE**[2] **Philippe VISMARA**[1,3]

[1] MISTEA, Montpellier SupAgro, INRA, Université Montpellier, France
[2] ITAP, Montpellier SupAgro, IRSTEA, Université Montpellier, France
[3] LIRMM, Montpellier SupAgro, Université Montpellier, CNRS, France
{baptiste.oger, bruno.tisseyre, philippe.vismara}@supagro.fr

Résumé

Le développement de l'agriculture de précision nécessite de collecter des données par échantillonnage de manière plus rationnelle. Cet échantillonnage s'appuie sur des outils statistiques qui n'intègrent pas toujours les contraintes opérationnelles. L'objet de cette étude est d'utiliser le raisonnement par contraintes pour optimiser l'échantillonnage dans une parcelle de vigne. Il s'agit à la fois de choisir, dans la parcelle, les points qui respectent les contraintes d'échantillonnage tout en optimisant le déplacement du technicien. C'est donc un problème de tournée, pour lequel la Programmation par Contrainte a développé de nombreux algorithmes de filtrage ces dernières années, notamment la contrainte WeightedSubCircuit [Briot et al. 2017] bien adaptée aux tournées ne passant que par un sous-ensemble de points. Nous proposons ici un modèle pour une version simplifiée du problème et les premiers résultats obtenus avec le solveur Choco.

Abstract

The development of precision agriculture pushes growers to collect sampling data more efficiently. Sampling methods rely on statistical tools that do not always consider operational constraints. The purpose of this study is to use Constraint Programming to optimize sampling in vineyard block. It is both about choosing sampling points that meet operational constraints in the vineyard and optimizing the path taken by the practionner. It is thus a routing problem for which Constraint Programing developed many filtering algorithms in the last few years, in particular the constraint WeightedSubCircuit [Briot et al. 2017], well suited for routes connecting a small subset of points. We expose in this paper a first model for a simplified approach of the problem as well as the first results that we get from the Choco solver.

*Papier doctorant : Baptiste OGER[1,2] est auteur principal.

1 Introduction

Malgré l'apparition de nouvelles méthodes d'acquisition de données, de nombreuses mesures continuent d'être effectuées par échantillonnage manuel dans les parcelles agricoles. Initialement effectué de façon aléatoire, cet échantillonnage peut aussi s'appuyer sur des outils statistiques et intégrer différentes contraintes opérationnelles, en particulier associées à l'environnement structuré des rangs de vigne. Afin de mettre au point une méthodologie d'échantillonnage en viticulture de précision, cet article s'intéresse à la conception d'un outil d'optimisation sous contraintes capable de déterminer les points à échantillonner tout en optimisant le trajet du technicien. Ce type de problème se prête bien à la programmation par contraintes, d'autant plus qu'il s'agit de pouvoir faire évoluer le modèle afin d'y incorporer différents critères.

En viticulture, l'estimation du rendement est importante pour faciliter l'organisation des vendanges. Des études récentes ont mis en évidence une corrélation entre le rendement et le NDVI (Normalized Difference Vegetation Index) qui fournit un indice de couverture végétale en chaque point de la parcelle [4]. Le NDVI est ainsi une donnée auxiliaire, c'est à dire une donnée reliée à la variable que l'on cherche à estimer. Elle est accessible à moindre coût par télédétection avec un niveau de précision en chaque point qui dépend de la résolution de l'image. En travaillant sur des parcelles pour lesquelles cette donnée auxiliaire est disponible, on souhaite exploiter cette corrélation pour orienter le choix des points d'échantillonnage.

Une contrainte majeure de l'échantillonnage est le temps disponible par parcelle. Celui-ci est généralement assez faible et le temps passé pour les déplace-

ments du technicien dans les rangs de vigne se fait au détriment du nombre de mesures. Ce n'est pas sans conséquences sur la qualité de l'estimation finale. L'optimisation du parcours emprunté par l'opérateur est donc une part importante du problème. Cette optimisation s'apparente à un problème de tournée de véhicule (VRP) [11].

Depuis plusieurs années, la Programmation par Contraintes permet de résoudre efficacement des problèmes faisant intervenir des contraintes de circuit, notamment quand elles sont combinées à d'autres types de contraintes. C'est le cas pour des problèmes de voyageur de commerce (TSP) ou des problèmes de tournée de véhicule (VRP) [9, 5, 7, 1, 6]. Ces contraintes ont été appliquées à de nombreux domaines y compris en viticulture de précision [2].

La particularité du problème d'échantillonnage est qu'il mêle problème de tournée et sélection des points de passage parmi un ensemble plus large de sites candidats. La contrainte WSC (WeightedSubCircuits) [3] permet justement de gérer ce genre de situation où les points de passage sont soumis à d'autres contraintes.

La suite de l'article présente une formalisation du problème d'échantillonnage en viticulture de précision et une première modélisation sous la forme d'un problème d'optimisation sous contraintes. Nous présentons ensuite quelques résultats préliminaires obtenus avec le solveur Choco sur des données obtenues sur l'unité expérimentale de Pech Rougre (INRA).

2 Le problème de l'échantillonnage en viticulture de précision

L'estimation précoce du rendement est une donnée fondamentale pour l'organisation logistique des vendanges. Cette opération, réalisée quelques jours en amont des vendanges permet d'optimiser les opérations de récolte (gestion des équipements, de la main d'œuvre etc.) et la qualité du produit. Dans ce but un opérateur est envoyé sur la parcelle pour faire des mesures de comptages sur plusieurs sites d'échantillonnage. Ces quelques mesures de rendement ponctuelles permettent ensuite d'obtenir une estimation globale du rendement de la parcelle.

Le nombre de sites qui seront échantillonnés au cours du parcours est supposé fixé à une valeur N donnée. L'objectif sera d'effectuer les N mesures le plus rapidement possible mais on pourra faire varier N pour déterminer le nombre d'échantillonnages réalisables dans un temps donné.

Afin de pouvoir optimiser le chemin parcouru par l'opérateur lors de son travail d'échantillonnage, il est nécessaire de calculer les distances effectives entre deux points de la parcelle. Les rangs des parcelles de vignes étant généralement palissées, il n'est pas possible de couper à travers la parcelle. La distance entre deux sites d'un même rang sera donc la distance euclidienne convertie en temps de parcours. Dans le cas où les deux sites ne sont pas dans le même rang le chemin qui les sépare passe par le bord de la parcelle. Cette matrice de distances (en temps) sera notée $\mathcal{M}$ par la suite.

Les points d'échantillonnage sont choisis parmi un ensemble de sites candidats. Il s'agit d'un sous ensemble de pieds de vigne qui peuvent être répartis sur une grille régulière ou obtenus par d'autres méthodes. Par exemple, la figure 1 montre une parcelle expérimentale où 53 sites candidats ont été échantillonnés afin de pouvoir tester différentes méthodes d'échantillonnage sur un sous-ensemble de ces points.

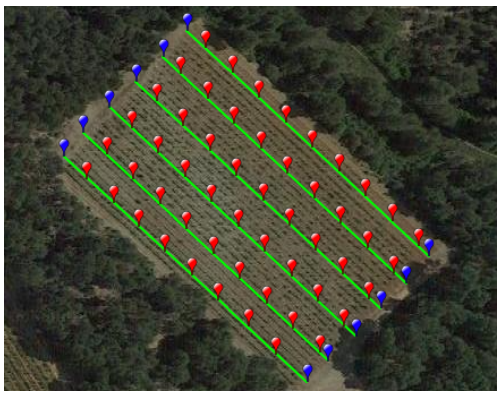


FIGURE 1 – Parcelle avec les sites d'échantillonnage candidats (rouge) et les extrémités de rangs (bleue).

L'intérêt d'utiliser une variable auxiliaire comme le NDVI pour l'échantillonnage est double : il s'agit d'une part d'aider à sélectionner les points d'échantillonnage les plus pertinents et, d'autre part, d'établir la corrélation entre les données mesurées et les données auxiliaires. Cette corrélation permettra ensuite d'obtenir une estimation de la variable d'intérêt (le rendement) pour l'ensemble des points pour lesquels on connait la valeur de la variable auxiliaire.

Pour choisir les points suivant cette approche on peut s'inspirer de la méthode de Kennard & Stone [8]. Cette méthode commence par la sélection des deux points les plus distants dans l'espace des variables auxiliaires. Dans le cas étudié ici, avec une seule variable auxiliaire (le NDVI), cet espace n'aura donc qu'une seule dimension et les points choisis correspondront aux valeurs minimales et maximales du NDVI. Les points suivants sont ensuite choisis de manière itérative en sélectionnant à chaque fois le point dont le NDVI sera le plus éloigné des valeurs déjà sélectionnées, toujours dans l'espace des variables auxiliaires. À chaque étape, un seul point d'échantillonnage est sélectionné. Dans la section suivante, nous proposerons une

variante de cette approche afin de proposer plusieurs points candidats à chaque étape, ce qui permettra de les sélectionner d'après les autres contraintes.

3 Modélisation par un COP

Le problème de l'échantillonnage pour l'estimation du rendement peut être modélisé comme un problème d'optimisation sous contraintes (COP). Il s'agit d'un modèle préliminaire qui doit permettre de tester différentes variantes du problème d'échantillonnage.

3.1 Variables et domaines

On suppose que les K sites d'échantillonnage candidats sont numérotés de 0 à $K-1$. La position initiale du technicien est associée au site K.

Un ensemble de $N+1$ variables $\{P_i\}_{i\in 0..N}$ désignera les N sites qui seront échantillonnées. La variable $P_N = K$ correspond au point de départ du technicien. Chacun des autres P_i désigne un point d'échantillonage ayant un certain niveau de NDVI (imposé par la contrainte 2) mais sans lien avec l'ordre de parcours. $D(P_i) = \{0, \ldots, K-1\}$ pour $i < N$.

Pour décrire le trajet du technicien, on définit $K+1$ variables telles que $\forall j \in 0 \ldots K,\ D(Next_j) = \{0, \ldots, K\}$. La variable $Next_j$ désigne le point suivant j dans la tournée du technicien.

Si le technicien ne visite pas le point j alors la contrainte 4 (WSC) imposera $Next_j = j$.

La variable $Cost$ est la variable à optimiser. Elle correspond au coût (en temps) du trajet du technicien.

Pour faciliter l'écriture du modèle, on introduit une variable ensembliste $Visited \subseteq \{0, \ldots, K\}$ contenant l'ensemble des points échantillonnés.

Une contrainte de channeling (Union dans le solveur Choco) permet de la lier aux variables P_i :

$$Visited = \{j \in 0 \ldots K \mid \exists i \in 0..N, P_i = j\} \qquad (1)$$

3.2 Prise en compte de la variable auxiliaire

L'objectif est de choisir les P_i pour que les valeurs auxiliaires (NDVI) associées à ces points soient réparties au mieux dans l'espace des variables auxiliaires.

Considérons le cas simple à une dimension (NDVI). On peut déterminer l'ensemble des points choisis par la méthode de Kennard & Stone [8]. Comme le montre la figure 2, il s'agit d'un ensemble de points qui subdivisent successivement l'espace des NDVI en deux. On définit pour cela une fonction ϕ_i telle que :

$\phi_0 = \text{NDVI}_{min}$ $\qquad$ $\phi_1 = \text{NDVI}_{max}$

$\forall i \in 2, \ldots, N-1 \phi_i = \frac{1+2\times(i-1-2^\lambda)}{2^{\lambda+1}} \times (\phi_1 - \phi_0) + \phi_0$

avec $\lambda = floor(\frac{log(i-1)}{log(2)})$

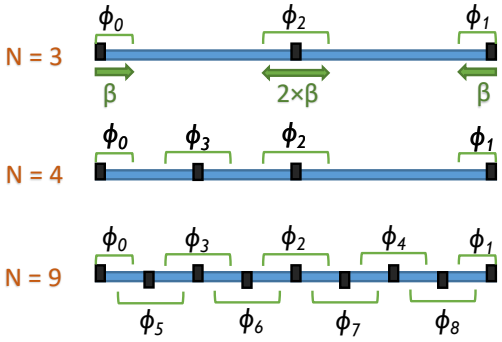


FIGURE 2 – Répartition des valeurs dans l'espace de la variable auxiliaire, suivant le nombre N d'échantillons.

Dans notre modèle, chaque site d'échantillonnage P_i sera choisi parmi l'ensemble des sites candidats ayant un NDVI proche à plus ou moins une constante β près de la valeur ϕ_i. Il s'agit donc d'une contrainte unaire sur les variables P_i :

$$|NDVI_{P_i} - \phi_i| < \beta \qquad (2)$$

Par ailleurs, suivant la valeur de β, les domaines des $P_i, i \in \{0, \ldots, N-1\}$, peuvent se recouper. On pose donc une contrainte de différence :

$$AllDifferent(P_0, \ldots, P_N) \qquad (3)$$

3.3 Contrainte de tournée

L'optimisation du trajet du technicien est gérée par la contrainte WeightedSubCircuit (WSC) de [3] qui est utilisée ici avec un seul circuit [1] :

$$WSC[\mathcal{M}](Visited, \{Next_j\}_{j\in 0..K}, Cost) \qquad (4)$$

Cette contrainte garantit que l'ensemble des $Next_i \neq i$ définissent un circuit Hamiltonien de coût $Cost$ (d'après les temps de parcours $\mathcal{M}$) sur l'ensemble des points contenus dans $Visited$.

4 Résultats

Le modèle décrit précédemment a été implémenté avec le solveur Choco [10] dans le langage Java. Il a été testé sur des données provenant d'une parcelle du vignoble de l'INRA Pech Rouge dans l'Aude (figure 1). Les calculs ont été effectués sur une machine Linux avec Intel(R) Xeon(R) CPU E5-2680 2.40GHz. La table 1 donne le résultat des expérimentations. La première colonne donne le nombre de site d'échantillonnage à instancier, la deuxième colonne le paramètre β qui influe sur la taille des domaines, la taille

1. la variable Z de la contrainte initiale est ici confondue avec $Cost$. De plus, l'ensemble Set_{dummy} des sommets non visités n'apparait pas puisqu'il est égal au complémentaire de $Visited$

N	β	$\lvert D(P_i)\rvert$	Temps CPU	Nœuds
5	5	3.80	0.2	452
5	10	7.80	1	6 531
5	15	13.00	5	31 537
6	10	7.17	2	23 444
6	15	11.83	19	153 056
7	10	7.14	8	59 763
7	15	11.29	111	640 643
8	10	9.50	100	1 119 260
8	15	13.37	1277	9 623 221
9	10	9.56	552	3 211 244
9	15	13.33	9 569	49 990 919
10	10	8.80	441	6 214 723
10	15	12.50	13 614	102 793 574

TABLE 1 – Résultats préliminaires

moyenne de ces derniers pour les N sites est donnée dans la troisième colonne, les dernières colonnes donnent respectivement le temps de calcul (sec) et le nombre de nœuds explorés.

On remarque que le solveur arrive à trouver la solution optimale dans un temps assez court pour des instances allant jusqu'à $N = 7$, ce qui correspond à un nombre d'échantillonnage courant. Au delà, le temps de calcul augmente avec la taille de l'instance mais reste exploitable pour tester des modèles, ce qui est le seul objectif de l'implémentation actuelle.

Il s'agit ici d'un modèle fonctionnel mais simplifié et incomplet du problème d'échantillonnage. Il est destiné à être complété par de nouvelles contraintes afin d'améliorer la prise en compte des contraintes opérationnelles (déplacement, choix des sites d'échantillonnages...), et d'affiner la qualité de l'estimation qui résulte de l'échantillonnage (prise en compte de l'autocorrélation spatiale des données, intégration de nouvelles approches statistiques ...). Dans un second temps, quand le problème sera bien spécifié, plusieurs pistes devront être explorées pour diminuer les temps de calcul (contraintes de symétrie, contraintes redondantes, ordre d'instanciation, ...).

5 Conclusion

Dans cet article, nous avons abordé le problème de l'échantillonnage sous contraintes. Nous avons proposé un premier modèle permettant de choisir les points d'échantillonnage en s'appuyant sur une variable auxiliaire tout en optimisant le trajet du technicien. Ce modèle utilise la contrainte WeightedSubCircuit [3] et les variables et contraintes ensemblistes du solveur Choco. L'implémentation de ce modèle, sans être très performante, va nous permettre d'explorer différentes variantes d'échantillonnages en viticulture de précision.

Ce travail a bénéficié d'une aide de l'État gérée par l'Agence Nationale de la Recherche au titre du programme d'Investissements d'Avenir portant la référence ANR-16-CONV-0004

Références

[1] P. Benchimol, W. Jan van Hoeve, J.-C. Régin, L.-M. Rousseau, and M. Rueher. Improved filtering for weighted circuit constraints. *Constraints*, 17(3) :205–233, 2012.

[2] N. Briot, C. Bessiere, and P. Vismara. A constraint-based approach to the differential harvest problem. In *Proc. 21st International Conference on Principles and Practice of Constraint Programming (CP 2015)*, volume 9255 of *LNCS*, pages 541–556. Springer, 2015.

[3] N. Briot, C. Bessiere, and P. Vismara. Une contrainte de circuit adaptée aux tournées multiples. In *Actes des Treizièmes Journées Francophones de Programmation par Contraintes (JFPC 2017)*, pages 137–144, 2017.

[4] E. Carrillo, A. Matese, J. Rousseau, and B. Tisseyre. Use of multispectral airborne imagery to improve yield sampling in viticulture. *Precision Agriculture*, 17(1) :74–92, 2015.

[5] Y. Caseau and F. Laburthe. Solving small tsps with constraints. In *Logic Programming, Proceedings of the Fourteenth International Conference on Logic Programming*, pages 316–330, 1997.

[6] S. Ducomman, H. Cambazard, and B. Penz. Alternative filtering for the weighted circuit constraint : Comparing lower bounds for the TSP and solving TSPTW. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 3390–3396, 2016.

[7] F. Focacci, A. Lodi, and M. Milano. Embedding relaxations in global constraints for solving TSP and TSPTW. *Annals of Mathematics and Artificial Intelligence*, 34(4) :291–311, 2002.

[8] W. Kennard and L. A. Stone. Computer Aided Design of Experiments. *Technometrics*, 11 :137–148, 1969.

[9] G. Pesant, M. Gendreau, J.-Y. Potvin, and J.-M. Rousseau. An exact constraint logic programming algorithm for the traveling salesman problem with time windows. *Transportation Science*, 32(1) :12–29, 1998.

[10] C. Prud'homme, J.-G. Fages, and X. Lorca. *Choco Documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2016.

[11] P. Toth and D Vigo. *Vehicle routing : problems, methods, and applications*. SIAM, 2014.

VNS itératif guidé par la décomposition arborescente pour la minimisation d'énergie dans les modèles graphiques

Résumé de l'article paru à la conférence Uncertainty in Artificial Intelligence (UAI-17), pages 550-559, Sydney, Australia, 2017

A. Ouali[1] **D. Allouche**[2] **S. de Givry**[2] **S. Loudni**[1] **Y. Lebbah**[3] **F. Eckhardt**[2] **L. Loukil**[3]

1 University of Caen Normandy, CNRS, UMR 6072 GREYC, 14032 Caen, France.
2 INRA, MIA Toulouse, UR-875, 31320 Castanet-Tolosan, France.
3 University of Oran 1 Ahmed Ben Bella, Lab. LITIO, 31000 Oran, Algeria.

Les modèles graphiques probabilistes (MGP) [4] unifient la théorie des probabilités et les modèles graphiques via des variables aléatoires reliées par une distribution de loi jointe donnant l'aspect probabiliste.

Formellement, un MGP [4] est un triplet $(\mathcal{X}, \mathcal{D}, \mathcal{F})$ avec $\mathcal{X} = \{X_1, \ldots, X_n\}$ un ensemble de n variables aléatoires discrètes, $\mathcal{D} = \{D_1, \ldots, D_n\}$ un ensemble de domaines finis de valeurs, et $\mathcal{F}$, un ensemble de *fonctions de probabilités conditionnelles* à valeurs réelles positives. Une affectation de l'ensemble $\mathcal{X}$ est le tuple $x = (x_1, \ldots, x_n)$, avec $x_i \in D_i$. L'ensemble de toutes les affectation possibles de $\mathcal{X}$ est noté $\Delta = \prod_{i=1}^{n} D_i$. Soit S un sous-ensemble de $V = \{1, \ldots, n\}$, X_S, x_S et Δ_S dénotent respectivement un sous-ensemble de variables aléatoires $\{X_i, i \in S\}$, l'affectation $(x_i, i \in S)$ obtenue à partir de x, et l'ensemble des affectations possibles de X_S. Soit $\mathcal{S}$ un sous-ensemble des parties de V, l'ensemble $\mathcal{F} = \{f_S\}_{S \in \mathcal{S}}$ définit une factorisation de la distribution de probabilité jointe de $\mathbb{P}$ ssi :

$$\mathbb{P}(x) = \frac{1}{Z} \prod_{f_S \in \mathcal{F}} f_S(x_S) \quad (1)$$

où $Z = \sum_{x \in \Delta} \prod_{f_S \in \mathcal{F}} f_S(x_S)$ est la constante de normalisation. Le problème *Most Probable Explanation* (MPE) consiste à trouver une affectation $x \in \Delta$ de toutes les variables de $\mathcal{X}$ de telle sorte que la probabilité jointe $\mathbb{P}(x)$ soit maximale. Il s'agit d'un problème NP-difficile [9]. Le problème MPE se traduit directement en un *réseau de fonctions de coût* [7] de minimisation de la somme de fonctions de coût, appelées aussi *fonctions d'énergie* [3].

Les méthodes de résolution sont qualifiées respectivement de méthodes complètes ou incomplètes selon leur capacité à prouver ou non l'optimalité de la solution trouvée. Elles font généralement appel soit à la recherche arborescente soit à la recherche locale. La combinaison de ces deux aspects a été étudiée dans peu de travaux. Parmi eux, les approches qui consistent à explorer les voisinages dans la recherche locale par une recherche arborescente de manière systématique ou non systématique. VNS/LDS+CP [5] combine une métaheuristique VNS [8] avec une recherche partielle de type LDS [2]. Récemment, Fontaine et al [1] ont proposé le premier cadre générique, appelé DGVNS, qui exploite une décomposition arborescente au sein de VNS. Dans ce papier, nous proposons `UDGVNS`, une variante itérative de DGVNS capable de prouver l'optimalité des solutions trouvées.

DGVNS itératif. L'algorithme 1 décrit le pseudo-code de `UDGVNS`. Il restaure l'exhaustivité de DGVNS en appliquant des appels successifs avec des valeurs croissantes de discrepancy (notée ℓ) pour LDS [1], en contrôlant si la recherche arborescente est partielle grâce au drapeau *opt* et le fait que le voisinage actuel garde certaines variables assignées dans l'affectation partielle A (test à la ligne 6). Dans `UDGVNS`, l'optimalité peut être prouvée dans deux cas : (i) lorsque le voisinage actuel correspond à l'ensemble des variables du problème et que la valeur de discrepancy est supérieure ou égale au nombre maximal de branches droites ou (ii) en examinant les bornes au nœud racine ($ub = \texttt{lb}(\mathcal{D})$, cf. lignes 2 et 5). Dans ce cas, l'optimalité est prouvée implicitement, l'espace de recherche n'est

1. Nous supposons un arbre de recherche binaire où à chaque nœud soit la variable sélectionnée est affectée à sa valeur préférée (branche gauche) soit la valeur est supprimée du domaine (branche droite). Chaque suppression correspond à une mauvaise décision prise par la recherche.

Algorithm 1: Unified DGVNS algorithm.

```
   Function UDGVNS(ℓ_min, ℓ_max, +_ℓ, k_min, k_max, +_k, ub :
     In/Out, x : In/Out) : boolean
       let (C_T, T) be a tree decomposition of (X, D, F) ;
       opt ← true ;
1      LDS^r(∞, D, ub, x, opt) ;        // initial solution
2      if (ub = lb(D)) then opt ← true;
       c ← 1 ;                          // current cluster index
       r ← 0 ;                          // number of iterations
       ℓ ← ℓ_min ;                      // initial discrepancy limit
       while (¬opt ∧ ℓ ≤ ℓ_max) do
           i ← 0 ;                      // nb. of failed neighbor.
3          k ← k_min ;                  // init. neighbo. size
           while (¬opt ∧ k ≤ k_max) do
               A ← getNeighborhood(x, C_c, k) ;
               ub' ← ub, opt ← true ;
4              LDS^r(ℓ, A, ub', x', opt) ;   // nei. search
5              if (ub' = lb(D)) then opt ← true;
6              else if (A ≠ D) then opt ← false;
7              if (ub' < ub) then
                   x ← x', ub ← ub' ;   // new best sol
8                  i ← 0, k ← k_min ;
9                  r ← 0, ℓ ← ℓ_min ;
               else
                   i ← i + 1 ;
                   if (k < k_max) then
                       k ← min(k_max, k_min +_k i) ;
                   else k ← ∞;
10             c ← 1 + c mod |C_T| ;   // get next clus.
           r ← r + 1 ;
           if (ℓ < ℓ_max) then
               ℓ ← min(ℓ_max, ℓ_min +_ℓ r) ;
           else ℓ ← ∞;
11     return opt ;
```

pas exploré. La solution initiale est obtenue à la ligne 1 par une version modifiée de LDS, notée $\texttt{LDS}^r$, qui stoppe après l'obtention de la première solution.

UDGVNS ajuste le compromis entre preuve d'optimalité et comportement anytime via deux paramètres : la limite de la discrepancy (ℓ) et la taille du voisinage (k). Dès qu'une meilleure solution est trouvée par $\texttt{LDS}^r$ dans le voisinage courant (ligne 4), nous arrêtons la recherche afin de réinitialiser les deux paramètres à leur valeur minimale, car il est plus rapide d'explorer de petites voisinages (lignes 8-9). Nous avons évalué trois stratégies itératives de mise à jour de ℓ et k pour l'opérateur $+_{\ell/k}$: augmente d'une unité ($+$), multiplie par deux (mult2), enfin selon une suite Luby [6][2] qui renvoie la valeur $a +_{\ell/k} b = \texttt{Luby}(a, b) = a \times luby(1 + b)$ pour $\forall a, b \in N^*$.

L'opérateur $+_k$ contrôle le compromis entre intensification et diversification. Le but de la stratégie de Luby est d'intensifier l'effort de recherche sur les petits voisinages en augmentant exponentiellement le nombre de voisinages de petite taille versus ceux de grande taille. Permettant ainsi de passer plus de temps sur les petits voisinages pour améliorer localement la solution actuelle, favorisant ainsi l'intensification. La stratégie mult2 réduit le nombre d'explorations de voisinage pour une valeur de discrepancy donnée, afin d'essayer plus rapidement des valeurs de ℓ plus grandes. Si le problème peut être résolu par une recherche complète dans les délais impartis, la preuve d'optimalité sera également accélérée.

L'opérateur $+_\ell$ contrôle l'équilibre entre recherche incomplète et complète. L'utilisation d'une stratégie de croissance rapide accentue l'exhaustivité alors qu'une croissance lente devrait favoriser les comportements anytime. Nous avons noté qu'il est plus efficace de couvrir toutes les variables par l'union des voisinages explorés afin de ne pas manquer certaines variables importantes. Nous avons testé une quatrième stratégie pour k qui consiste en un incrément lent (de $+1$) au début jusqu'à $k = \max_{i \in I}(|C_i|) + |C_T| - 1$ puis *saute* directement à $k = k_{max}$. Cela garantit que k augmente lentement jusqu'à ce que le plus grand groupe ait été totalement exploré par au moins une recherche de voisinage. Quand $k = k_{max} = |\mathcal{X}|$, UDGVNS effectue une nouvelle recherche de solution via $\texttt{LDS}^r$ sur l'ensemble du problème. S'il ne parvient pas à trouver une meilleure solution, une discrepancy plus grande est envisagée et UDGVNS poursuit son processus d'intensification en commençant par une petite taille de voisinage (ligne 3).

Expérimentations. UDGVNS a été évalué sur 3016 instances issues de compétitions sur les modèles graphiques. Elles recouvrent divers domaines d'applications (Probabilistic Inference Challenge, Computer Vision and Pattern Recognition, OpenGM2 benchmark, Cost Function Library). À notre connaissance, ce travail est la première tentative de restauration de la complétude sur une méthode de recherche locale (i.e. VNS). Les résultats montrent que l'approche offre un bon compromis entre comportement anytime et preuve d'optimalité. L'article à UAI-17 présente également une version parallèle d'UDGVNS.

2. Rappelons que $luby(i) = \{1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, \ldots\}$.

Références

[1] M Fontaine, S Loudni, and P Boizumault. Exploiting tree decomposition for guiding neighborhoods exploration for VNS. *RAIRO OR*, 47(2) :91–123, 2013.

[2] W Harvey and M Ginsberg. Limited discrepancy search. In *Proc. of IJCAI*, pages 607–615, 1995.

[3] B Hurley, B O'Sullivan, D Allouche, G Katsirelos, T Schiex, M Zytnicki, and S de Givry. Multi-Language Evaluation of Exact Solvers in Graphical Model Discrete Optimization. *Constraints*, 21(3) :413–434, 2016.

[4] D Koller and N Friedman. *Probabilistic graphical models : principles and techniques*. The MIT Press, 2009.

[5] S Loudni and P Boizumault. Solving constraint optimization problems in anytime contexts. In *Proc. of IJCAI*, pages 251–256, 2003.

[6] M Luby, A Sinclair, and D Zuckerman. Optimal speedup of Las Vegas algorithms. In *Proc. of TCS*, pages 128–133, 1993.

[7] P. Meseguer, F. Rossi, and T. Schiex. Soft constraints processing. In *Handbook of Constraint Programming*, chapter 9. Elsevier, 2006.

[8] N Mladenović and P Hansen. Variable Neighborhood Search. *Comput. Oper. Res.*, 24(11) :1097–1100, November 1997.

[9] S Shimony. Finding MAPs for belief networks is NP-hard. *Artificial Intelligence*, 68 :399–410, 1994.

Techniques de décisions hiérarchiques pour l'ordonnancement de tâches

Adriana Pacheco[*], **Cédric Pralet**, **Stéphanie Roussel**
ONERA / DTIS, Université de Toulouse, F-31055 Toulouse – France

prénom.nom@onera.fr

Résumé

Dans cet article, nous nous intéressons à des problèmes d'ordonnancement dans lesquels les tâches candidates sont hiérarchisées. Nous commençons par définir formellement un cadre de modélisation des problèmes d'ordonnancement hiérarchique. Nous proposons une première traduction de ce cadre vers la programmation par contraintes, ainsi qu'une méthode de décision basée sur des mécanismes d'abstractions et de décompositions des tâches du problème. Nous présentons différentes heuristiques d'abstraction et de décomposition associées à cette deuxième méthode. Ces deux méthodes ont été implémentées et nous montrons les résultats préliminaires obtenus sur des benchmarks représentatifs d'une application multi-robots.

Abstract

In this article, we consider scheduling problems in which candidate tasks are hierarchical. Fist of all, we formally define a framework for modelling hierarchical scheduling problems. We propose a first interpretation of this framework towards constraint programming, as well as a decision method based on abstraction and decomposition mechanisms of the problem tasks. We present several heuristics associated with the second method for the abstraction and the decomposition step. These two methods have been implemented and we present the obtained results on representative benchmarks of a multi-robot application.

1 Introduction

De nombreuses applications qui présentent un problème de décision ou d'optimisation combinatoire peuvent être formalisées à l'aide de modèles « tâches-ressources ». Ces derniers reposent sur un ensemble de tâches candidates et un ensemble de ressources disponibles pour les réaliser, et optimisent certains critères liés au temps ou aux ressources. C'est par exemple le cas des problèmes d'exploration multi-robots dans lesquels les tâches sont les différentes observations à réaliser et les déplacements associés à ces observations et les ressources sont les robots, les fenêtres ou canaux de communication qu'ils utilisent, le terrain sur lequel ils évoluent, l'énergie dont ils disposent, *etc.* (voir [1] pour une description d'un cas d'application). C'est également le cas des problèmes de planification des activités de satellites dans lesquels les tâches sont les observations et les vidages que le satellite doit réaliser et les ressources sont les instruments du satellite et les stations sol (vue d'ensemble de l'état de l'art présentée dans [4]). On peut finalement citer des problèmes d'allocation de fonctions sur des architectures avioniques où les tâches sont les fonctions avioniques et les ressources sont les calculateurs et le réseau de communication physique entre ces derniers ([7]).

En pratique, dans les problèmes listés ci-dessus, il est souvent utile de raisonner à différents niveaux d'abstraction. Dans le cas des applications de type exploration multi-robots, on décide par exemple à haut niveau de l'allocation des zones à explorer à chaque robot, puis à plus bas niveau de l'ordre dans lequel les observations associées à chaque zone sont réalisées et encore à plus bas niveau de la trajectoire fine de chacun des robots.

Pour prendre en compte ces niveaux d'abstraction, une première approche souvent utilisée en pratique consiste à décomposer explicitement le problème à résoudre en plusieurs sous-problèmes et à définir pour chacun de ces sous-problèmes une technique de résolution dédiée. Une seconde approche consiste à utiliser des solutions plus génériques, comme cela est fait

[*]Papier doctorant : Adriana Pacheco est auteur principal.

en planification avec le cadre des HTNs (Hierarchical Task Networks [6, 8, 2]), dans lequel le problème générique considéré est de décomposer des tâches de haut niveau en tâches dites atomiques, en utilisant un catalogue de méthodes de décomposition de tâches fourni en entrée.

Le cadre originel des HTNs n'est pas adapté pour modéliser les aspects « tâches-ressources ». En effet, la consommation des ressources par les tâches sont dans ce cas modélisées par des paramètres de l'état courant du système pouvant rendre complexe des raisonnements sur l'utilisation de ces ressources à différents niveaux de la hiérarchie de décision. Plusieurs travaux récents ([5, 10]) étendent le cadre HTN pour prendre en compte la notion de ressource. A l'inverse, peu de travaux cherchent à étendre le modèle classique « tâches-ressources » en y intégrant des aspects hiérarchiques.

Dans ce papier, nous suivons cette dernière approche et proposons un premier cadre de modélisation pour le problème d'ordonnancement hiérarchique défini par un ensemble de tâches pouvant être *atomiques* (i.e. pouvant être réalisées directement) ou *composite* (i.e. se décomposant en un sous-ensemble de tâches) et un ensemble de ressources disponibles pour les réaliser. Le problème d'ordonnancement hiérarchique s'intéresse au calcul de dates d'exécution optimales, en tenant compte des contraintes temporelles et des contraintes sur la disponibilité des ressources requises. Dans ce papier, nous nous intéressons plus particulièrement aux problèmes d'ordonnancement disjonctif, c'est-à-dire dans lesquels chaque ressource ne peut exécuter qu'une tâche à la fois.

Dans la section 2, nous définissons formellement le type de problèmes considérés ainsi qu'une première stratégie de résolution utilisant la programmation par contraintes. Pour réduire la combinatoire, nous introduisons des mécanismes d'abstraction des tâches composites du problème dans la section 3. Un algorithme de décomposition itérative des tâches du problème est ensuite proposé dans la section **??**. Les méthodes définies ont été implémentées et nous montrons les résultats obtenus sur des benchmarks représentatifs d'une application multi-robots dans la section 5. Nous concluons en section 6 sur les perspectives de ce travail et de la thèse associée.

2 Problèmes d'ordonnancement hiérarchique

Dans cette partie, nous définissons formellement la classe des problèmes d'ordonnancement hiérarchique auxquels nous nous intéressons. Des classes plus générales pourraient être considérées, mais nous traitons une classe relativement simple pour réaliser nos premières études.

Modèle général Un problème d'ordonnancement hiérarchique est un tuple $(\mathcal{R}, \mathcal{A}, \mathcal{C})$ dans lequel :

- $\mathcal{R}$ est un ensemble de *ressources disjonctives* (pouvant réaliser au plus une activité à la fois) ;
- $\mathcal{A}$ est un ensemble d'*activités* à réaliser, aussi appelées des *tâches atomiques* ; chaque activité $a \in \mathcal{A}$ est définie par une durée du_a et par l'ensemble des ressources $\mathcal{R}_a \subseteq \mathcal{R}$ qu'elle consomme pendant toute cette durée ;
- $\mathcal{C}$ est un ensemble de *tâches composites* ; chaque tâche composite $c \in \mathcal{C}$ est définie par :
  - un ensemble de sous-tâches $SubTsk_c \subset \mathcal{C} \cup \mathcal{A}$;
  - un ensemble de ressources $\mathcal{R}_c \subseteq \mathcal{R}$ consommées tant que la tâche composite est active (c'est-à-dire de la date de début de la première sous-tâche associée à c jusqu'à la date de fin de la dernière sous-tâche associée à c) ;
  - un ensemble de contraintes de précédence acycliques $\mathcal{P}_c$ entre les sous-tâches de c ($\mathcal{P}_c \subset SubTsk_c \times SubTsk_c$).

Dans ce qui suit, pour toute ressource $r \in \mathcal{R}$, on note $\mathcal{T}_r$ l'ensemble des tâches τ (atomiques ou composites) qui utilisent la ressource r, c'est-à-dire telles que $r \in \mathcal{R}_\tau$.

Hypothèses additionnelles Nous supposons que les problèmes d'ordonnancement hiérarchiques auxquels on s'intéresse sont *bien formés*. Plus précisément, on suppose que :

- le graphe de décomposition des tâches est acyclique ; formellement, ce graphe est le graphe orienté dont les nœuds sont les tâches de $\mathcal{A} \cup \mathcal{C}$, et qui contient pour chaque tâche composite c et chaque sous-tâche $\tau \in SubTsk_c$ un arc $c \to \tau$;
- chaque tâche atomique ou composite apparaît comme sous-tâche d'au plus une tâche ; cela implique que le graphe de décomposition des tâches définit une forêt de tâches ;
- une ressource déclarée comme étant consommée par une tâche composite c n'est pas déclarée comme consommée par une tâche τ descendante de c. Formellement, pour toute tâche composite c et toute tâche τ telles qu'il existe un chemin c vers τ dans le graphe de décomposition des tâches, on a $\mathcal{R}_c \cap \mathcal{R}_\tau = \emptyset$.

Commentaires sur le modèle Le modèle obtenu fait intervenir des décompositions hiérarchiques. Il permet de représenter facilement des problèmes de types *Job Shop Scheduling* ou *Open Shop Scheduling* [9], avec dans ce cas un seul niveau de décomposition (une tâche

composite par job). Dans le cas du *Job Shop*, la modélisation fait apparaître des contraintes de précédence définissant les séquences d'activités associées aux différents jobs. Dans le cas *Open Shop*, où l'ordre des tâches d'un job est laissé libre, la modélisation fait également apparaître une ressource factice supplémentaire pour interdire que des tâches d'un même job soient réalisées en parallèle.

De manière orthogonale, dans la mesure où chaque tâche peut consommer plusieurs ressources, le modèle couvre également les RCPSP (Resource Constrained Project Scheduling Problems [3]). Pour le moment nous ne considérons que des ressources de capacité unitaire, et l'extension aux ressources cumulatives est laissée pour des travaux futurs. Enfin, il est important de noter que les consommations de ressources peuvent être associées directement aux tâches composites, ce qui permet de modéliser des scénarios impliquant des ressources monopolisées pendant toute la durée d'un ensemble de sous-tâches, par exemple un scénario impliquant un opérateur devant réaliser tout seul un ensemble de travaux (tout seul pour éviter des "changements de contexte") mais devant attendre que des outillages partagés entre plusieurs opérateurs soient disponibles.

Exemple Nous nous intéressons à un problème d'ordonnancement hiérarchique basé sur une application d'exploration multi-robots, dans lequel des robots doivent se déplacer sur un terrain et réaliser un certain nombre d'observations de zones. Plus précisément, on attribue aux robots des zones à observer. Chaque observation de zone se décompose en un ensemble d'observations atomiques à réaliser en séquence, entre lesquelles le robot doit se déplacer. Ces déplacements peuvent eux-mêmes être décomposés en points de passage à rallier séquentiellement. On suppose que les robots ne peuvent pas observer plus d'une zone à la fois. Ils doivent de plus transmettre en temps réel chaque observation atomique au centre de mission, et pour cela ils utilisent une fréquence d'émission spécifique liée à la nature de l'observation réalisée. Finalement, les points de passage ne peuvent pas être occupés par plus d'un robot à chaque pas de temps.

Un exemple jouet associé à cette application est illustré sur la figure 1 :

- deux zones Z_0 et Z_1 doivent respectivement être explorées par deux robots r_0 et r_1 ;
- pour observer la zone Z_0, le robot r_0 doit d'abord réaliser l'observation O_0 et la retransmettre sur la fréquence f_0, puis réaliser O_1 sur f_1 ;
- le déplacement M_{01} se décompose en 3 étapes : M^0_{01} sur le point p_0, M^1_{01} sur le point p_1 et M^2_{01} sur le point p_2 ;
- un découpage similaire est réalisé pour la zone Z_1.

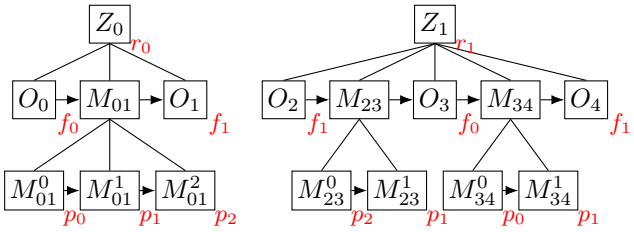


FIGURE 1 – Problème d'exploration multi-robots

Avec le cadre défini précédemment, cet exemple jouet se modélise comme suit :

- $\mathcal{R} = \{r_0, r_1, f_0, f_1, p_0, p_1, p_2\}$;
- $\mathcal{A} = \{O_0(2), O_1(3), O_2(3), O_3(1), O_4(2), M^0_{01}(1), M^1_{01}(2), M^2_{01}(1), M^0_{23}(2), M^1_{23}(1), M^0_{34}(2), M^1_{34}(1)\}$ (la durée de chaque activité est ici précisée entre parenthèses) ;
- $\mathcal{C} = \{Z_0, Z_1, M_{01}, M_{23}, M_{34}\}$;
- $SubTsk_{Z_0} = \{O_0, M_{01}, O_1\}$;
- $\mathcal{P}_{Z_0} = \{(O_0, M_{01}), (M_{01}, O_1)\}$;
- $SubTsk_{M_{01}} = \{M^0_{01}, M^1_{01}, M^2_{01}\}$;
- $\mathcal{P}_{M_{01}} = \{(M^0_{01}, M^1_{01}), (M^1_{01}, M^2_{01})\}$;
- $SubTsk_{Z_1} = \{O_2, M_{23}, O_3, M_{34}, O_4\}$;
- $\mathcal{P}_{Z_1} = \{(O_2, M_{23}), (M_{23}, O_3), (O_3, M_{34}), (M_{34}, O_4)\}$;
- $SubTsk_{M_{23}} = \{M^0_{23}, M^1_{23}\}$;
- $\mathcal{P}_{M_{23}} = \{(M^0_{23}, M^1_{23})\}$;
- $SubTsk_{M_{34}} = \{M^0_{34}, M^1_{34}\}$;
- $\mathcal{P}_{M_{34}} = \{(M^0_{34}, M^1_{34})\}$;
- $\mathcal{T}_{r_0} = \{Z_0\}$; $\mathcal{T}_{r_1} = \{Z_1\}$;
- $\mathcal{T}_{f_0} = \{O_0, O_3\}$; $\mathcal{T}_{f_1} = \{O_1, O_2, O_4\}$;
- $\mathcal{T}_{p_0} = \{M^0_{01}, M^0_{34}\}$;
- $\mathcal{T}_{p_1} = \{M^1_{01}, M^1_{23}, M^1_{34}\}$;
- $\mathcal{T}_{p_2} = \{M^2_{01}, M^0_{23}\}$.

L'exemple présenté fait intervenir des ressources utilisées pendant l'exécution des tâches d'intérêt (les observations utilisant des robots et des fréquences de communication) et un réseau partagé utilisé pour la mise au point de certaines ressources (les ressources robots qui doivent se déplacer dans l'environnement via des points de passage). La problématique serait la même pour des applications d'ordonnancement de fonctions sur une architecture embarquée, rencontrées par exemple en aéronautique, avec dans ce cas des ressources calculateurs utilisées pour exécuter les tâches d'intérêt (des calculs liées aux lois de commande, à l'estimation de la position d'un engin...) et un réseau embarqué constitué de liens réseaux et de nœuds réseaux à partager lors de la transmission de données entre certaines fonctions calculées.

Solution Une solution pour un problème d'ordonnancement hiérarchique associe une date de début s_τ et une date de fin e_τ à chaque tâche τ de $\mathcal{A} \cup \mathcal{C}$. Plusieurs contraintes doivent être satisfaites : pour chaque activité (ou tâche atomique), la date de fin de l'activité est donnée par la somme de sa date de début et de sa durée (équation 1) ; les dates de début et de fin d'une tâche composite c doivent couvrir exactement les sous-tâches de c (équations 2-3) ; deux tâches consommant la même ressource ne peuvent pas se chevaucher temporellement (équation 4) ; une précédence entre deux tâches implique que la date de fin de la première tâche doit être inférieure ou égale à la date de début de la seconde (équation 5).

$$\forall a \in \mathcal{A},\ e_a = s_a + du_a \quad (1)$$
$$\forall c \in \mathcal{C},\ s_c = \min\{s_\tau \mid \tau \in SubTsk_c\} \quad (2)$$
$$\forall c \in \mathcal{C},\ e_c = \max\{e_\tau \mid \tau \in SubTsk_c\} \quad (3)$$
$$\forall r \in \mathcal{R}, \forall \tau \neq \tau' \in \mathcal{T}_r^2,\ (e_\tau \leq s_{\tau'}) \vee (e_{\tau'} \leq s_\tau) \quad (4)$$
$$\forall c \in \mathcal{C},\ \forall (\tau, \tau') \in \mathcal{P}_c,\ s_{\tau'} \geq e_\tau \quad (5)$$

Une solution est dite optimale si elle minimise le *makespan*, défini comme la date de fin de la dernière tâche du plan ($\max\{e_\tau \mid \tau \in \mathcal{A} \cup \mathcal{C}\}$).

La figure 2 illustre la consommation des ressources associée à une solution de l'exemple jouet respectant les différentes contraintes du problème. Notons que sur cette figure, les tâches M_{01}, M_{23} et M_{34} n'apparaissent pas car elles ne consomment pas directement de ressources. Leurs dates de début et de fin sont : $s_{M_{01}} = 2$, $e_{M_{01}} = 6$, $s_{M_{23}} = 3$, $e_{M_{23}} = 6$, $s_{M_{34}} = 7$, $e_{M_{34}} = 10$.

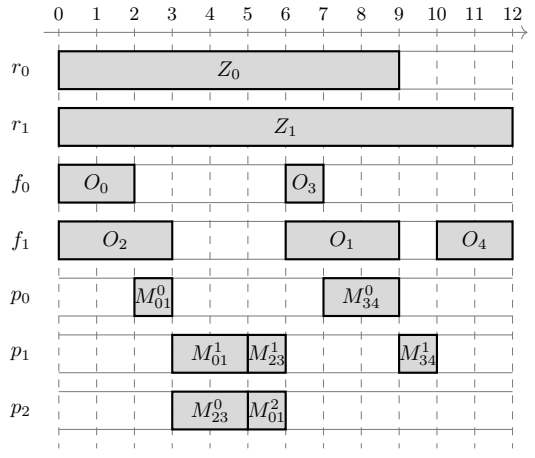


FIGURE 2 – Consommation des ressources pour une solution du problème jouet

Résolution en programmation par contraintes L'encodage en programmation par contraintes du problème d'ordonnancement hiérarchique défini à la section précédente est relativement direct. Il est par exemple possible de réutiliser les primitives d'ordonnancement disponibles dans l'outil IBM ILOG CpOptimizer [1], dont certaines servent justement à définir des hiérarchies de tâches. La modélisation obtenue est présentée ci-après. Elle fait tout d'abord intervenir, pour chaque tâche τ du problème, un intervalle temporel itv_τ (appelée une *variable intervalle* dans CpOptimizer). Chaque intervalle itv est défini par une variable $startOf(itv)$ représentant la date de début de l'intervalle, une variable $endOf(itv)$ représentant la date de fin de l'intervalle, et une longueur $lengthOf(itv)$ donnant la distance entre le début et la fin de l'intervalle. Pour les intervalles associés aux activités, cette distance est connue initialement et égale à la durée de l'activité (voir équation 6). Pour les intervalles associés aux tâches composites, cette distance n'est pas connue initialement (voir équation 7), et il est seulement possible de spécifier que les intervalles en question doivent se finir avant un horizon temporel maximum T considéré.

Sur cette base, on retrouve ensuite dans le modèle obtenu les différentes contraintes listées précédemment dans les équations 1 à 5. La formalisation utilise notamment la contrainte *span* spécifiant qu'un intervalle doit couvrir exactement un ensemble d'intervalles, la contrainte *noOverlap* imposant un non chevauchement temporel entre des intervalles, et la contrainte *endBeforeStart* imposant que la date de fin d'un intervalle soit inférieure ou égale à la date de début d'un autre intervalle. Le problème ainsi modélisé peut ensuite être résolu à l'aide de l'outil CpOptimizer.

Variables :

$$\forall a \in \mathcal{A},\ \textbf{dvar interval}\ itv_a \,\text{size}\, du_a \,\text{in}\, [0..\text{T}] \quad (6)$$
$$\forall c \in \mathcal{C},\ \textbf{dvar interval}\ itv_c \,\text{in}\, [0..\text{T}] \quad (7)$$

Constraints :

$$\forall c \in \mathcal{C},\ span(itv_c, \{itv_\tau \mid \tau \in SubTsk_c\}) \quad (8)$$
$$\forall r \in \mathcal{R},\ noOverlap(\{itv_\tau \mid \tau \in \mathcal{T}_r\}) \quad (9)$$
$$\forall c \in \mathcal{C},\ \forall (\tau, \tau') \in \mathcal{P}_c,\ endBeforeStart(itv_\tau, itv_{\tau'}) \quad (10)$$

3 Abstractions d'une tâche composite

La méthode de programmation par contraintes présentée précédemment s'appuie sur une version complètement dépliée du réseau de tâches hiérarchiques. Ce dépliage complet peut cependant être coûteux du point de vue de la résolution lorsque le nombre de tâches atomiques et composites augmente. Pour faciliter le passage à l'échelle et guider la résolution à plus haut niveau, nous définissons une méthode qui évite le dépliage systématique de toutes les tâches au début de

1. `https://www-01.ibm.com/software/commerce/optimization/cplex-cp-optimizer/`

la recherche. La méthode introduite raisonne sur des *abstractions des tâches composites* dans un premier temps, et elle *raffine* ensuite pas à pas ces abstractions lorsque cela s'avère nécessaire, en revenant progressivement à une modélisation non abstraite pour certaines tâches composites.

Dans ce contexte, abstraire une tâche composite signifie ne pas représenter finement toutes les sous-tâches qui la composent et raisonner à plus gros grain en estimant l'impact global de ces sous-tâches sur le problème d'ordonnancement à résoudre. Plus précisément, nous cherchons à abstraire chaque tâche composite c par une tâche atomique notée $Abs(c)$ définie simplement par une durée $du_{Abs(c)}$ et par un ensemble ressources $\mathcal{R}_{Abs(c)}$ consommées pendant toute la durée de c. Nous détaillons ci-après les techniques utilisées pour définir les quantités $du_{Abs(c)}$ et $\mathcal{R}_{Abs(c)}$. Les abstractions sont calculées en partant des tâches atomiques (avec pour toute tâche atomique $a \in \mathcal{A}$ la convention $Abs(a) = a$) et se propagent progressivement jusqu'aux tâches de plus haut niveau dans la hiérarchie des tâches.

3.1 Durée de l'abstraction d'une tâche composite

Pour définir la durée $du_{Abs(c)}$ associée à l'abstraction d'une tâche composite $c \in \mathcal{C}$, il est tout d'abord possible de considérer le problème de l'ordonnancement des abstractions des sous-tâches de c indépendamment des autres tâches composites du problème. Ce problème Pb contient une activité $Abs(\tau)$ pour chaque sous-tâche $\tau \in SubTsk_c$, et une contrainte de précédence $Abs(\tau) \rightarrow Abs(\tau')$ pour chaque contrainte de précédence $\tau \rightarrow \tau' \in \mathcal{P}_c$. On suppose ensuite que l'on dispose d'une procédure capable de produire rapidement une solution S au problème Pb. Cette procédure peut par exemple correspondre à l'utilisation d'une règle heuristique qui insère les tâches les unes après les autres dans le plan en suivant un ordre d'insertion fonction des caractéristiques des tâches du problème. Lorsque le problème Pb reste simple, elle peut également correspondre à une résolution à l'aide d'un outil de résolution complet capable de produire un ordonnancement minimisant le makespan. Notons que dans le cas où les contraintes de précédence associées à c sont telles que toutes les sous-tâches de c doivent être réalisées en séquence, obtenir une solution S minimisant le makespan est immédiat (dans ce cas, le makespan optimal vaut $\sum_{\tau \in SubTsk_c} du_{Abs(\tau)}$).

La solution S trouvée en ordonnançant les abstractions des sous-tâches de c peut ensuite être utilisée pour associer à l'abstraction de c une durée $du_{Abs(c)}$ égale au makespan $mk(S)$ de la solution S. Intuitivement, cette abstraction est pessimiste dans le sens où elle considère une durée qui est de toutes façons suffisante pour réaliser l'intégralité de la tâche c.

La figure 3 montre une tâche composite de départ et un plan minimisant le makespan qu'il est possible d'obtenir directement vu les contraintes de précédence. La solution donnée à la figure 3(b) donnerait une durée égale à 4 unités de temps avec l'abstraction choisie.

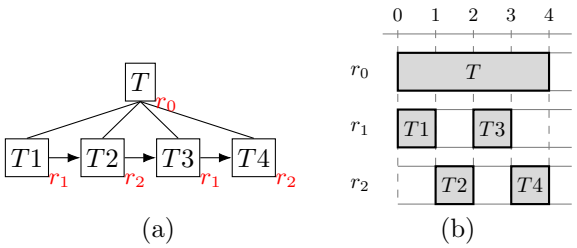


FIGURE 3 – Ordonnancement sur une tâche composite individuelle : (a) tâche composite T de départ faisant intervenir 4 sous-tâches de durée 1, pour un problème impliquant trois ressources r_0, r_1, r_2 ; (b) ordonnancement solution pour le problème restreint à la tâche T

3.2 Consommations de ressource pour l'abstraction d'une tâche composite

Pour abstraire les consommations de ressources provenant des sous-tâches de c, nous considérons deux versions :

- une version notée $C0$, donnée à l'équation 11, dans laquelle on inclut dans les consommations de $Abs(c)$ uniquement les ressources de $\mathcal{R}_c$ qui sont consommées directement par c ;
- une version notée $C1$, donnée à l'équation 12, dans laquelle on ajoute dans les consommations de $Abs(c)$ l'ensemble des ressources consommées par les abstractions des sous-tâches de c (nous rappelons que les abstractions sont construites en suivant une approche *bottom-up* dans la hiérarchie des tâches).

Intuitivement, la version $C0$ est une version plutôt optimiste dans laquelle on considère que de toutes façons les consommations des ressources par les sous-tâches ne seront pas limitantes pour l'ordonnancement (sur l'exemple du dernier niveau de hiérarchie pour le problème de déploiement de robots, cette approche revient à considérer en première approximation que la ressource réseau ne freinera pas les déplacements des robots). La version $C1$ est quant à elle une version plus pessimite (ou *robuste*) dans le sens où elle réserve pendant toute la durée de l'abstraction toutes les ressources qui pourraient être consommées par des sous-tâches.

$$C0 \ : \ \mathcal{R}_{Abs(c)} = \mathcal{R}_c \tag{11}$$

$$C1 \ : \ \mathcal{R}_{Abs(c)} = \mathcal{R}_c \cup (\cup_{\tau \in SubTsk_c} \mathcal{R}_{Abs(\tau)}) \tag{12}$$

Nous obtenons ainsi deux méthodes d'abstraction à étudier, notées respectivement Abs^{C0}, Abs^{C1}. Ces abstractions sont illustrées à la figure 4. L'approche Abs^{C1} conduit à des ordonnancements de tâches qui d'une certaine manière cloisonnent les consommations de ressources par les tâches. Cette stratégie permet de construire des ordonnancements qui ne contiennent aucune *interférence* entre les sous-tâches de tâches composites différentes. Cela a également pour effet de garantir que l'ordonnancement construit sur la base des abstractions peut à tout instant être étendu à un ordonnancement complet. Dans ce sens, l'abstraction Abs^{C1} est *robuste*. A l'opposé, l'abstraction Abs^{C0} est optimiste.

Notons enfin que les abstractions obtenues raisonnent implicitement comme si les sous-tâches de tâches différentes ne pouvaient pas être *entrelacées* les unes avec les autres sur les ressources, c'est-à-dire en ne considérant pas de schéma d'exécution dans lequel une ressource est utilisée d'abord pour les sous-taches d'une tâche composite c_1, puis par des sous-tâches d'une autre tâche composite c_2, puis à nouveau par des sous-tâches de c_1. Les optimisations utilisant de l'entrelacement de tâches sur les ressources sont laissées à des phases ultérieures de la recherche, au cours desquelles les tâches abstraites sont raffinées si cela s'avère utile.

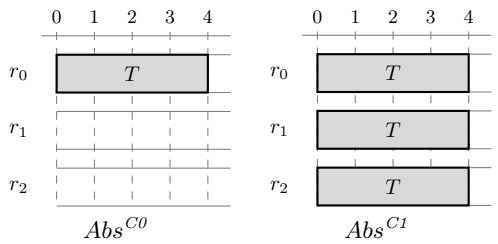


FIGURE 4 – Différentes formes d'abstraction obtenues à partir de la solution de la figure 3

3.3 Remarques

La démarche globale utilisée fait le choix d'essayer de mettre à profit la hiérarchie de décomposition des tâches pour guider la recherche. La hiérarchie de décomposition est ainsi exploitée au niveau algorithmique, et pas seulement pour des raisons de modélisation. D'autres méthodes seraient envisageables, comme chercher à grouper automatiquement des tâches du problème qui sont fortement contraintes les unes par rapport aux autres, réaliser des abstractions par catégories de ressources... Par ailleurs, la méthode utilisée explore l'utilisation d'abstractions simples en définissant une durée et un ensemble de consommations pour chaque tâche composite. D'autres méthodes utilisant une vue plus détaillée de l'ordonnancement solution trouvé pour chaque tâche composite pourront être explorées dans des travaux futurs. Enfin, notons que les abstractions sont calculées sur la base d'un ordonnancement qui correspond à une solution effective pour chaque tâche composite, contrairement à une approche qui raisonnerait uniquement par propagation de contraintes pour obtenir des bornes inférieures sur la durée de ces tâches.

4 Stratégies de décomposition itérative

Nous définissons maintenant la stratégie globale de raffinement utilisée pour passer progressivement d'un plan construit pour un problème P_0 qui contient uniquement les tâches atomiques et les abstractions des tâches de haut niveau, à un plan construit pour un problème P_n qui correspond au problème d'origine $(\mathcal{R}, \mathcal{A}, \mathcal{C})$. Pour passer du problème P_i au problème raffiné P_{i+1}, le principe adopté consiste à sélectionner à chaque étape de l'algorithme un ensemble non vide de tâches composites qui sont présentes dans P_i sous leur forme abstraite, et à raffiner ces tâches de telle sorte que le nouveau problème P_{i+1} obtenu contienne au moins une tâche abstraite en moins. Différents réglages doivent être faits pour définir exactement comment passer du problème P_i au problème P_{i+1}.

Nombre de tâches à raffiner Le premier réglage à réaliser concerne le nombre de tâches abstraites qui sont raffinées à chaque étape, avec comme réglages possibles la sélection d'une seule tâche abstraite à chaque étape ou la sélection de K tâches abstraites à raffiner simultanément.

Heuristique de sélection des tâches à raffiner Le second réglage à réaliser concerne le choix d'une heuristique permettant de privilégier le raffinement des tâches abstraites les plus intéressantes. Dans les expérimentations, nous étudions trois heuristiques de sélection, notées respectivement $H1$, $H2$ et $H3$. Ces abstractions sont illustrées :

— la première heuristique privilégie les tâches abstraites dont la date de début est minimale ; cette heuristique est notamment bien adaptée pour les approches dites en ligne dans lesquelles l'exécution du plan est réalisée en parallèle de la planification, et donc pour lesquelles les premières actions du plan doivent être engagées à un certain instant potentiellement proche ;

— la deuxième heuristique privilégie les tâches abstraites situées au plus haut niveau de la hiérarchie, c'est-à-dire les tâches abstraites pour

lesquelles le nombre de tâches composites ascendantes est minimal ; cette approche reprend quelque part des principes d'une approche classique de gestion des problèmes d'ordonnancement hiérarchique, consistant à traiter d'abord complètement le problème de décision de plus haut niveau avant d'aborder les spécifications plus fines du problème ;

— la troisième heuristique privilégie les tâches abstraites qui apparaissent sur un chemin critique dans le plan solution S trouvé pour le problème P_i, c'est-à-dire les tâches abstraites qui ont une flexibilité temporelle nulle dans S ; l'objectif est ici de concentrer d'abord la recherche sur les points durs du problème.

Pour toutes les heuristiques, s'il existe des tâches abstraites ex æquo au sens des critères utilisés par l'heuristique, alors ces tâches sont départagées aléatoirement.

Informations transmises entre les itérations Pour que la résolution du problème pas à pas présente un intérêt plutôt que le raisonnement direct sur le problème complet, il est utile de transmettre des informations entre les résolutions successives. Autrement dit, il est utile de définir des méthodes pour que les premières résolutions guident les résolutions ultérieures, ces dernières pouvant en effet se retrouver plus facilement coincées dans certaines zones d'un espace de recherche potentiellement beaucoup plus grand suite au raffinement. Deux types de transmissions d'information sont envisagées :

— soit la transmission d'une contrainte imposant que le makespan obtenu pour le problème P_{i+1} soit inférieur ou égal au makespan de la solution trouvée pour le problème P_i ; cette contrainte est effectivement utilisée uniquement lorsque les abstractions choisies sont pessimistes, car dans le cas contraire il n'est pas garanti que le makespan optimal de P_{i+1} puisse être inférieur ou égal au makespan de P_i ;

— soit la transmission de contraintes de précédence de type "start-to-start" entre tâches, l'intuition étant que les résolutions gros grain ont potentiellement permis de synthétiser de bons ordres d'utilisation des ressources par les tâches ; de manière plus précise, partant du plan solution trouvé pour le problème P_i, on extrait pour chaque ressource $r \in \mathcal{R}$ toutes les tâches de P_i qui utilise r, on classe ces tâches par date de début croissante, et pour toute paire de tâches τ, τ' qui se succèdent dans cet ordre, on ajoute à P_{i+1} la contrainte $s_\tau \leq s_{\tau'}$ imposant que τ commence avant τ' si cette contrainte n'était pas déjà présente dans P_i.

Algorithme de décomposition itérative L'algorithme 1 décrit la méthode globale utilisée. Dans cette dernière, on retrouve une phase initiale de construction du problème où toutes les tâches composites sont abstraites (ligne 1), une résolution de ce problème avec un temps de calcul maximum *MaxTimeIter* donné en entrée de l'algorithme (ligne 2), puis des décompositions itératives utilisées tant qu'il reste des tâches abstraites (lignes 3 à 7). Lors de chaque étape de décomposition, l'algorithme sélectionne un ensemble de tâches abstraites à décomposer (ligne 4), extrait des contraintes à partir de la solution courante S (ligne 5), calcule un problème P actualisé (ligne 6) et résout ce nouveau problème toujours avec un temps de calcul maximum (ligne 7). La solution trouvée sur le dernier problème est enfin renvoyée (ligne 8). Ce pseudo-code pourrait être adapté pour répartir le temps de calcul global autrement qu'en allouant le même temps de calcul à chaque itération du raffinement. L'étude de réglages plus évolués du temps de calcul par itération est laissée pour des travaux ultérieurs.

Algorithm 1: $iterativeDecomp(\mathcal{R}, \mathcal{A}, \mathcal{C}, MaxTimeIter)$: algorithme de décomposition itérative pour un problème d'ordonnnancement hiérarchique

```
1 P ← fullAbstractProblem(R, A, C);
2 S ← solve(P, MaxTimeIter);
3 while getAbsTasks(P) ≠ ∅ do
4     ToDecompose ← selectAbsTasks(P);
5     Ctrs ← extractConstraints(S);
6     P ← update(P, ToDecompose, Ctrs);
7     S ← solve(P, MaxTimeIter);
8 return S;
```

5 Expérimentations

Dans cette partie, nous montrons les résultats préliminaires obtenus sur différents benchmarks avec les méthodes décrites précédemment.

5.1 Benchmarks

Problème d'exploration multi-robot De manière à tester les méthodes sur des problèmes de grande taille proches des applications visées, nous avons généré des benchmarks autour de l'exploration multi-robots décrite dans l'exemple de la section 2. Nous avons créé un générateur pour ce type de benchmarks qui prend en paramètre les éléments suivants :

— **nZones :** nombre de zones d'observations ;

— **nObsPerZone :** nombre d'observations nécessaires à l'observation de chaque zone ;

— **nRobots :** nombre de robots disponibles pour réaliser les observations de zone ;

— **nFréquences :** nombre de fréquences disponibles pour émettre pendant une observation ;
— **nPointsTransferts :** nombre de points de passage total ;
— **nPointsPerMove :** nombre de points de passage pour un seul transfert pour un robot ;

Il est également possible de paramétrer les bornes des durées des différentes tâches atomiques du problème.

Nous montrons les résultats sur deux instances représentatives. Nous nommons la première *5Z* (5 zones) : c'est une petite instance avec 5 zones à observer, 2 robots pour les réaliser, 3 observations par zone, 10 points de passage au total, 3 points de passage par déplacement et 5 fréquences disponibles. La deuxième instance est nommée *50Z* (50 zones) et correspond à une plus grande instance avec 50 zones, 4 robots pour les réaliser, 10 observations par zone, 15 points de passage au total, 3 points de passage par déplacement et 3 fréquences disponibles.

Problème d'Open shop et de Job shop Nous avons également testé les méthodes sur des benchmarks de la littérature, à savoir les problèmes de type *Open Shop* et *Job Shop* disponibles sur la page de E .Taillard [2]. Pour les problèmes d'*Open shop*, nous utilisons deux instances 10×10 et 20×20 [3]. Plus précisément, nous transformons l'instance 10×10 (resp. 20×20) en un problème hiérarchique à deux niveaux avec 10 (resp. 20) tâches composites, 10 (resp. 20) sous-tâches par tâche, et 10 (resp. 20) ressources disjonctives. Pour les problèmes de *Job Shop*, nous utilisons les instances de taille 20×20 et 100×20. Nous transformons l'instance 20×20 (resp. l'instance 100×100) en un problème hiérarchique à deux niveaux avec 20 (resp. 100) tâches composites, 20 sous-tâches par tâche, et 20 ressources disjonctives.

5.2 Résultats

Le tableau 1 présente les résultats pour les deux instances du problème multi-robot. Pour la résolution de ces problèmes, un temps de calcul maximum *MaxTime* est fixé à 1 minute pour l'instance *5Z* et à 5 minutes pour l'instance *50Z* du problème. Lors de la résolution avec abstraction, le nombre d'itérations *MaxTimeIter* décrit dans l'algorithme 1 correspond au *MaxTime* réparti équitablement entre toutes les itérations. Dans ces expérimentations, les informations transmises entre les itérations sont des précédences de type "start-to-start".

2. `http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html`
3. Plusieurs instances ont ces tailles. Nous considérons les premières à chaque fois.

Pour chaque résolution du problème abstrait en utilisant les heuristiques de décomposition décrites précédemment, il faut choisir le nombre de tâches abstraites à décomposer simultanément K. Nous exprimons ici ce nombre en pourcentage p du nombre total de tâches composites. Formellement, $K = \lceil p \cdot \mathbf{card}(\mathcal{C}) \rceil$. Le pourcentage p est indiqué entre parenthèses à côté de l'heuristique utilisée. Nous avons ici choisi de décomposer soit 5% des tâches à chaque itération, soit 20%. La colonne *TB* indique le temps en secondes pour trouver la meilleure solution et la colonne *TT* indique le temps total utilisé (en secondes). Par rapport à l'algorithme 1 présenté dans la section 4, on note *TT* la date à laquelle chaque solution S est calculée, et *TB* représente la date à laquelle on trouve le makespan de la solution finale. Autrement dit, le makespan de la solution n'est pas amélioré entre *TB* et *TT*. Étant donné que pour l'abstraction Abs^{C0}, le makespan obtenu à chaque itération ne peut pas être considéré comme une vraie borne supérieure, le *TB* obtenu n'est pas indiqué dans les tableaux suivants.

Inst.	Abs.	H(%)	BestMks	TB	TT
5Z	Sans Abstraire		**158***	0.03	17.4
	Abs^{C0}	H1(5)	158	-	31.2
		H1(20)	158	-	30.9
		H2(5)	158	-	34.3
		H2(20)	158	-	44.2
		H3(5)	158	-	15.7
		H3(20)	158	-	31.8
	Abs^{C1}	H1(5)	158	17.0	29.7
		H1(20)	158	12.4	37.6
		H2(5)	158	24.9	46.2
		H2(20)	158	31.9	43.2
		H3(5)	158	31.0	43.3
		H3(20)	158	24.3	45.3
50Z	Sans Abstraire		3785	28	*MaxTime*
	Abs^{C0}	H1(5)	3919	-	*MaxTime*
		H1(20)	3910	-	*MaxTime*
		H2(5)	3919	-	*MaxTime*
		H2(20)	3910	-	*MaxTime*
		H3(5)	3919	-	*MaxTime*
		H3(20)	3910	-	*MaxTime*
	Abs^{C1}	H1(5)	3919	297	*MaxTime*
		H1(20)	3910	299	*MaxTime*
		H2(5)	3919	300	*MaxTime*
		H2(20)	3910	298	*MaxTime*
		H3(5)	3919	298	*MaxTime*
		H3(20)	3910	299	*MaxTime*

TABLE 1 – Résultats pour le problème d'exploration multi-robots.

La ligne *Sans Abstraire* correspond à la résolution de la traduction du problème en Programmation par

Contraintes par l'outil CpOptimizer. Lorsque CpOptimizer a prouvé que la borne était l'optimum, nous l'indiquons avec le symbole $*$.

Les tableaux 2 et 3 présentent les résultats pour les deux instances du problème d'*Open Shop* et les deux instances du problème de *Job Shop* respectivement, avec un temps de calcul maximum de 10 secondes.

Inst.	Abs.	H(%)	BestMks	TB	TT
10×10	Sans Abstraire		**637***	0.67	1.16
	Abs^{C0}	H1(20)	637	-	1.88
		H2(20)	637	-	1.89
		H3(20)	637	-	1.86
	Abs^{C1}	H1(20)	637	8.82	9.35
		H2(20)	637	8.86	9.52
		H3(20)	637	8.68	9.68
20×20	Sans Abstraire		**1155***	0.66	1.57
	Abs^{C0}	H1(20)	1155	-	1.9
		H2(20)	1155	-	1.91
		H3(20)	1155	-	1.82
	Abs^{C1}	H1(20)	1155	9.30	9.75
		H2(20)	1155	9.40	9.81
		H3(20)	1155	9.35	9.79

TABLE 2 – Résultats du problème d'Open shop

Les résultats obtenus sont des résultats préliminaires dans le contexte de nos objectifs de recherche et nous serviront de guide pour les expériences futures. Ils révèlent que l'outil d'optimisation CpOptimizer arrive à trouver des bonnes solutions assez rapidement. En effet, même pour le plus gros problème testé, une première solution est trouvée en 28 secondes, mais cette solution n'est pas améliorée dans la suite de la résolution.

Sur des petits problèmes, notre méthode est capable de trouver le makespan optimal trouvé par CP Optimizer. En revanche, sur des plus grands problèmes, en rajoutant les contraintes de précédence de type "start-to-start" entre les résolutions successives, l'espace de recherche est coupé et on risque potentiellement, d'enlever la solution optimale de l'espace de recherche. Notons tout de même que les solutions sont proches du $BestMks$ trouvé pour le problème sans abstraire (3.5%).

Même si les deux abstractions ont des résultats similaires dans les tableaux présentés ici, il s'avère que leur comportement dans la phase de recherche est très différent. Par exemple, en utilisant l'heuristique de décomposition $H3(5\%)$ pour l'instance 50Z, la première borne obtenue pour l'abstraction Abs^{C0} est de 3713, et cette dernière est considérablement plus élevée pour l'abstraction Abs^{C1} avec une valeur de 8083.

L'écart entre CpOptimizer et la méthode basée sur l'abstraction et la décomposition vient certainement

Inst.	Abs.	H(%)	BestMks	TB	TT
20×20	Sans Abstraire		**1217***	0.02	1.87
	Abs^{C0}	H1(20)	1217	-	2.38
		H2(20)	1217	-	2.48
		H3(20)	1217	-	2.35
	Abs^{C1}	H1(20)	1217	9.47	9.85
		H2(20)	1217	9.54	9.91
		H3(20)	1217	9.48	9.98
100 $\times$20	Sans Abstraire		**5464***	0.99	3.09
	Abs^{C0}	H1(20)	5464	-	6.73
		H2(20)	5464	-	7.05
		H3(20)	5464	-	6.74
	Abs^{C1}	H1(20)	5464	9.56	10.0
		H2(20)	5464	9.57	10.0
		H3(20)	5464	9.47	10.0

TABLE 3 – Résultats du problème de Job shop

du fait que les abstractions utilisées sont très agressives dans leur manière de définir la consommation de ressources. Les travaux futurs consisteront à définir des abstractions plus fines.

6 Conclusion

Dans ce papier, nous avons défini un premier cadre de modélisation pour les problèmes d'ordonnancement hiérarchique. Nous avons proposé une traduction en programmation par contraintes et définissons une méthode basée sur des mécanismes d'abstraction et de décomposition des tâches. Les expérimentations montrent que cette dernière pourrait largement être améliorée en considérant des stratégies plus fines d'abstraction. Une piste serait de définir des abstractions sur des problèmes utilisant des ressources cumulatives (ressources avec une certaine capacité), et en conséquence pouvoir considérer le pourcentage de consommation d'une ressource par une sous-tâche. Cela permettra notamment de prendre en compte plus finement la consommation des ressources des sous-tâches. Il sera utile de réexaminer aussi, les informations qui sont transmises entre les itérations, pour que l'espace de recherche ne se retrouve pas largement découpé, comme c'est le cas avec les contraintes de précédence "start-to-start".

D'autres perspectives consistent à étendre le cadre de modélisation. On pourra par exemple prendre en compte les ressources flexibles, c'est-à-dire permettre à chaque tâche de consommer une ressource parmi un ensemble de ressources disponibles. On pourra également définir pour chaque tâche composite des méthodes de décomposition, comme dans le cadre HTN.

Références

[1] P. Bechon, M. Barbier, C. Grand, S. Lacroix, C. Lesire, and C. Pralet. Integrating planning and execution for a team of heterogeneous robots with time and communication constraints.

[2] P. Bechon, M. Barbier, G. Infantes, C. Lesire, and V. Vidal. Hipop : Hierarchical partial-order planning. In *In Proceedings of the 7th European Starting AI Researcher Symposium (STAIRS'14)*, 2014.

[3] P. Brucker, A. Drexl, R. Möring, K. Neumann, and E. Pesch. Resource-constrained Project Scheduling : Notation, Classification, Models, and Methods. *European Journal of Operational Research*, 112(1) :3–41, 1999.

[4] J. Colomé, P. Colomer, J. Guàrdia, I. Ribas, J. Campreciós, T. Coiffard, .L Gesa, F. Martínez, and F. Rodler. Research on schedulers for astronomical observatories. In *Observatory Operations : Strategies, Processes, and Systems IV*, volume 8448, page 84481L. International Society for Optics and Photonics, 2012.

[5] F. Dvorak, R. Barták, A. Bit-Monnot, F. Ingrand, and M. Ghallab. Planning and acting with temporal and hierarchical decomposition models. In *26th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2014, Limassol, Cyprus, November 10-12, 2014*, pages 115–121, 2014.

[6] K. Erol, J. Hendler, and D. S. Nau. HTN planning : Complexity and expressivity. In *In Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'94)*, pages 1123–1128. AAAI Press, 1994.

[7] S. Girbal, D. G. Pérez, J. Le Rhun, M. Faugère, C. Pagetti, and G. Durrieu. A complete toolchain for an interference-free deployment of avionic applications on multi-core systems. In *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*, pages 7A2–1–7A2–14, Sept 2015.

[8] D. Nau, T. C. Au, O. Ilghami, U. Kuter, D. Wu, F. Yaman, H. Munoz-Avila, and J. W. Murdock. Applications of shop and shop2. *IEEE Intelligent Systems*, 20(2) :34–41, March 2005.

[9] M. L. Pinedo. *Scheduling : Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 3rd edition, 2008.

[10] C. Qi, D. Wang, H. Muñoz-Avila, P. Zhao, and H. Wang. Hierarchical task network planning with resources and temporal constraints. *Knowledge-Based Systems*, 133 :17–32, 2017.

Contraintes Flexibles Evidentielles

Aouatef Rouahi[1] **Kais Ben Salah**[2] **Khaled Ghedira**[3]

[1] ISG of Tunis, Tunis University, Tunisia
[2] FCIT, University of Jeddah, SA
[3] Central University, Tunisia
rouahi.aouatef@hotmail.fr kaisuria@yahoo.fr khaled.ghedira@isg.rnu.tn

Résumé

Alors qu'ils sont souvent amenés à raisonner à partir des informations incomplètes et incertaines, les agents ne sont pas capables d'exprimer leurs préférences selon leurs désirs mais en concordance avec leurs croyances. Dans cet article, nous introduisons un modèle pour les préférences évidentielles basées sur les croyances en combinant l'approche des contraintes flexibles et la théorie de l'évidence. En faisant ainsi, les préférences d'un agent sont manipulées avec une perspective à deux niveaux : (1) l'agent exprime ses croyances sur ses préférences et éventuellement son ignorance ou son hésitation, (2) les préférences de l'agent sont dérivées de sa base de croyances. En plus, en exploitant la richesse de la théorie de l'évidence, nous avons été capables de modeler des préférences plus sophistiquées, à savoir, les préférences conflictuelles et les préférences distordues. Les techniques de résolution des problèmes de satisfaction de contraintes sont ensuite adaptées pour résoudre les problèmes avec ce genre de préférences.

1 Introduction

Motivation Alors que l'imperfection envahit notre vie réelle, les agents sont tenus d'exprimer leurs préférences en fonction de leurs désirs, tandis que leurs croyances sont ignorées ou confondues avec leurs préférences. Cependant, la distinction et la séparation entre les croyances et les préférences sont importantes pour plusieurs raisons. D'abord, il y a des situations dans lesquelles un agent peut seulement avoir des informations partielles et/ou incertaines concernant les alternatives sur lesquelles il est tenu de fournir ses préférences, c'est-à-dire, des alternatives mal définies. Dans ce cas, les croyances doivent être introduites afin de dériver les préférences de l'agent. Ensuite, accepter que les préférences d'un agent sont fonction de ses croyances donne un sens au changement de préférences sous le changement des croyances. Enfin, même si l'information est disponible, elle peut être ambiguë, contradictoire ou excessive. Ainsi, une distinction entre les croyances et les préférences est nécessaire, de sorte qu'un agent peut exprimer son hésitation en éprouvant une certaine confusion. En plus, les agents peuvent fournir des préférences inconsistantes ou conflictuelles en ce qui concerne différents contextes, par exemple, lorsque on se retrouve avec une alternative a qui est préférée à une alternative b dans un contexte, mais pas dans un autre différent. En outre, indépendamment des informations disponibles, la préférence elle-même peut être mal définie même si les alternatives sont bien définies. En d'autres termes, les préférences fournies peuvent ne pas refléter les préférences réelles de l'agent, à savoir, les préférences distordues. Évidemment, cela nécessite une combinaison d'une théorie doxastique et d'un formalisme pour les préférences. Jusqu'à présent, il n'y a pas eu aucun formalisme permettant à toutes ces sortes de préférences d'être représentées de façon compacte et efficacement manipulées. La théorie de l'évidence [3, 12, 14] offre un outil naturel pour gérer l'imperfection et fournit une base mathématique solide qui reconnaît conformément tous nos états de connaissance allant de la connaissance totale à l'ignorance totale. De plus, en considérant le Modèle des Croyances Transférables (MCT), une interprétation non probabiliste de la théorie de l'évidence développée par [14] qui permet la distinction entre la représentation des connaissances et la prise de décision, nous introduisons un modèle de préférence à deux niveaux : (1) une base de croyances où l'agent exprime ses croyances concernant ses préférences voire son hésitation ou son ignorance ; (2) les préférences finales dérivées de cette base.

D'autre part, les contraintes flexibles [2], équipées d'un mécanisme de résolution puissant, fournissent une approche intéressante pour modéliser et raisonner avec les préférences quantitatives. Cependant, la notion de préférence n'avait toujours pas été adéquatement traitée dans le cadre de contraintes flexibles dans le sens où les préférences sont essentiellement utilisées pour relâcher les problèmes sur-contraints, pour distinguer entre les différentes solutions ou pour réduire le coût de recherche. Ainsi, une structure de préférence n'est pas clairement définie lorsque certaines relations de préférence spécifiques sont implicitement déclarées telles que la relation d'indifférence, où l'agent doit évaluer toutes les alternatives et associer le même degré de préférence aux alternatives entre lesquelles il est indifférent ce que lui pose une charge cognitive, alors qu'on peut évaluer ces alternatives sous forme d'ensembles. Une autre relation de préférence importante, qui est ignorée dans le cadre de contraintes flexibles, est la relation d'incomparabilité. Notre but dans cet article est de mettre en évidence la connexion intéressante entre les croyances et les préférences en introduisant un formalisme de contrainte flexibles basé sur la théorie de l'évidence. Nous définissons la notion de relation de préférence évidentielle et ses propriétés dans ce cadre. Ensuite, nous montrons comment les techniques de résolution des problèmes de satisfaction de contraintes peuvent être adaptées pour résoudre ces problèmes avec ce genre de préférences.

Travaux Connexes Les préférences basées sur les croyances sont devenues un sujet d'intérêt parmi les philosophes, les psychologues, les économistes, mais rarement pris en compte dans la littérature IA malgré la multitude d'approches pour les préférences sauf quelques études sur la logique des préférences [9, 10] étudiant la dynamique des préférences en fonction du changement des croyances. Au meilleur de nos connaissances, dans le domaine de la satisfaction des contraintes, ce travail fournit la première connexion entre la théorie de l'évidence et le formalisme des contraintes flexibles. Néanmoins, diverses propositions ont été introduites pour étendre le cadre des contraintes flexibles afin de représenter et gérer les préférences imparfaites. Le travail dans [5] considère les problèmes de contraintes flexibles incomplètes où certaines préférences peuvent être manquantes tant qu'il est possible de trouver une solution optimale, sinon, l'agent devra fournir certaines préférences. Dans ce travail, l'incomplétude est interprétée comme une incapacité temporaire ou une réticence à fournir des préférences sur certaines alternatives. Dans notre approche, nous considérons l'incomplétude comme un jugement décisif à ne pas comparer certaines alternatives par rapport à l'évidence disponible, ainsi, nous n'exigeons pas que l'agent fournisse des informations supplémentaires. Une autre proposition [6] considère les intervalles d'intensités pour modéliser les préférences imprécises. Dans notre travail, nous supposons que les intensités de préférence sont, précisément, énoncées. Cependant, nous autorisons l'agent d'exprimer ses préférences sur des sous ensembles des alternatives au lieu de se limiter aux singletons ce qui permet de spécifier la relation d'indifférence. Un autre travail qui adresse les contraintes flexibles incertaines utilisant la théorie des possibilités est présenté dans [11] où certaines alternatives peuvent être mal définies, c'est-à-dire qu'on ne peut pas décider leurs valeurs. Dans notre travail, nous abordons l'incertitude à deux niveaux. Premièrement, nous considérons le cas où les alternatives peuvent être mal définies, de sorte que l'agent ne peut pas exprimer ses préférences sous la forme de "oui/non" mais il peut répondre "je préfère quelque peu cette alternative" en utilisant des intensités de préférences, il peut également hésiter à exprimer ses préférences en répondant "je ne suis pas sûr", ou il peut simplement dire "je ne sais pas" pour exprimer son ignorance. Deuxièmement, nous considérons le cas où la relation de préférence elle-même peut être mal définie, c'est-à-dire que les préférences réelles d'un agent peuvent être distordues. Ainsi, dans notre proposition, l'intensité de la vérité d'une préférence donnée est évaluée. Enfin, grâce à notre approche à deux niveaux, nous avons pu capturer les différentes positions préférentielles d'un agent, à savoir, la préférence stricte, l'indifférence et l'incomparabilité. De plus, notre modèle permet de représenter et gérer adéquatement des préférences éventuellement conflictuelles et distordues dans un formalisme unifiant.

Le reste de l'article est organisé comme suit : Quelques préliminaires sont discutés dans la section 2. Nous explorons la connexion étroite entre les préférences et les croyances dans la section 3, où nous introduisons la notion de préférence évidentielle basée sur les croyances, ainsi que les notions de préférences conflictuelles et distordues. Dans la section 4, nous montrons comment les mécanismes de résolution basés sur les contraintes sont adaptés pour résoudre le modèle de préférence évidentielle basé sur les croyances. Les résultats sont présentés dans la section 5 et les conclusions et les perspectives sont données dans la section 6.

2 Préliminaires

2.1 Le Formalisme des Contraintes Flexibles

Les contraintes flexibles, à savoir les problèmes de satisfaction de contraintes basés sur les semi-anneaux

"Semiring" (SCSP) et les problèmes de satisfaction de contraintes valués (VCSP) [2] sont des formalismes génériques pour les préférences quantitatives couvrant plusieurs autres spécifiques.

En général, dans un SCSP, une contrainte flexible ou une relation de préférence est définie en associant un degré d'un ensemble partiellement ordonné avec chaque alternative impliquée indiquant à quelle mesure une alternative est préférée. En plus, deux opérations avec certaines propriétés sont fournies pour comparer $(+)$ et pour combiner $(\times)$ les degrés de préférence afin de sélectionner la meilleure solution. Formellement, un c-semi-anneau est un quintuplet $< A, +, \times, 0, 1 >$ tels que : A est un ensemble, et $0, 1 \in A$; $+$ est commutative, associative, idempotent, 0 est son élément identité, et 1 est son élément absorbant ; $\times$ est commutative, associative, distributive par rapport $+$, 1 est son élément identité et 0 est son élément absorbant. Considérons la relation $\leq_S$ définie sur A tel que $a \leq_S b$ ssi $a + b = b$. De ce fait, $\leq_S$ est un ordre partiel ; $+$ et $\times$ sont monotones par rapport $\leq_S$; 0 est son minimum et 1 son maximum ; $< A, \leq_S >$ est un treillis et pour tout $a, b \in A, a + b = lub(a, b)$. En plus, si $\times$ est idempotent, alors $< A, \leq_S >$ est un treillis distributif et $\times$ est son glb. Étant donné un c-semi-anneau $S =< A, +, \times, 0, 1 >$, un ensemble fini D, et un ensemble ordonné de variables V, une contrainte est le doublet $< def, con >$ où $con \subseteq V$ et $def : D^{|con|} \longrightarrow A$.

Dans un VCSP, chaque contrainte flexible, c'est-à-dire l'ensemble entier des alternatives, est associée à un degré d'un ensemble totalement ordonné indiquant à quelle mesure la satisfaction d'une contrainte donnée est préférée. En plus, une opération avec certaines propriétés est fournie pour combiner $(\circledast)$ les différentes contraintes. Formellement, une structure de valuation est un quintuplet $(E, \circledast, \succ, \top, \bot)$ tels que : E est un ensemble de valuations ; $\succ$ est un ordre total défini sur E ; $\top$ et $\bot$ sont respectivement l'élément maximum et l'élément minimum de E induit par $\succ$; $\circledast$ est un opérateur binaire commutative et associative par rapport E, $\bot$ est son élément identité et $\top$ est son élément absorbant, $\circledast$ est monotone par rapport $\succ$.

Dans notre approche, nous avons adopté les deux approches pour gérer les préférences incertaines à deux niveaux différents.

2.2 La Théorie de l'évidence

La théorie de l'évidence, encore connue sous le nom théorie de Dempster-Shafer ou théorie des fonctions de croyance, a été initiée par [3], puis étendue par [12]. Plusieurs interprétations ont été introduites telles que le MCT établi par [14].

2.2.1 Concepts de Base

Concéderons Θ un cadre de discernement représentant un ensemble fini d'alternatives élémentaires. Une fonction de masse de croyance est une application $m : 2^\Theta \longrightarrow [0, 1]$ tel que :

$$\sum_{\theta \in 2^\Theta} m(\theta) = 1. \quad (1)$$

La masse $m(\theta)$, affecté à un sous ensemble θ de Θ, est une quantité finie positive de support dérivée des éléments d'évidence disponibles et exactement allouée à l'ensemble θ et à aucun sous-ensemble spécifique de θ par manque d'évidence. Alors que l'hypothèse $m(\varnothing) = 0$ correspond à l'hypothèse du monde clos [12], permettant $m(\varnothing) \geq 0$ correspond à l'hypothèse du monde ouvert [14] où la masse allouée à l'ensemble vide est considérée comme le conflit interne au sein d'une fonction de masse de croyance individuelle [1]. Dans notre modèle, nous employons l'hypothèse du monde ouvert pour permettre à l'agent d'exprimer son hésitation par rapport à ses préférences.

2.2.2 Prise de Décision

Étant donné un ensemble d'alternatives Θ et une fonction de masse de croyance m, nous voulons établir un ordre sur Θ basé sur m. Plusieurs critères de décision ont été développés. Dans notre approche, nous considérons la décision basée sur le maximum de probabilité pignistique ($BetP$) qui offre un compromis entre les stratégies pessimiste et optimiste, où un degré de probabilité plus élevé indique une alternative plus préférée. Par conséquent, la fonction de masse de croyance m est transformée à une mesure de probabilité subjective $BetP$ comme suit :

$$BetP(A) = \frac{1}{1 - m(\varnothing)} \sum_{\theta \subseteq \Theta} \frac{|A \cap \theta|.m(\theta)}{|\theta|}; \forall A \in \Theta. \quad (2)$$

2.2.3 Mesure de Conflit

Traditionnellement, le conflit externe entre différents corps d'évidence est représenté par la masse attribuée à l'ensemble vide résultant d'une combinaison conjonctive. Cette interprétation semble être inadéquate, surtout, en constatant que le conflit entre fonctions de masse de croyance identiques n'est pas nul. Pour cette raison, dans la suite, nous proposons une mesure de conflit alternative entre les différentes relations de préférence comme la distance entre les probabilités pignistiques.

2.2.4 Affaiblissement

La procédure d'affaiblissement permet de prendre en compte la fiabilité de la source fournissant la fonction de masse de croyance m. Elle consiste à pondérer les masses de croyance fournies par chaque source en utilisant un coefficient d'affaiblissement, de sorte que plus la fiabilité est faible, plus l'affaiblissement est fort. Soit m une fonction de masse de croyance donné par la source S sur Θ et soit $\alpha \in [0,1]$ le degré de confiance attribué à la source S. Si la source n'est pas entièrement fiable, la fonction de masse de croyance fourni est affaiblie dans une nouvelle fonction plus faible et moins informative notée m^α, où chaque masse perdue est réaffectée à Θ comme ignorance totale :

$$\begin{cases} m^\alpha(\theta) = \alpha.m(\theta), \forall \theta \subset \Theta \\ m^\alpha(\Theta) = (1-\alpha) + \alpha.m(\Theta) \end{cases} \quad (3)$$

2.2.5 Measure de Non-Spécificité

La mesure de Non-Spécificité (NS) quantifie le degré d'imprécision d'une fonction de masse de croyance donnée comme suit [13] :

$$NS(A) = \sum_{\theta \subseteq \Theta, \theta \neq \varnothing} m(\theta).log_{|\Theta|}(|\theta|). \quad (4)$$

La fonction de masse de croyance m est la plus imprécise (moins informative) que $NS(m)$ est grande. Par conséquent, la fonction la plus non-spécifique ($NS(m) = 1$) est la fonction représentant l'ignorance totale donnée lorsque le seul élément, totalement supporté, est Θ. Cependant, la fonction la plus spécifique ($NS(m) = 0$) est obtenue lorsque tous les éléments positivement supportés sont des singletons. Nous avons besoin d'une telle mesure pour évaluer dans quelle mesure une relation de préférence donnée est informative.

3 Modèle de Préférences Evidentielles

Alors que les modèles de préférences conventionnels ont pris la préférence en tant que concept primitif, sans tenir compte de son origine, en particulier, sous l'incertitude, nous soutenons que la préférence d'un agent devrait être, rationnellement, dérivée de ses croyances. De cette manière, nous introduisons un nouveau modèle de préférence à deux niveaux : la base de croyances en tant que phase de réflexion sur les préférences et le niveau de préférence en tant que phase de décision.

Un modèle de préférences evidentielles $\wp$ est un triple (X, D, C^p), impliquant un ensemble fini de variables X, ses domaines finis associés D et un ensemble fini de contextes de préférences C^p. Un contexte de préférence c^p est une contrainte flexible évidentielle basé sur les croyances définie par le quadruplet (S, A, B, R), où $S \subseteq X$ est la portée de la préférence délimitant l'ensemble de variables que c^p implique, $A \subseteq D^{|S|}$ est l'ensemble des alternatives sur lesquelles la relation de préférence est établie, B est la base de croyances définie sur A et R est la relation de préférence évidentielle dérivée de B. Le triple (S, A, B) spécifie le contexte de R. La notion de contexte introduite est importante car une variable peut être impliquée dans plus d'un contexte de préférence, ce qui peut conduire à un conflit. Nous reviendrons sur ce problème plus tard dans la suite.

Exemple 1. (révisé depuis [7]). Dans un restaurant, Alice compose son menu de déjeuner basé sur un plat principal "Plat" (poisson "p", viande "v"), une boisson "Boisson" (vin blanc "vb", vin rouge "vr", eau "e") et un dessert "Dessert" (gâteau "g", glace "gl") . Nous avons $X = \{Plat, Boisson, Dessert\}$ avec $D_{Plat} = \{p, v\}, D_{Boisson} = \{vb, vr, e\}$ et $D_{Dessert} = \{g, gl\}$, et $C^p = \{c_1^p, c_2^p, c_3^p\}$.

3.1 Modéliser les croyances

Étant donné la portée S de la préférence et l'ensemble d'alternatives A associé, les croyances de l'agent B sur ses préférences définies sur A sont modélisées en terme d'un ordre partiel $\trianglerighteq$ induit par la fonction de masse de croyance m sur $\{a_i \cup \varnothing\}$ à [0,1], tel que, $\bigcup_{(i=1)}^{k} a_i = A$:

$$\trianglerighteq = \{(\theta_1, \theta_2) | m(\theta_1) \geq m(\theta_2)\}. \quad (5)$$

L'instance $\theta_1 \trianglerighteq \theta 2$ signifie que "l'utilité de θ_1 est au moins aussi supportée que celle de θ_2", étant donné l'évidence détenue par l'agent. $\trianglerighteq$ est réflexive, transitive and antisymétrique comme son composant strict $\triangleright$("est strictement supporté") est irréflexive, transitive et asymétrique, son composant d'indifférence $\equiv$("est aussi supporté que") est réflexive, symétrique et composé de (θ, θ) paires seulement, et sa relation d'incomparabilité associée $\bowtie$ ("est incomparable à") est irréflexive, non transitive et symétrique.

Dans l'**Exemple 1.**, Alice adore les fruits de mer, c'est sa première visite mais on lui a dit que les fruits de mer de ce restaurant ne sont pas frais. En outre, Alice croit, comme beaucoup le font, que le poisson devrait être jumelé avec du vin blanc et de la viande avec du vin rouge, mais elle ignore les autres combinaisons possibles. Pourtant, elle évite les jumelages avec l'eau parce qu'elle croit que boire de l'eau non embouteillée est généralement dangereux. Pour son dessert, elle croit que le gâteau associé à un vin blanc doux sera plus satisfaisant mais elle est confuse par la surcharge des alternatives. Les contextes de préférences évidentielles de Alice sont décrits dans la Table 1.

	c_1^p	c_2^p	c_3^p
S	$\{Plat\}$	$\{Plat, Boisson\}$	$\{Boisson, Dessert\}$
A	$\{p, v\}$	$\{(p, vb), (p, vr), p, e), (v, vb), (v, vr), (v, e)\}$	$\{(vb, g), (vb, gl), (vr, g), (vr, gl), (e, g), (e, gl)\}$
B	v :0.8 p :0.2	(p,vb) :0.5 (v,vr) :0.3 (p,vr),(v,vb) :0.2 (p,e),(v,e) :0.0	(vb,g) :0.4 (vr,gl) :0.3 (vr,g),(vb,gl) :0.2 $\varnothing$:0.1 (e,g),(e,gl) :0.0
$\trianglerighteq$	v$\triangleright$p	(p,vb)$\triangleright$(v,vr) (p,vb)$\equiv$ (p,vb)	(vb,g)$\bowtie$(vb,gl) [1]

Table 1: Les contextes de préférences évidentielles induits de l'**Exemple 1**.

Grâce à notre approche à deux niveaux, nous avons pu capturer tous les états épistémiques de l'agent vers ses préférences.

- Connaissance complète : prenons l'**Exemple 1.**, si le restaurant permet à ses clients de choisir leur fruits de mer à servir, Alice sera bien informée de l'alternative qu'elle désire. Ceci sera traduit par une fonction de masse de croyance certaine et précise :$(p : 1.0; v : 0.0)$.
- Ignorance partielle : Dans le contexte de préférence c_1^p, Alice a une fonction de masse de croyance incertaine mais précise concernant l'utilité des alternatives. Dans le contexte de préférence c_2^p, elle a une fonction incertaine et imprécise en liant les alternatives (p, vr) et (v, vb).
- Ignorance totale : Dans l'**Exemple 1.**, si le restaurant introduit de nouvelles spécialités de poisson et de viande qu'Alice n'a jamais goûtées, elle sera totalement ignorante des deux alternatives. Ceci sera traduit par une fonction de masse de croyance certaine et imprécise : $(p, v : 1.0)$.
- Hésitation : Au lieu d'exprimer son ignorance totale, Alice peut vouloir exprimer ses croyances sur ses préférences avec un certain degré d'hésitation représenté par la masse associée à l'ensemble vide comme dans c_3^p. Son hésitation est due à sa confusion par la surcharge des alternatives. Comme déjà mentionné dans la section 2.2.1, Nous considérons la masse positive de l'ensemble vide comme le conflit interne lors de l'expression des préférences.
- Support nul : l'agent n'a aucune évidence de croire qu'une alternative peut être en quelque sorte bonne ou mauvaise.

Étant donné la base de croyances et les préférences dérivées, l'ignorance peut être quantifiée en utilisant la mesure de non-spécificité définie à la section 3.2.5. Le degré d'hésitation correspond à la masse assignée à l'ensemble vide autorisée par l'hypothèse du monde ouvert. Enfin, la modélisation des croyances sur les préférences séparément du corpus de préférences finales, en plus de distinguer l'ignorance de l'hésitation et leur quantification, est important dans un large éventail d'applications. Par exemple, dans les systèmes de recommandation basés sur la connaissance, les préférences du client peuvent être stimulées en fonction de sa base de croyances grâce à de multiples interactions afin de réduire son ignorance et/ou son hésitation jusqu'à ce qu'il trouve son élément préféré.

1. (vb,g) est incomparable à (vb,gl) parce que nous ne connaissons pas la masse exacte associée à (vb,gl) car elle est liée à (vr,g) par manque d'évidence.

3.2 Dériver les préférences

Au moment où la base de croyances de l'agent est définie, ses relations de préférences sont dérivées comme un pré-ordre partiel $\succeq^B$ induit par les mesures BetP sur A, l'ensemble des alternatives :

$$\begin{aligned}\succeq^B = \{(a_1, a_2) | (BetP(a_1) \geq BetP(a_2) \\ \wedge (min(BetP(a_1), BetP(a_2)) > 0)\}\end{aligned} \quad (6)$$

La relation $\succeq^B$ est réflexive and transitive. Étant donné la relation de préférence évidentielle $\succeq^B$ et deux alternatives $a_1, a_2 \in A$, trois relations spécifiques entre a_1 et a_2 sont définies :

- a_1 est strictement préférée à a_2, notée $a_1 \succ^B a_2$, ssi $a_1 \succeq^B a_2$ et $a_2 \neg \succeq^B a_1$. $\succ^B$ est irréflexive, transitive and asymétrique.
- a_1 est indifférent à a_2 notée $a_1 \approx^B a_2$, ssi $a_1 \succeq^B a_2$ et $a_2 \succeq^B a_1$. $\approx^B$ est réflexive, transitive and symétrique.
- a_1 est incomparable à a_2, notée $a_1 \sim^B a_2$, ssi $a_1 \neg \succeq^B a_2$ et $a_2 \neg \succeq^B a_1$. $\sim^B$ est irréflexive, non transitive and symétrique.

Puisque la fonction de masse de croyance est définie sur les partitions de l'ensemble des alternatives et l'ensemble vide, chaque alternative est, exactement, supportée par l'une de ces partitions. Par conséquent, la mesure BetP est déterminée comme suit :

$$BetP(a) = \frac{1}{1 - m(\varnothing)} \frac{m(B)}{|B|} \; tel\ que,\ a \in B \subseteq A \quad (7)$$

les relations de préférence évidentielles R dérivées de la base de croyances B (voir Table 1) sont décrites dans la Table 2.

3.2.1 Croyances et Préférences

Quand le serveur demande à Alice : "Que préféreriez-vous : du poisson ou de la viande ?" elle préférera la viande au poisson si et seulement si elle croit que manger de la viande est mieux pour elle que manger du poisson même si elle désire du poisson. Si les croyances d'Alice sont suffisamment justifiées par l'évidence, elle

	c_1^p	c_2^p	c_3^p
R	v :0.8 p :0.2	(p,vb) :0.5 (v,vr) :0.3 (p,vr) :0.1 (v,vb) :0.1 (p,e) :0.0 (v,e) :0.0	(vb,g) :0.5 (vr,gl) :0.33 (vr,g) :0.085 (vb,gl) :0.085 (e,g) :0.0 (e,gl) :0.0
$\succeq^B$	v$\succ^B$p	(p,vb)$\succ^B$(v,vr) (p,vr)$\approx^B$ (v,vb)	(vb,g)$\sim^B$(e,g)

Table 2: Les relations de préférences dérivées pour l'**Exemple 1**.

sera satisfaite car elle a échappé à tomber malade à cause des poissons qui sont pas frais. Contrairement aux préférences primitives, dans tels scénarios réels, lorsqu'un agent n'a que des informations incomplètes et/ou incertaines sur les alternatives parmi lesquelles il choisit, les croyances comptent comme des raisons pour les préférences formant alors une base pour leur justification. Dz cz fait, les préférences sont le reflet des croyances :

— Préférence stricte ($a_1 \succ^B a_2$) : l'évidence détenue par l'agent fournit plus de support [2] pour a_1 que a_2.
— Indifférence ($a_1 \approx^B a_2$) : l'évidence de l'agent ne supporte pas a_1 plus fortement que a_2 et ne supporte pas a_2 plus fortement que a_1.
— Incomparabilité ($\sim^B y$) : L'agent n'a pas d'évidence (concernant a_1 ou a_2 ou les deux) pour comparer a_1 et a_2.

3.3 Préférences conflictuelles

En pratique, il y a souvent plusieurs relations de préférences qui doivent être considérées, chacune mettant l'accent sur une facette différente du problème traité. Dans notre modèle de préférence, nous traitons différents contextes de préférences, dans lesquels une variable peut être impliquée dans plus qu'un seul contexte. Un modèle de préférence est consistent si et seulement s'il n'a pas de contextes de préférences c_i^p et c_j^p tels que $a_1 \succ_i^B a_2$ et $a_2 \succ_j^B a_1$ pour toute alternative a_1 et a_2. Comme la relation de préférence n'est plus une question de oui/non, la notion de consistance devient aussi une question de degré. Dans notre modèle, nous supposons que plus deux relations de préférence sont distantes l'une de l'autre, plus elles sont en conflit. Ainsi, nous proposons de définir le conflit entre deux relations de préférence évidentielles basées sur les croyances R_i et R_j en utilisant une métrique L1 normalisée entre leurs distributions BetP respectives comme suit :

$$\begin{array}{ll} Conf(i,j) & \\ = \frac{\sum_{(a\in A)} |BetP_i^{S_i\downarrow S}(a) - BetP_j^{S_i\downarrow S}(a)|}{|A|} & \text{si } S \neq \varnothing \\ = 0 & \text{si } S = \varnothing. \end{array} \quad (8)$$

Tel que : $S = S_i \cap S_j$; A = ensemble d'alternatives associé à S ; $|A|$: Cardinalité de A et $BetP_i^{S_i\downarrow S}(a) = max_{a_i \in A_i : a_i^{\downarrow S} = a} BetP(a_i)$.

— Si $Conf(i,j) = 0$, les relations de préférence R_i et R_j sont totalement concordants ;
— Si $0 < Conf(i,j) < 1$, les relations de préférence R_i et R_j sont partiellement conflictuelles ;
— Si $Conf(i,j) = 1$, les relations de préférence R_i et R_j sont totalement conflictuelles.

Prenons l'**Exemple 1.** : Conf(1,2)=0.4 ; Conf(1,3)=0 ; Conf(2,3)=0.01.

La consistance est le dual du conflit :

$$Consistance(\wp) = 1 - max(Conf(i,j)) \quad (9)$$

Dans notre **Exemple 1.**, $Consistance(\wp) = 1 - 0.4 = 0.6$, donc $\wp$ est partiellement consistant. Dans notre approche, dès que deux relations de préférence sont totalement conflictuelles, le modèle de préférence est totalement inconsistant, c'est à dire, qui n'a pas de solution. L'évaluation des conflits entre deux relations de préférences peut servir à la détection précoce de l'inconsistance du modèle, ce ce qui apporte des économies de coûts et de temps. En outre, nous pouvons quantifier à quel degré une relation de préférence donnée R_i, dans un modèle de préférence $\wp$ avec n relations de préférence, est en conflit avec les autres (n-1) relations de préférence comme suit :

$$Conf(i,\wp) = \frac{1}{n-1} \sum_{j=1, i\neq j}^{n} Conf(i,j) \quad (10)$$

Dans l'**Exemple 1.** : Conf(1, $\wp$)=0.2 ; Conf(2, $\wp$)=0.205 ; Conf(3, $\wp$)=0.005.

3.4 Préférences distordues

Dans certains cas, les préférences réelles de l'agent peuvent être distordues pour diverses raisons. Par conséquent, la fiabilité de la source fournissant les préférences devrait être évaluée. En dépit de son importance pour la prise de décision, l'entrelacement des préférences et de la fiabilité est plutôt inexploré dans la littérature de l'IA. Lorsque nous pouvons quantifier la mesure dans laquelle les préférences fournies reflètent les préférences réelles, nous pouvons réviser la relation de préférence fournie en affaiblissant sa distribution

2. Le terme support désigne une masse de croyance non nulle ; sinon, nous ne pouvons pas nous référer à une masse nulle en tant que support.

BetP correspondante en utilisant la règle d'affaiblissement de la section 3.2.4. Étant donné $\alpha \in [0,1]$, l'intensité de vérité de R, indiquant la fiabilité de sa source, la mesure BetP affaiblie est définie comme suit :

$$BetP^{\alpha}(a) = \frac{1}{1-\alpha.m(\varnothing)} \left[\frac{\alpha.m(B)}{|B|} + \frac{1-\alpha}{|A|} \right], \quad \forall a \in B \subset A \tag{11}$$

— Si $\alpha = 1$, les préférences fournies correspondent aux réelles donc l'affaiblissement n'affecte pas la relation de préférence de sorte que $BetP^{\alpha}(a) = BetP(a)$.
— Si $\alpha = 0$, les préférences fournies sont totalement distordues donc l'affaiblissement induit le cas d'indifférence totale où toutes les informations fournies par R sont abandonnées.

Comme on peut le constater, la procédure d'affaiblissement vise à affaiblir les relations de préférence stricte et d'incomparabilité en faveur de la relation d'indifférence. Cependant, si nous avons déjà une situation d'indifférence totale, l'affaiblissement n'aura aucune influence. Pour cette raison, dans l'équation 14, nous avons exclu le cas où les alternatives sont supportées par tout l'ensemble des alternatives (la masse de A). D'un autre côté, considérer les intensités de vérité peut réduire le conflit global entre les préférences puisqu'elles sont utilisées pour affaiblir les moins fiables et renforcer ainsi la consistance du problème.

Prenons l'**Exemple 1.**. Le restaurant prévoit de lancer un nouveau service pour offrir des paniers-repas aux employés des bureaux adjacents. Ainsi, ils veulent décider quelles conceptions des paniers ils vont opter en se basant sur les préférences de leurs clients. Afin de faire de ce nouveau service un succès, ils ne veulent pas appuyer leur étude sur des sources de préférences non fiables. Par conséquent, ils décident d'associer à chaque préférence fournie une intensité de vérité. Alice n'est pas une cliente fréquente, donc aucune de ses préférences fournies n'est considérée comme totalement vraie (voir Table 3).

	c_1^p	c_2^p	c_3^p
α	0.2	0.7	0.4
R^{α}	v :0.56 p :0.44	(p,vb) :0.4 (v,vr) :0.26 (p,vr) :0.12 (v,vb) :0.12 (p,e) :0.05 (v,e) :0.05	(vb,g) :0.26 (vr,gl) :0.22 (vr,g) :0.14 (vb,gl) :0.14 (e,g) :0.12 (e,gl) :0.12

Table 3: Les relations de préférences affaiblies pour l'**Exemple 1**.

Si nous recalculons les degrés de conflit selon les relations de préférence affaiblies, nous aurons : Conf(1,2)=0.17 ; Conf(1,3)=0 ; Conf(2,3)= 0.11. Alors que, la consistance du modèle de préférence augmente à 0,83.

Enfin, il est nécessaire de prendre en compte cette méta-connaissance concernant les préférences, notamment pour les problèmes de décision reposant sur les préférences des agents tels que la configuration des produits et des services, les enchères multi-articles, etc. Au lieu de se méfier ou de se fier aux préférences fournies, peu importe à quel point elles sont distordues, on peut vouloir évaluer leur intensité de vérité afin de décider si nous devons compter sur ces préférences. Dans certaines applications critiques, les décisions ne doivent dépendre que de préférences non distordues. Dans d'autres applications, nous pouvons tolérer la distorsion dans une certaine mesure. Sinon, l'évaluation de la fiabilité peut aider les décideurs à prendre des mesures pour tenir les agents bien informés et corriger les croyances erronées. Bien que cela dépasse le cadre de cet article, dans notre modèle, le degré de fiabilité de la source peut être estimé à partir du degré de conflit avec les autres préférences, à partir le degré d'hésitation ou le degré d'ignorance.

4 Un algorithme de Séparation et Évaluation Dirigé par Les Préférences (SEDP

4.1 Aggregation des Préférences

Résoudre un problème de préférence consiste à trouver une instanciation complète ω de toutes les variables du modèle, si elle existe, qui satisfait toutes les préférences de l'agent. Cependant, comme la préférence n'est plus une question de type oui ou non, la notion de satisfaction de préférence devient ainsi une question de degré. Soit R_i une relation de préférence évidentielle basé sur les croyances, définie sur un ensemble de variables S_i, et soit $\delta_{(i,a)} = BetP_i(a)$ le degré local de satisfaction local de R_i par une alternative donnée $a \in A_i$, R_i est dite satisfaite par a, noté $a \vDash R_i$, ssi $\delta_{(i,a)} > 0$. Le degré global de satisfaction d'un ensemble de préférences, dans un modèle de préférence $\wp$ défini sur un ensemble de variables X, par une instanciation donnée ω est ensuite obtenu en combinant les degrés locaux de satisfaction comme suit :

$$\delta_{(\wp,\omega)} = \prod_{R_i \in \wp, R_i^{\uparrow X}} \delta(i,a) \tag{12}$$

Toute instanciation satisfaisant l'ensemble de préférences à un degré global supérieur à 0 (c'est-à-dire qu'aucune préférence n'est falsifiée) est considérée comme une solution au problème en question. Le degré global de satisfaction induit un pré-ordre total sur

l'ensemble des solutions, de sorte que la solution optimale ω^* sera celle qui satisfait au maximum l'ensemble des préférences :

$$\delta_{(\wp,\omega^*)} = max(\delta(\wp,\omega)) \quad (13)$$

D'autres méthodes peuvent être employées pour combiner les différentes relations de préférence en utilisant une autre t-norme telle que Min. Cependant, une telle Min-combinaison égalitaire ne discrimine pas les solutions satisfaisant les préférences au même degré car elle n'est concernée que par le degré de satisfaction de la préférence la moins satisfaite, quelle que soit les degrés de satisfaction des autres préférences. L'opérateur produit employé est commutatif, associatif, 1 est son élément identité et 0 est son élément absorbant, et il est monotone par rapport à la satisfaction des préférences individuelles, c'est-à-dire, $\delta_{(i,a)} \leqslant \delta_{(i,a')}$ implique $\delta_{(i,a)}.\delta_{(j,a'')} \leqslant \delta_{(i,a')}.\delta_{(j,a'')}$.

Étant donné un modèle de préférence, de nombreuses tâches peuvent être effectuées. La première et la plus intuitive tâche à effectuer peut être de vérifier si le modèle est consistant. Si un modèle est consistant, nous pourrions être intéressés à trouver une solution, toutes les solutions, toutes les solutions satisfaisant le modèle à un certain degré ou simplement trouver la solution optimale.

4.2 Algorithme SEDP

Généralement, pour les problèmes d'optimisation, l'algorithme de Séparation et Évaluation (SE) est le plus adapté. Il construit, de façon incrémentale, les instanciations prospectées pour être des solutions, où il abandonne tôt toute instanciation partielle ne pouvant pas être étendu pour construire une solution, soit parce qu'elle falsifie certaines préférences, soit parce qu'elle ne peut aboutir à une meilleure solution que celles déjà trouvées.

Comme la relation de préférence est l'élément focal de notre approche, nous nous intéressons à l'algorithme CDBT (Constraint Directed BackTracking) proposé dans [8] qui recherche à instancier les variables non à partir de leurs domaines associés mais à partir des relations de contraintes qu'elles impliquent. Ainsi, un nœud dans l'arbre de recherche ne sera pas une seule valeur d'une variable mais une instanciation d'un ensemble de variables couvertes par une contrainte donnée. Il a été montré que CDBT a un espace de recherche plus limité que l'espace de recherche du BT classique. De ce fait, Nous proposons d'étendre le CDBT à un algorithme de Séparation et Évaluation dirigé par les Préférences (SEDP) basé sur l'algorithme SE introduit dans [4] pour les Max-CSP.

Initialement, le SEDP sélectionne m relations de préférence, à partir d'un ensemble de n préférences, qui couvrent toutes les variables du problème, notées R_c. Dans notre **Exemple 1.**, les relations de préférence sélectionnées seront R_1 et R_2. Une borne supérieure **B** est fixée au plus petit degré global de satisfaction toléré afin de rejeter chaque solution donnant un degré de satisfaction $\leq$**B**. Après cela, à chaque niveau, une relation de préférence est explorée en essayant chaque alternative à partir de son ensemble d'alternatives associé. La solution partielle actuelle est ensuite étendue à cette alternative de sorte que les variables impliquées par la relation de préférence, non couvertes précédemment, soient instanciées. Un degré exact de satisfaction de toutes les préférences couvrant les variables instanciées est calculé et combiné à une approximation optimiste du degré de satisfaction du reste des préférences, aboutissant à une borne inférieure **b** de sorte que chaque relation de préférence, dont les variables sont couvertes, est supprimé de l'ensemble de préférences. Si **b** > **B**, l'algorithme passe au niveau suivant, sinon, le retour en arrière se produit et le sous-arbre du nœud courant est élagué. Si toutes les variables du problème sont instanciées, une nouvelle solution est trouvée avec un degré de satisfaction strictement supérieur à **B**, ainsi la solution est affichée et la limite supérieure **B** est mise à jour. L'algorithme se termine quand aucune meilleure solution ne peut être trouvée (voir Algorithme 1). La borne inférieure **b** d'une solution partielle est calculée comme suit :

$$\delta^i_{(\wp,\omega)} = \delta_{(i,a)}. \prod_{R_j \in \wp, R_j^{\uparrow X}} (max(\delta_{(j,a')})) \quad (14)$$

De plus, en appliquant l'heuristique "l'alternative donnant le meilleur degré de satisfaction (max-BetP) est sélectionné en premier", nous nous assurons que la solution optimale est construite tôt et que l'espace de recherche est réduit. Nous illustrons, dans la Figure 1, l'exécution de SEDP pour résoudre le modèle de préférence décrit dans l'**Exemple 1** de la section 4.2. Dans la Figure 1, les instanciations ayant un (X)rouge

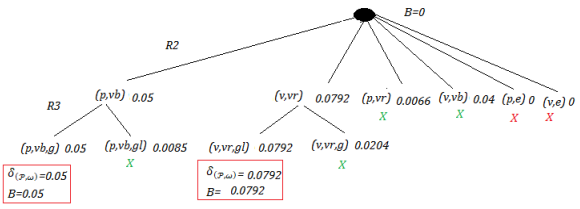


Figure 1: SEDP pour le problème décrit dans l'**Exemple 1**.

sont rejetés parce qu'elles falsifient au moins une préférence, cependant, les instanciations avec un (X) vert sont avortés parce qu'elles ne peuvent pas conduire à une meilleure solution que ceux déjà trouvés selon le

Algorithme 1 : Algorithme SEDP

Entrées : $(X, R_c, S, \omega, B, b)$
Sorties : (ω^*, B)

tant que $R_c \neq \varnothing$ **faire**
  sélectionne et supprime une relation $R_i \in R_c$;
  $S \leftarrow S \cup S_i$;
  tant que $A_i \neq \varnothing$ **faire**
    sélectionne et supprime la meilleure alternative $a \in A_i$;
    $\omega \leftarrow \omega \cup a$;
    Calcule une borne inférieure b pour ω;
    si $b > B$ **alors**
      si $S = X$ **alors**
        $B \leftarrow b$;
        $\omega^* \leftarrow \omega$;
        Affiche (ω^*, B);
        si $B = 1$ **alors**
          retourne "Fini";
      sinon
        SEDP$(X, R_c, S, \omega, B, b)$;
    sinon
      SEDP$(X, R_c, S, \omega - a, B, b)$;

degré de satisfaction estimé. Pour l'**Exemple 1.**, le meilleur menu pour Alice est {Plat principal :viande ; Boisson :vin rouge ; Dessert :glace}.

Il a été prouvé dans [8] que la taille de l'arbre de recherche donné par l'algorithme CDBT est inférieure à la taille de l'arbre BT standard. La même chose est prouvée pour notre algorithme SEDP contre l'algorithme SE. Cependant, comme SEDP et SE sont tous les deux des algorithmes basés sur BT, la complexité au pire des cas des deux algorithmes est exponentielle avec la taille du problème.

5 Expérimentation

L'algorithme SEDP a été implémenté en Java et testé contre l'algorithme SE classique (dirigé par les domaines) sur des problèmes générés aléatoirement avec des relations de préférences binaires, c'est-à-dire que les relations de préférence impliquent au plus deux variables.

Nous avons employé un générateur modèle B $< n, m, p1, p2, i >$, tels que "n" est le nombre de variables ; "m" est la taille uniforme des domaines ; "p1" est la pourcentage des relations de préférences unaires et binaires dans le problème par rapport au nombre total des relations unaires (n) et binaires (n(n-1)/2) possibles qui peuvent être définies sur "n" variables de sorte que le nombre total de préférence sera (p1 n(n+1)/2) ; "p2" est le pourcentage des alternatives avec un $BetP > 0$ (i.e., masse > 0 dans la base de croyances) pour chaque relation de préférence par rapport au nombre total des alternatives (m) pour les préférences unaires et (m^2) pour celles binaires ; le dernier paramètre "i" est spécifique à notre approche, c'est une mesure de l'imprécision (non spécificité) de la base de croyances, c'est le pourcentage de partitions de l'ensemble des alternatives de chaque relation de préférence dans la base de croyances, tel que (i = 0) correspond à l'état de précision totale où tous les éléments supportés dans base sont des singletons, nous avons (m) partitions pour les relations de préférences unaires et (m^2) partitions pour celles binaires, alors que (i = 1) correspond à l'état d'imprécision totale (ignorance totale) où l'élément unique et totalement supporté dans la base de croyances est l'ensemble entier des alternatives. Dans cet article, nous rapportons les résultats pour la classe de problèmes $< 25, 3, 0.064, 5/9, 0 >$ pour lesquels une étude préliminaire a montré que 60% de ses instances ont des solutions. Dans la Figure 2, le nombre moyen de nœuds visités (axe vertical) pour trouver la première solution, la meilleure solution et pour prouver qu'il n'y a pas de meilleure solution (axe horizontal) est enregistré pour 100 instances de cette classe. La figure 2, montre que SEDP est moins coûteux que SE classique. Pour la même classe de problème, nous rapportons l'impact de l'imprécision sur le coût de recherche pour le SEDP dans la Figure 3. Nous pouvons remarquer que le coût de la recherche (axe vertical) diminue à mesure que l'imprécision (axe horizontal) augmente. En fait, une imprécision plus élevée signifie que la relation d'indifférence est étendue sur plus d'alternatives, par conséquent, le solveur, après avoir trouvé la solution optimale, ne va pas loin pour prouver qu'il n'y a pas de meilleure solution.

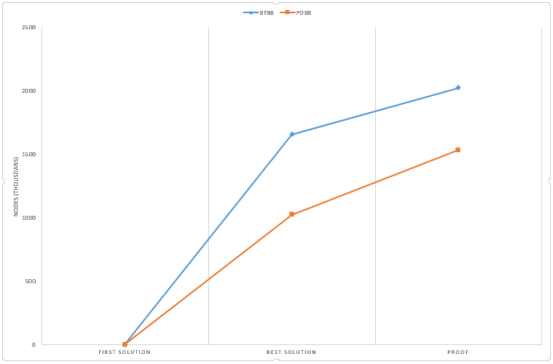


Figure 2: Coût de recherche pour obtenir la première et la meilleure solution pour $< 25, 3, 0.064, 5/9, 0 >$.

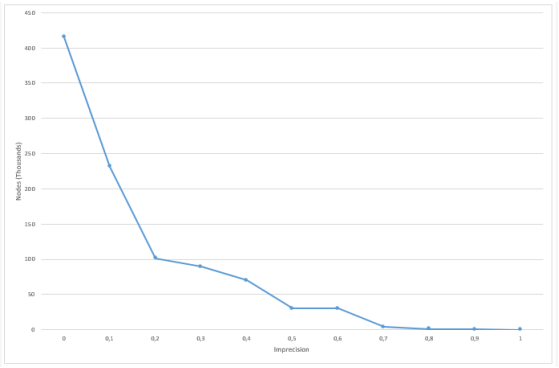


Figure 3: Coût de recherche pour obtenir la meilleure solution pour $< 25, 3, 0.064, 5/9, i >$.

6 Conclusion et Perspectives

Dans cet article, nous avons introduit une nouvelle approche pour montrer comment les préférences d'un agent peuvent être dérivées de ses croyances dans une perspective à deux niveaux lorsque seulement des informations partielles et/ou incertaines sont disponibles. En outre, nous avons abordé deux autres problématiques plus complexes, à savoir, les préférences conflictuelles et distordues. L'espace ne nous a pas permis de discuter dans cet article d'autres méthodes et heuristiques de résolution. Cependant, nous avons montré que les techniques de résolution des CSP classiques peuvent être facilement adaptées au modèle de préférence proposé. En outre, l'algorithme SEDP pourrait être amélioré en introduisant des heuristiques pour l'ordre d'exploration des préférences en se basant sur les mesures de conflit introduites qui peut encore réduire le coût de recherche. D'autres axes de recherche visent à exploiter l'expressivité offerte par le modèle de préférences évidentielles afin d'élargir la portée des problèmes qui peuvent être abordés, comme les préférences prioritaires et la dynamique des préférence en utilisant le processus de révision des croyances. Nous pouvons également aborder les préférences bipolaires exploitant les notions de croyances négatives et positives. Nous avons également l'intention d'introduire la relation de préférence faible en utilisant des seuils. Enfin, nous prévoyons d'explorer comment notre modèle peut être utilisé dans les applications d'aide à la décision comme les systèmes de recommandation.

Références

[1] A. Ayoun and P. Smets (2001). Data association in multi-target detection using the transferable belief model. *International Journal of Intelligent Systems*. 16(10) : 1167–1182.

[2] S. Bistarelli, H. Fargier, U. Montanari, F. Rossi, T. Schiex and G. Verfaillie (1999). Semiring-based CSPs and valued CSPs : Frameworks, properties and comparison. *Constraints*, 4 : 199–240.

[3] A. P. Dempster (1967). Upper and lower probabilities induced by a multivalued mapping. *The Annals of Mathematical Statistics*, 38(2) : 325-339.

[4] E. Freuder and R. Wallace (1992). Partial constraint satisfaction. Artificial Intelligence, 58 :21–70.

[5] M. Gelain, M. S. Pini, F. Rossi and K. B. Venable (2007). Dealing with Incomplete Preferences in Soft Constraint Problems. *In Proceedings of CP 2007* : 286-300.

[6] M. Gelain, M. S. Pini, F. Rossi, K. B. Venable and N. Wilson (2010). Interval-valued soft constraint problems. *Ann. Math. Artif. Intell.* 58(3-4) : 261-298.

[7] S. Kaci. Working with Preferences : Less Is More. Springer.

[8] W. Pang and S. Goodwin (1997). Constraint directed backtracking. *Advanced Topics in AI.* 1342 : 47-56. Springer Verlag.

[9] J. Lang and L. Van der Torre (2010). Preference change triggered by belief change : A principled approach. *G. Bonanno, B. Löwe, W. van der Hoek (Eds.), Logic and the Foundations of Game and Decision Theory (LOFT 8), Lecture Notes in Computer Science.* 6006 : 86-111, Springer.

[10] F. Liu (2008). Changing for the better : Preference dynamics and agent diversity. *Ph.D. dissertation*, University of Amsterdam.

[11] M. S. Pini and F. Rossi (2005). Uncertainty in Soft Constraint Problems. *In Proceedings of CP 2005*,3709 : 865.

[12] G. Shafer (1976). A Mathematical Theory of Evidence. *Princeton University Press*, Princeton, NJ.

[13] F. Smarandache, A. Martin and C. Osswald (2011). Contradiction measures and specificity degrees of basic belief assignments. *International Conference on Information Fusion.* Chicago, United States.

[14] P. Smets and R. Kennes (1994). The Transferable Belief Model. *Artifical Intelligence*, 66 : 191–234.

Extension de Compact-Table aux Tables Simplement Intelligentes

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

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

Résumé

Ces dernières années, plusieurs algorithmes pour la contrainte table ont été proposés pour assurer la propriété de cohérence d'arc généralisée (GAC). Compact-Table (CT) [1] est un algorithme récent de l'état-de-l'art, que nous avons étendu dans notre article [2], intitulé "Extending Compact-Table to Basic Smart Tables" et publié à CP-17, aux tables simplement intelligentes (contenant des éléments tels que $= v$, $*$, $\neq v$, $\geq v$, $> v$, $< v$,$\leq v$, $\in S$ et $\notin S$, où v est une valeur et S un ensemble de valeurs). Nos expérimentations montrent une amélioration du temps de résolution lorsqu'il y a compression.

1 Introduction

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

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

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

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

Dans cet article, nous étendons l'algorithme Compact-Table, initialement introduit pour les tables positives (chaque tuple est une solution possible) sans aucune compression [1], afin de pouvoir gérer les tables simplement intelligentes. Ces tables sont des tables positives pouvant contenir dans une même table :

- l'assignation à une valeur unique $= v$ (initialement accepté par Compact-Table)
- la valeur universelle $*$ (déjà supporté dans une première version étendue CT* [3])
- la simple restriction $\neq v$
- les restrictions de bornes $\geq v$, $> v$, $< v$ et $\leq v$
- les restrictions d'ensembles $\in S$ et $\notin S$

2 Modification introduites

2.1 Gérer les $\neq v$

En partant de CT*, il est trivial d'ajouter la possibilité d'intégrer les $\neq v$. Lorsqu'il reste deux valeurs ou plus dans le domaine, il est trivial de voir qu'au moins une valeur rend le tuple valide (du point de vue de la variable considérée). Dans ce cas, le tuple doit donc rester valide comme dans le cas d'une $*$. Dans le cas limite où une seule valeur est présente dans le domaine, le bitset `support` est celui utilisé. En remplissant les `supports`* avec 0 pour le bit correspondant (comme pour les $*$) et les `supports` avec 1 pour tous les supports associés à une valeur autre que v, la propagation correcte est assurée.

2.2 Gérer les $\geq v$, $> v$, $< v$ et $\leq v$

Une variable x respecte la contrainte $\geq v$ (ou $> v$ qui est trivialement équivalent à $\geq v+1$ dans le cas de variables entières) si sa valeur maximale est plus grande ou égale à v (ou strictement plus grande que v). Pour aider à la vérification, nous avons donc ajouté de nouveaux bitsets, appelés `supportMax`, associés à chacune des valeurs du domaine. Le remplissage de ces bitsets se fait en supposant que la valeur associée est la valeur maximale du domaine : pour toutes les valeurs respectant $\geq v$ (ou $> v$), le bit correspondant est mis à 1, 0 dans l'autre cas. Les bits associées à un élément autre que $\geq v$ et $> v$ sont tous mis à 1. Lorsqu'une mise à jour de type *classique* est lancée et que le maximum à changé, une intersection entre `currTable` et le bitset correspondant au nouveau maximum est effectué, retirant ainsi les tuples contenant un $\geq v$ ou un $> v$ qui ne sont plus valide.

Gérer les $\leq v$ et $< v$ a été fait suivant le même principe avec introduction de bitsets appelés `supportMin`.

2.3 Gérer les $\in S$ et $\notin S$

Dû au nombre exponentiel d'ensembles différents pouvant se trouver dans une même table, la seule solution viable est l'application systématique d'une mise à jour de type *reset* pour les variables auxquelles une restriction d'ensemble est appliquée.

3 Algorithme de compression

Le problème de la compression optimale étant NP-complet, nous avons créé un algorithme de type glouton récursif introduisant des $\leq v$ et $\geq v$ (avec post-traitement pour ajouter des $\neq v$ et $*$). A chaque étape, l'algorithme prend le sous-ensemble des tuples contenant N éléments de compression et essaye d'en extraire un maximum de tuples avec $N+1$ éléments de compression. Nous avons pu observer la compression des différentes tables présentes dans nos différents bancs d'essai Fig.1 (ratio de compression en fonction du nombre de tuple initial) et en observer la diversité. Cela varie entre aucune compression (bancs d'essais Kakuro ou Nonogram) à une compression importante (PigeonsPlus).

4 Résultats

Les résultats obtenus (Fig.2) sur des instances variées ayant plus de 5% de compression nous montre que l'utilisation de CT^{bs}est compétitif. Lorsque la compression est moindre, il y peut y avoir un léger ralentissement dû aux opérations supplémentaires.

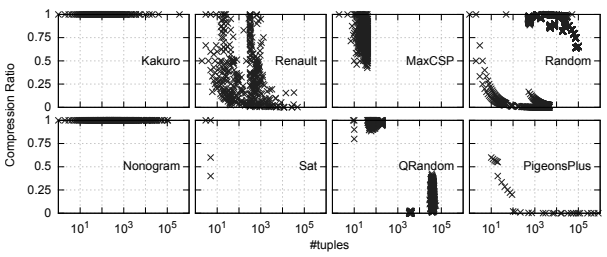


FIGURE 1 – Distribution du ratio de compression sur 8 séries d'instances

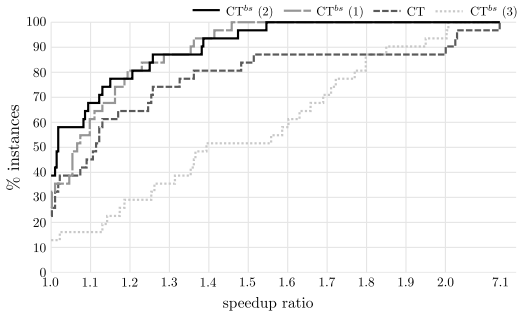


FIGURE 2 – Profile de performance comparant CT^{bs}et CT

5 Conclusion

Nous avons proposé une extension de Compact-Table (CT^{bs}) permettant de gérer les tables simplement intelligentes de manière efficace lorsqu'il y a une compression significative (à partir de 5% de tuples en moins). Nous avons également proposé un algorithme glouton pour la compression de table en table simplement intelligentes. Pour plus de détails concernant l'implémentation et les résultats, n'hésitez pas à lire l'article originel.

Références

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

[2] Hélène Verhaeghe, Christophe Lecoutre, Yves Deville, and Pierre Schaus. Extending compact-table to basic smart tables. In *Proceedings of CP'17*, pages 297–307, 2017.

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

Stratégies de sélection de sous-domaines pour les systèmes de contraintes sur les flottants *

Heytem Zitoun[1] †**Claude Michel**[1] **Michel Rueher**[1] **Laurent Michel**[2]

[1] Université Côte d'Azur, CNRS, I3S, France
[2] University of Connecticut, Storrs, CT 06269-2155
prenom.nom@unice.fr ldm@engr.uconn.edu

Résumé

La vérification de programmes est un enjeu clé pour les applications critiques telles que l'aviation, l'aérospatiale, ou les systèmes embarqués. Les approches basées sur le « bounded model checking (BMC) » et la programmation par contraintes (CBMC, CBPV, ...) recherchent des contre-exemples qui violent une propriété du programme. La recherche de tels contre-exemples peut-être particulièrement longue et coûteuse lorsque le programme à vérifier contient des opérations sur les flottants. En effet, les stratégies de recherche existantes en programmation par contraintes ont été conçues pour les domaines discrets et, dans une moindre mesure, les domaines continus. Dans un précédent article, nous avons introduit un ensemble de stratégies de choix de variables qui tirent parti des spécificités des flottants, e.g, la densité du domaine, les phénomènes de cancellation et d'absorption. Dans cet article, nous présentons de nouvelles stratégies de sélection de sous-domaines ciblant les domaines impliqués dans l'absorption et utilisant des techniques dérivées de consistances fortes. Les expériences préliminaires sur un ensemble important de benchmarks sont particulièrement prometteuses.

Abstract

Program verification is a key issue for critical applications such as aviation, aerospace, or embedded systems. Bounded model checking (BMC) and constraint programming (CBMC, CBPV, ...) approaches are based on counter-examples that violate a property of the program to verify. Searching for such counter-examples can be very long and costly when the programs to check contains floating point computations. This stems from the fact that existing search strategies have been designed for discrete domains and, to a lesser extent, continuous domains. In [26], we have introduced a set of variable choice strategies that take advantages of the specificities of the floats, e.g., domain density, cancellation and absorption phenomena. In this paper we introduce new sub-domain sélection strategies targeting domains involved in absorption and using techniques derived from higher order consistencies. Preliminary experiments on a significant set of benchmarks are very promising.

1 Introduction

Les programmes avec des calculs sur les flottants contrôlent des systèmes complexes et critiques dans de nombreux domaines, notamment l'automobile et d'autres systèmes de transport, l'énergie nucléaire ou encore dans les dispositifs médicaux. Les calculs sur les flottants sont dérivés de modèles mathématiques sur les réels [11], mais les calculs sur les nombres à virgule flottante sont différents des calculs sur les réels. Par exemple, avec les nombres flottants, certains réels ne sont pas représentable (e.g., 0.1 n'a pas de représentation exacte dans les flottants binaires). Sur les flottants, les opérateurs arithmétiques ne sont ni associatifs, ni distributifs, et peuvent être sujet à des phénomènes tels que l'absorption et la cancellation. De plus, le comportement des programmes contenant des calculs en virgule flottante varie selon le langage de programmation, le compilateur, le système d'exploitation ou l'architecture matérielle.

La Figure 1 illustre un programme simple où des erreurs de calcul dues à l'arithmétique spécifique des flottants ($\mathbb{F}$) entrainent un résultat sensiblement différent du résultat attendu sur les réels ($\mathbb{R}$). En interprétant le programme sur les réels, l'instruction `doThenPart` devrait être exécutée. Cependant, un problème d'absorption sur les flottants (la valeur 1 est absorbée par

*Ces travaux ont été partiellement supportés par l'ANR COVERIF (ANR-15-CE25-0002).

†Papier doctorant : Heytem Zitoun[1] est auteur principal.

```
void foo(){
  float a = 1e8f;
  float b = 1.0f;
  float c = -1e8f;
  float r = a + b + c;
  if(r >= 1.0f){
    doThenPart();
  } else {
    doElsePart();
  }
}
```

FIGURE 1 – Exemple motivant

`1e8f` [1]) conduit le programme à exécuter la branche `else`.

Cet écart de comportement des programmes peut avoir de lourdes conséquences si, par exemple, elle conditionne le freinage (ou non) dans un système ABS.

Pour toutes ces raisons, les calculs en virgule flottante sont une source supplémentaire d'erreurs dans les programmes embarqués. Fr plus, la majorité des programmes soient écrits par des programmeurs qui ont la sémantique des nombres réels à l'esprit. C'est pourquoi il est essentiel de s'assurer que certaines propriétés ne peuvent pas être violées à cause des erreurs arithmétiques sur les flottants. Par conséquent, l'identification de contre-exemples est une question critique dans la vérification des programmes embarqués.

La programmation par contraintes [5, 6] a été utilisée pour traiter de tels problèmes, mais la recherche de contre-exemples reste longue et coûteuse. Les stratégies de recherche standards sont en effet peu efficaces pour trouver de tels contre-exemples. De nombreuses stratégies de recherche ont été proposées sur les entiers [2, 10, 18, 20, 23] et, dans une moindre mesure, sur les réels [15, 14]. Or, les stratégies propres aux entiers et aux réels sont mal adaptées aux flottants. Le sous-ensemble des entiers d'un domaine borné par deux entiers est fini et réparti de manière uniforme ; il est donc possible d'en énumérer toutes les valeurs lorsque le domaine est suffisamment petit. Le sous-ensemble des réels borné par deux flottants étant infini et uniforme, il n'est pas énumérable. Aussi, les stratégies de recherche adaptées au continu utilisent des techniques sur les intervalles telles que la bissection ainsi que des propriétés mathématiques pour prouver l'existence ou l'absence de solution dans un petit intervalle. A contrario, l'ensemble des flottants est lui fini, mais sa cardinalité est très élevée et la répartition des flottants n'est pas du tout uniforme : la moitié des flottants sont dans le domaine [-1,1]. Les techniques précédentes, comme l'énumération, sont donc inenvisageables dans le cas général. Les flottants correspondent à une approximation des réels, mais n'héritent pas des mêmes propriétés mathématiques, telles que la continuité. Il est donc difficile de tirer profit des stratégies de recherches basées sur les propriétés des réels.

Dans [26], nous avons introduit un ensemble de *stratégies de sélection de variables* basées sur des propriétés spécifiques des flottants telles que la densité du domaine, les phénomènes de cancellation et d'absorption. Les stratégies de recherche qui en résultent sont beaucoup plus efficaces, mais ne permettent pas vraiment le passage à l'échelle et la résolution de benchmarks plus difficiles et réalistes. En effet, comme dans d'autres applications de techniques de contrainte, les solveurs efficaces nécessitent non seulement des stratégies de sélection de variables appropriées, mais aussi des stratégies de sélection de valeurs pertinentes. Ainsi, cet article se concentre sur les *stratégies de sélection de valeur* pour les solveurs de contraintes sur les flottants dédiés à la recherche de contre-exemples dans les applications de vérification de programme.

Les stratégies standards de sélection des valeurs sur les flottants sont dérivées de *stratégies de sélection de sous-domaines* utilisées sur les réels ; les sous-domaines étant générés en utilisant diverses techniques de coupe, eg, $x \leq v$ ou $x > v$ avec $v = \frac{\underline{x}+\overline{x}}{2}$.

Dans cet article, nous présentons deux nouvelles stratégies de sélection de sous-domaines. La première, exploite les phénomènes d'absorption, alors que la seconde reprend des idées dérivées des consistance forte.

Nous avons évalué ces nouvelles stratégies de sélection de sous-domaines sur un ensemble significatif de benchmarks provenant de la vérification de programme. Nous avons implanté plus de 200 stratégies de recherche qui sont des combinaisons de stratégies de sélection de variables précédemment introduites, de stratégies de sélection de sous-domaines présentées dans les pages suivantes et de variations de différents critères comme le filtrage.

Aucune de ces stratégies ne résout tous les benchmarks en un temps raisonnable [2]. Deux approches sont présentées pour remédier à ce problème : un portfolio de stratégies et une stratégie de recherche adaptative. En raison de la quantité de stratégies, une approche portfolio n'est ni particulièrement attrayante ni élégante, et plus difficile à implanter que la stratégie de recherche adaptative proposée et qui est basée sur un seul critère. Cette dernière approche est simple, élégante, facile à mettre en œuvre et performante. Toutes les stratégies ont été implantées dans Objective-CP, l'outil d'optimisation introduit dans [25].

En résumé, les contributions sont des *nouvelles stra-*

1. sur les flottants simple précision et avec un arrondi au plus près.

2. Un timeout d'une minute.

tégies de sélection de sous-domaines dédiées au système de contraintes sur les flottants, et une *stratégie de recherche adaptative* qui résout un ensemble significatif de benchmarks.

Le reste de cet article est organisé comme suit. La section 2 présente quelques notations et définitions nécessaires à la compréhension de cet article. La section 3 fournit un bref rappel des stratégies présentées dans [26]. La section 4 explique les nouvelles stratégies de sélection de sous-domaines que nous proposons. La section 5 est consacrée à l'analyse des résultats expérimentaux. Enfin, la section 6 traite des travaux en cours et des perspectives.

2 Notations et définitions

2.1 Les nombres à virgule flottante

Les nombres à virgule flottante ont été introduits pour approximer des nombres réels dans un ordinateur. La norme IEEE 754 [13] spécifie la représentation, ainsi qu'un ensemble de propriétés arithmétiques spécifiques aux flottants. Les deux principaux formats de représentations des flottants introduits dans IEEE 754 sont les formats *simple* et *double*. Selon le format utilisé, le nombre de bits de représentation est de 32 bits pour les flottants à simple précision, et de 64 bits pour les flottants à double précision. Un nombre à virgule flottante est composé de trois parties : le **signe** s (0 ou 1), la **mantisse** `m` et **l'exposant** `e` [11]. Un flottant normalisé est représenté par :

$$(-1)^s 1.m \times 2^e$$

Pour représenter les nombres flottants ayant une valeur absolue très petite, la norme IEEE 754 introduit la notion de nombre dénormalisé. Un flottant dénormalisé est représenté par :

$$(-1)^s 0.m \times 2^0$$

La **mantisse** `m` est représentée par 23 bits pour les flottants à simple précision, et par 52 bits pour le format double précision. Dans la suite de l'article, la **taille** de mantisse sera notée `p`.
L'exposant `e` est représenté par 8 bits pour le format simple (11 bits pour le format double).

2.2 Absorption

L'absorption est un phénomène qui apparait lors de l'addition de deux flottants d'ordres de grandeur très différents. Le résultat de l'addition est alors le flottant le plus grand.

Par exemple, en C, sous Unix avec le format simple précision et un arrondi au plus près :

$$10^8 + 1.0 = 10^8$$

Dans cet exemple, la valeur 1.0 est absorbée par la valeur 10^8.

2.3 Cancellation

La cancellation correspond à une élimination des chiffres significatifs de poids les plus élevés. Elle apparait lors de la soustraction de deux flottants très proches résultant d'un calcul. Le phénomène est d'autant plus important lors d'une accumulation de calculs avec des erreurs d'arrondis. Cette accumulation d'erreurs de calcul est mise en évidence par la soustraction qui est exacte lorsque les opérandes sont proches [24].

Par exemple, en C, sous Unix avec le format simple précision et un arrondi au plus près :

$$((1.0 - 10^{-7}) - 1.0) * 10^7$$

Dans cet exemple, la soustraction entre les valeurs correspondant au résultat de $(1.0 - 10^{-7})$ et 1.0 entraine une perte des chiffres significatifs. Or, le résultat de la soustraction est utilisé dans une multiplication et met en valeur le phénomène de cancellation intervenu lors de la soustraction. Le résultat devrait être -1 alors qu'il est de 1.1920928955078125.

2.4 Notations

Dans la suite de cet article, x, y et z dénotent les variables et $\mathbf{x}$, $\mathbf{y}$ et $\mathbf{z}$, leurs domaines respectifs. Quand cela est requis, $x_{\mathbb{F}}$, $y_{\mathbb{F}}$ et $z_{\mathbb{F}}$ représentent des variables sur $\mathbb{F}$ et $\mathbf{x}_{\mathbb{F}}$, $\mathbf{y}_{\mathbb{F}}$ et $\mathbf{z}_{\mathbb{F}}$, leurs domaines respectifs, alors que $x_{\mathbb{R}}$, $y_{\mathbb{R}}$ et $z_{\mathbb{R}}$ dénotent les mêmes variables sur $\mathbb{R}$ et $\mathbf{x}_{\mathbb{R}}$, $\mathbf{y}_{\mathbb{R}}$ et $\mathbf{z}_{\mathbb{R}}$, leurs domaines respectifs. Notons que $\mathbf{x}_{\mathbb{F}} = [\underline{x}_{\mathbb{F}}, \overline{x}_{\mathbb{F}}] = \{x_{\mathbb{F}} \in \mathbb{F}, \underline{x}_{\mathbb{F}} \leq x_{\mathbb{F}} \leq \overline{x}_{\mathbb{F}}\}$ avec $\underline{x}_{\mathbb{F}} \in \mathbb{F}$ et $\overline{x}_{\mathbb{F}} \in \mathbb{F}$. De même, $\mathbf{x}_{\mathbb{R}} = [\underline{x}_{\mathbb{R}}, \overline{x}_{\mathbb{R}}] = \{x_{\mathbb{R}} \in \mathbb{R}, \underline{x}_{\mathbb{R}} \leq x_{\mathbb{R}} \leq \overline{x}_{\mathbb{R}}\}$ avec $\underline{x}_{\mathbb{R}} \in \mathbb{F}$ et $\overline{x}_{\mathbb{R}} \in \mathbb{F}$. Soit $x_{\mathbb{F}} \in \mathbb{F}$, alors $x_{\mathbb{F}}^+$ est le plus petit nombre flottant strictement supérieur à $x_{\mathbb{F}}$ et $x_{\mathbb{F}}^-$ est le plus grand flottant strictement inférieur à $x_{\mathbb{F}}$.

3 Stratégies de recherches basées sur les propriétés des nombres flottants

Dans un article précédent [26], nous avons introduit un ensemble de stratégies basées sur des propriétés propres aux flottants. Ces propriétés se concentrent sur les domaines des variables, ou encore sur la structure des contraintes. Toutes ces propriétés caractérisent des comportements spécifiques aux flottants, qu'on ne retrouve pas forcément sur les entiers.

Parmi les propriétés qui se focalisent sur les domaines des variables, on compte la taille, la cardinalité, la densité et la magnitude de ces domaines.

Pour les propriétés qui se basent sur les contraintes on trouve, le degré, le nombre d'occurrences, l'absorption, et la cancellation.

Les stratégies basées sur ces propriétés sélectionnent la variable qui minimise ou maximise ces quantités. Par exemple, la stratégie `maxDens` sélectionne la variable dont le domaine maximise la densité.

3.1 stratégies de sélection de sous-domaine

Dans le même article, nous avons également introduit des stratégies de sélection de sous-domaines très simple qui mixent *coupe* et *énumération*. L'intérêt de ces stratégies est de tenir un peu plus compte de la nature même des flottants. Les domaines flottants se trouvent entre les domaines entiers et réels : fini, mais non uniforme. Les techniques d'énumération ou de bissection seules ne sont pas adaptées, d'où l'idée de mixer les deux. La Figure 2 rappelle ces stratégies.

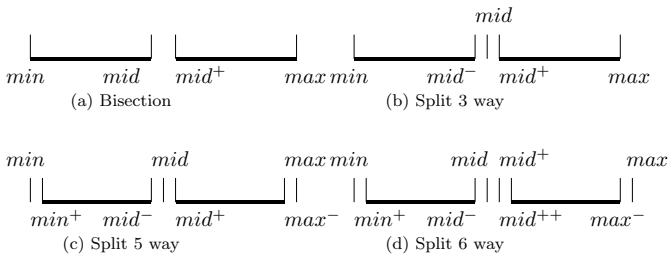


FIGURE 2 – Précédentes stratégies de sélection de sous-domaines

La Figure 2a correspond à la bissection classique. La Figure 2b complète la bissection en énumérant à chaque étape le milieu du domaine. Les Figures suivantes étendent cette dernière en augmentant les énumérations (au niveau des bornes ou des extrémités)

4 Stratégies de sélection de sous-domaines

Nous proposons, deux nouvelles stratégies de sélection de sous-domaines : la première tire profit de l'absorption, alors que la seconde est inspirée des consistances fortes comme la 3B-consistance.

4.1 Stratégie de sélection de sous-domaines basée sur l'absorption

La stratégie de sélection de sous-domaines branche sur deux variables en même temps : la variable avec le plus fort taux d'absorption (représentée par x dans la Figure 3), et la variable qui a le plus de chance de se faire absorber par elle (représentée par y dans la Figure 3). Une fois ces deux variables sélectionnées, la stratégie va générer jusqu'à trois sous-domaines sur chaque variable. La Figure 3 présente un cas où il y a deux sous-domaines (cas où les domaines sont positifs). Pour avoir les sous-domaines manquants, il suffit d'étendre ces cas par symétrie. Dans cette Figure, les sous-domaines les plus intéressants sont pour x : $[2^{e_{\overline{x}}}, \overline{x}]$ et pour y : $[0, 2^{e_{\overline{x}}-p-1}]$ (p = 23 en simple précision). Le premier représente le sous-domaine qui absorbe des valeurs de y. Le second représente le sous-domaine totalement absorbé par x. L'objectif de cette stratégie est de se concentrer sur les phénomènes d'absorptions les plus prometteurs, en regardant en priorité les sous-domaines réellement considérés lors d'une absorption.

FIGURE 3 – Sous-domaines générés par la stratégie $(abs(\underline{x}) < abs(\overline{x}))$

Cette stratégie est complémentaire à la stratégie de sélection de variables `MaxAbs` qui sélectionne la variable maximisant le taux d'absorption. Elle peut également être combinée avec une autre stratégie par défaut, qui est appelée dans le cas où x n'a, par exemple, aucunes valeurs qui absorbent y.

4.2 Stratégie de sélection de sous-domaines inspirée de la 3B-consistance : 3BSplit

La seconde stratégie introduite s'inspire des idées de la 3B-consistance [17]. Cette consistance est basée sur une 2B-consistance et cherche à renforcer la consistance des bords des domaines. La stratégie de sélection de sous-domaines `3BSplit` reprend cette idée. Elle tente d'abord de réfuter les extrémités des domaines, puis termine avec une simple bissection. La Figure 4 illustre cette stratégie. `3BSplit` explore un ensemble de sous-domaines de la forme : $[\underline{x}, \underline{x} + \delta]$ en doublant le δ à chaque succès. Si la recherche dans le sous-domaine est trop coûteuse (profondeur du sous-arbre trop importante), et une fois le sous-domaine complètement réfuté, la stratégie effectue le même processus sur la borne supérieure. Les sous-domaines associés à la borne supérieure sont de la forme $[\overline{x} - \delta, \overline{x}]$.

Cette stratégie de sélection de sous-domaines peut-être combinée avec une autre stratégie lors de la recherche dans un sous-domaine (correspond à la ligne *sub-cut* dans la Table 1). Par exemple, une simple bisection pourrait être utilisée, pour réfuter $[\underline{x}, \underline{x} + \delta]$

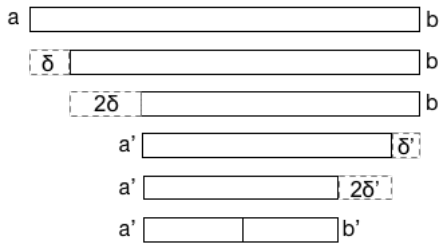


FIGURE 4 – Illustration de 3BSplit

5 Évaluation Expérimentale

Nous avons combiné différentes stratégies de sélection de variables, avec des stratégies de sélection de sous-domaines sur un ensemble de 95 benchmarks. On a également fait varier le type de consistance des stratégies, les sous-coupes (notée sub-cut), et la possibilité de resélectionner ou non la même variable au nœud suivant. Les sous-coupes correspondent à une stratégie de sélection de sous-domaines alternative qui est pertinente pour *splitAbs* et $3BSplit$. Pour *splitAbs*, cette stratégie alternative est appelée lorsque aucune absorption se produit. Pour $3BSplit$, cette stratégie est utilisée pour réfuté les sous-domaines. La stratégie standard est basée sur un ordre lexicographique pour la sélection de variables, une bissection pour le choix de sélection de sous-domaines ainsi qu'un filtrage 2B. Le nombre de stratégies issues de ces combinaisons est important. Les benchmarks ont été comparés sur plus de 200 stratégies différentes.

Toutes les expérimentations ont été effectuées avec un système Linux, et un processeur Intel Xeon cadencé à 2.40GHz et avec 12GB de mémoire. Toutes les stratégies ont été implantées dans le solveur Objective-CP [21]. Tous les calculs flottants sont effectués en simple précision et avec un mode d'arrondi "au plus près".

5.1 Benchmarks

Les benchmarks utilisés lors de ces expérimentations viennent du test et de la vérification de programme avec des nombres flottants. SMTLib [1], FPBench [7], et CBMC [3] (mais aussi [5, 4, 8]) sont les principales sources utilisées pour ces benchmarks. Il y a autant de benchmarks SAT que UNSAT. Le nombre de contraintes et variables de ces derniers varie de 2 à environ 3000. Les sections "résultat" et "analyse" ci-dessous se concentrent essentiellement sur les benchmarks satisfiables.

5.2 Résultats

Dans l'ensemble des tableaux présentés dans cette section, les temps sont en millisecondes et le timeout est d'une minute. Le tableau 1 synthétise les résultats de l'ensemble des stratégies sur les benchmarks satisfiables. Ce tableau est trié selon la somme cumulée des temps de résolution. Par souci de concision, en plus de la stratégie de référence, seules les 11 meilleurs et les 11 moins bonnes stratégies sont présentées. La totalité du tableau peut être trouvée à l'adresse : `www.i3s.unice.fr/~hzitoun/jfpc2018/benchmark.html`. Dans ce tableau, la première colonne correspond au nom du benchmark, les 11 colonnes suivantes correspondent aux résultats des meilleures stratégies. La colonne du milieu correspond à la stratégie de référence. Les 11 dernières quant à elles forment les moins bonnes stratégies. La colonne `VBS`, correspond au *virtual best solver*, où le temps pour chaque benchmark est le temps minimal de résolution sur l'ensemble des stratégies. La dernière colonne `hybrid` correspond à la stratégie adaptative présentée dans la section 5.3.2. Pour chaque stratégie, la première ligne révèle le critère de sélection de variables, la seconde le choix de stratégie de sélection de sous-domaine. La stratégie de sélection de sous-domaines basée sur la `3B` est suivie par le pourcentage (corresponds à δ dans la figure 4). La troisième ligne présente la stratégie de sélection de sous-domaines alternative `sub-cut`. La quatrième ligne explicite le type de filtrage utilisé (2B-consistance, ou 3B-consistance) ainsi que le pourcentage utilisé pour ces derniers. La cinquième ligne indique la resélection de la même variable au noeud suivant. Enfin, la dernière ligne explicite pour chaque stratégie, la somme des temps sur l'ensemble des benchmarks.

Le Tableau 2 présente la première occurrence de chaque stratégie de sélection de variables. La première ligne donne le nom du critère de sélection de variables. La seconde ligne explicite le classement de la meilleure stratégie basée sur ce critère.

De la même manière, le Tableau 3 présente les classements des stratégies de sélection de sous-domaine.

5.3 Analyse

Dans le Tableau 1, la stratégie standard (ordre lexicographique avec une bissection, un filtrage 2B à 5% et la resélection est autorisée) se trouve à la 108^{eme} position sur 231 stratégies. Il y a donc 107 stratégies parmi celles introduites qui sont clairement plus efficaces que la stratégie standard pour la résolution de ce genre de problème. Le `VBS` est 120 fois plus rapide que la stratégie de référence. La meilleure stratégie (colonne 1) est 4 fois plus rapide que la stratégie de référence. Utiliser les spécificités des flottants pour guider la recherche a donc un impact clair sur le temps de résolution. Pour les stratégies les plus efficaces sur cet ensemble de benchmarks (Tableau 1 et 2), les choix

TABLE 1 – Classement des 231 stratégies

	1	2	3	4	5	6	7	8	9	10	11	···	108	···	221	222	223	224	225	226	227	228	229	230	231		
Var Choice	Lex	MaxOcc	MaxOcc	Lex	Lex	Lex	Lex	MaxDens	MaxAbs	MaxOcc	MaxAbs	···	**Lex**	···	MaxDens	MaxWidth	MaxWidth	MaxWidth	MaxWidth	MaxWidth	MaxWidth	MaxDens	MaxWidth	MaxWidth	MaxCard		
Cut	*6Split*	*6Split*	*6Split*	*6Split*	*6Split*	*3B-10*	*6Split*	*6Split*	*6Split*	*6Split*	*3B-10*	···	**Split**	···	*3B-10*	*6Split*	*Split*	*6Split*	*3B-10*	*Split*	*6Split*	*3B-10*	*3B-1*	*3B-1*	*3B-1*		
SubCut						*6Split*					*6Split*	···		···	*6Split*				*6Split*			*split*	*3B-1*	*3B-1*	*3B-1*		
Filtering	*2B(5)*	*2B(5)*	*2B(0)*	*2B(5)*	*2B(0)*	*2B(5)*	*2B(0)*	*2B(5)*	*3B(5)*	*2B(0)*	*3B(5)*	···	**2B(5)**	···	*2B(5)*	*2B(0)*	*2B(5)*	*2B(5)*	*2B(0)*	*2B(0)*	*2B(0)*	*2B(5)*	*2B(5)*	*2B(5)*	*2B(5)*		
Reselect	*n*	*n*	*n*	*y*	*n*	*n*	*y*	*n*	*y*	*y*	*n*	···	**y**	···	*y*	*n*	*y*	*y*	*y*	*y*	*y*	*y*	*y*	*n*	*y*		
benchmark	t(ms)	t(ms)	t(ms)	t(ms)	t(ms)	t(ms)	t(ms)	t(ms)	t(ms)	t(ms)	t(ms)	···	t(ms)	···	t(ms)	t(ms)	t(ms)	t(ms)	t(ms)	t(ms)	t(ms)	t(ms)	t(ms)	t(ms)	t(ms)	**VBS**	**Hybrid**
PID_diff_opt	221	44	41	TO	373	21	TO	21	3117	44	2678	···	TO	···	21	TO	28455	50599	TO	54396	TO	TO	TO	TO	TO	19	2678
sine_1	7	6	7	7	3	54	7	7	4	6	196	···	11	···	86	11	19	6	13	29	10	59	10	11	101	3	196
sine_2	5	4	6	7	6	51	5	4	5	7	215	···	11	···	5	14	32	6	13	44	11	31	9	10	9	3	215
sine_3	4	5	6	5	6	51	5	5	5	5	240	···	11	···	5	12	31	7	13	36	15	35	8	10	12	4	240
PID_diff_opt_5	27	57	55	27	29	22	28	23	2414	54	4134	···	TO	···	TO	TO	2056	TO	TO	7427	TO	TO	TO	TO	TO	22	4134
PID_diff_opt_6	64	74	73	674	78	27	1034	27	3185	71	5679	···	TO	···	TO	TO	9364	TO	TO	24458	TO	TO	TO	TO	TO	25	5679
PID_diff_opt_7	1530	90	90	4316	2451	31	7412	32	9833	91	7925	···	TO	···	TO	TO	15267	TO	TO	31537	TO	TO	TO	TO	TO	30	7925
PID_diff_opt_8	3601	112	114	10823	5198	33	14643	36	5373	112	10024	···	TO	···	TO	TO	40216	TO	TO	TO	TO	TO	TO	TO	TO	32	10024
PID_diff_opt_9	78	132	133	84	99	38	108	42	8291	136	13100	···	TO	···	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO	38	13100
PID_diff_opt_10	TO	157	159	TO	TO	42	TO	43	10259	158	15383	···	TO	···	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO	42	15383
square_1	8	7	9	5	12	18	12	6	28	11	62	···	9	···	8	11	11	8	17	14	13	9	14	14	12	4	5
square_2	7	7	13	7	13	15	12	7	30	15	67	···	7	···	9	14	7	8	19	15	12	10	17	15	16	6	7
square_3	11	11	14	10	13	17	14	10	31	16	51	···	11	···	11	14	11	9	18	16	13	13	15	17	13	8	10
square_4	6	7	4	6	6	4	5	7	21	5	39	···	9	···	5	7	10	6	5	9	5	7	10	10	9	3	6
square_5	5	5	5	4	6	7	5	5	21	7	43	···	8	···	5	3	10	5	6	11	6	7	9	9	7	3	4
square_6	3	5	5	6	6	7	4	2	21	7	38	···	8	···	5	7	11	4	8	6	5	8	9	8	9	2	6
square_7	5	5	4	6	6	5	6	7	21	5	38	···	9	···	5	7	6	7	6	9	7	9	10	8	9	2	6
square_8	6	6	7	4	4	6	8	4	20	7	39	···	7	···	7	6	10	6	6	9	4	7	9	12	11	2	4
newton_1_4	6	7	7	6	6	10	5	7	130	6	404	···	47	···	TO	33	83	26	27	123	31	TO	83	92	181	4	404
newton_1_5	8	7	6	6	7	153	7	5	100	8	401	···	44	···	7	30	88	17	35	153	28	TO	90	92	7	5	401
newton_1_6	7	6	6	7	8	172	6	7	67	7	680	···	48	···	8	30	96	188	192	259	407	29	101	119	120	4	680
newton_1_7	7	7	5	8	7	5	8	21	65	8	521	···	45	···	TO	31	TO	TO	TO	TO	TO	TO	140	268	TO	4	521
newton_1_8	6	6	6	7	7	441	7	7	24	5	160	···	17	···	TO	10068	TO	TO	TO	TO	TO	TO	425	TO	TO	5	160
newton_2_6	9	94	107	10	9	176	11	8	95	121	2101	···	25	···	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO	8	2101
newton_2_7	8	11	12	8	7	15081	9	TO	970	TO	125	···	162	···	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO	7	125
newton_2_8	8	11	13	11	12	489	9	TO	608	TO	82	···	31	···	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO	8	82
newton_3_6	11	268	1676	12	10	TO	10	TO	1042	153	247	···	41	···	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO	10	247
newton_3_7	11	TO	TO	13	12	6764	12	TO	1336	TO	1353	···	1350	···	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO	9	1353
newton_3_8	12	TO	TO	10	11	TO	11	TO	890	TO	913	···	43	···	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO	9	913
e1_2	1923	1945	132	257	131	1891	133	1922	271	132	1020	···	175	···	TO	131	179	1454	132	132	134	TO	TO	TO	TO	128	257
slope26+10	6	6	6	6	7	5	5	4	12	4	15	···	7	···	4	7	4391	65	33	10	23	6	23336	TO	7	3	6
heron156	335	263	574	TO	520	TO	TO	16	2324	530	756	···	TO	···	TO	194	TO	201	201	TO	196	TO	TO	TO	TO	16	756
heron	5	5	6	7	4	6	6	7	16	4	45	···	8	···	TO	24	8	317	340	11	102	TO	TO	501	TO	3	45
slope26-1	4	5	7	6	5	5	3	5	5	5	5	···	6	···	5	7	TO	7	4	TO	3	5	4	7	6	2	6
solve_quadratic	4	5	5	4	4	3	5	5	4	6	4	···	4	···	3	3	2	5	1	6	5	5	4	6	4	1	4
solve_cubic	TO	8	8	212	TO	TO	172	8	292	6	**TO**	···	TO	···	197	8	TO	5	83	TO	4	TO	TO	TO	TO	4	212
MullerKahan	62	555	557	59	59	58	58	60	57	562	63	···	59	···	60	59	60	62	61	58	58	59	58	62	60	55	59
optimized_heron	11	21	9	7	5	5	9	32	TO	8	38	···	TO	···	3186	TO	TO	35	TO	TO	TO	31	TO	33	TO	5	38
heron10-8	7	7	6	8	6	7	7	TO	33	7	45	···	TO	···	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO	5	45
Odometrie_1	13	22	22	14	13	13	14	14	734	22	1455	···	26	···	TO	15	37	12	13	32	13	TO	15	16	66	9	1455
Odometrie_10	95	462	477	95	105	86	102	94	TO	476	**TO**	···	392	···	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO	86	95
Odometrie_50	709	9297	10388	692	1827	683	1834	700	TO	10425	**TO**	···	2284	···	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO	683	692
Odometrie_150	2273	TO	TO	2235	28704	2180	28801	2250	TO	TO	**TO**	···	5793	···	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO	2180	2235
Odometrie_200	3020	TO	TO	3005	59691	2914	TO	3067	TO	TO	**TO**	···	7582	···	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO	2914	3005
runge_kutta_1_1	119	9	8	6561	68	19	1364	7	50	9	202	···	TO	···	7	TO	24	TO	774	45	TO	16	945	TO	819	6	202
runge_kutta_1_2	TO	14	14	TO	TO	12	TO	11	347	14	2199	···	TO	···	11	TO	TO	TO	TO	TO	TO	31	TO	TO	441	9	2199
runge_kutta_1_3	13	18	18	13	15	13097	13	14	506	19	1804	···	32	···	14	TO	TO	TO	TO	TO	TO	43	TO	TO	TO	13	13
runge_kutta_1_4	16	22	24	15	18	13211	17	16	4851	25	1018	···	39	···	16	TO	TO	TO	TO	TO	TO	54	TO	TO	TO	13	15
runge_kutta_1_5	20	29	31	17	18	13237	19	17	14529	30	1146	···	48	···	18	TO	TO	TO	TO	TO	TO	69	TO	TO	TO	17	17
Sum	**194316**	253844	254875	269302	279605	311172	355965	368592	372037	373349	376753	···	**798409**	···	1443708	1450746	1480484	1493065	1502020	1558845	1561105	1620543	1645321	1681320	1681919	**6473**	**77965**

Var choice	Lex	MaxOcc	MaxDens	MaxAbs	MaxCanc	MaxCard	MaxWidth	Ref
First apparition	1	1	8	9	20	69	77	108

TABLE 2 – Première occurrence pour chaque stratégie de choix de variable

Cut	6Split	3B-10	Split	3B-5	3B-1	SplitAbs
First apparition	1	6	29	34	41	45

TABLE 3 – Première occurrence pour chaque stratégie de choix de sous-domaine

de sélection de variables sont basés : sur l'ordre lexicographique, sur le nombre d'occurrences, la densité ou l'absorption. Alors que les stratégies basées sur la taille, choix de sélection de variables classiques sur les domaines entiers, peinent à résoudre les problèmes dès qu'il commence à devenir un peu réaliste. Les stratégies basées sur la cardinalité sont également dans le même que cas. Les premières apparitions de `MaxWidth` et `MaxCard` se trouvent respectivement à la 69^{eme} et 77^{eme} place.

Les stratégies de sélection de sous-domaines introduites dans cet article ont de bons résultats. En effet, comme l'explicite le Tableau 3, la première stratégie exploitant la `3B` est en 6^{eme} position. La meilleure stratégie qui utilise `splitAbs` quant à elle est 44^{eme}. Ces stratégies se comportent tout aussi bien pour des problèmes insatisfiables, en résolvant autant de benchmarks que le VBS.

Les stratégies que l'on a proposées ont un impact positif sur les temps de résolution des benchmarks. Néanmoins aucune de ces stratégies à elle seule ne résout la totalité des benchmarks. Plusieurs possibilités sont envisageables pour pallier à cette problématique. Nous proposons deux solutions : la mise en place d'un portfolio, ou une stratégie hybride.

5.3.1 Portfolio

Le portfolio de stratégies est une solution conceptuellement simple. Plusieurs travaux traitent de ce sujet [9, 22, 19]. L'idée principale est de lancer en parallèle un ensemble de stratégies, puis de s'arrêter lorsque l'une d'elles trouve une solution. Au vu des résultats expérimentaux, on remarque que seules 23 stratégies sont nécessaires pour couvrir exactement le VBS. On pourrait utiliser une méthode de résolution basée sur un portfolio de stratégies exécutées de manière concurrente et avec une allocation dynamique des temps de résolution.

5.3.2 Stratégie hybride

Une solution plus pertinente consiste à définir une nouvelle stratégie hybride basée sur une combinaison de stratégies. Cette stratégie hybride est conceptuellement proche du portefolio. Elle correspond à une seule stratégie qui selon un critère qui reflète la structure du problème va choisir une stratégie ou une autre. Pour déterminer cette stratégie hybride, il faut trouver un ensemble de stratégies qui lorsqu'elles sont combinées résolvent la totalité des benchmarks. L'étape la plus difficile consiste à trouver un ensemble de critères permettant de décider lorsqu'il faut changer de stratégie. Ces critères peuvent être déterminés par une analyse de corrélation [16, 12] sur un ensemble de métadonnées tirées du modèle (e.g nombre de variables, contraintes, occurences,...). L'avantage principal de cette solution est qu'elle est conceptuellement simple et facile à mettre en oeuvre.

Dans notre cas, l'analyse de corrélation n'a pas donné de résultat satisfaisant. Néanmoins, à la suite d'une observation simple, nous avons déterminé un critère permettant de choisir la stratégie adéquate pour le problème. En effet, en regardant la stratégie basée sur l'absorption (colonne 11 du Tableau 1), nous avons essayé de comprendre pourquoi cette stratégie provoquait des timeouts. Cette stratégie se base sur l'absorption, c'est donc un des critères à analyser. Le tableau 4 présente le pourcentage de variables non fixées (domaine non dégénéré) avec de l'absorption. Pour chaque benchmark où cette stratégie effectue un timeout (benchmarks en gras dans les Tables 1 et 4), le pourcentage de variables non fixées est faible (inférieur à 5%). Cette constatation est cohérente avec l'état d'esprit de cette stratégie. Elle est construite pour tirer profit des phénomènes d'absorption. S'il n'y a pas ou peu d'absorption, elle est inefficace. On a donc un critère : il faut changer de stratégie lorsque ce pourcentage est inférieur à 5%. Les stratégies basées sur un ordre lexicographie (colonne 4), et celle basée sur la densité (colonne 8) sont de bonnes candidates pour compléter cette stratégie. Cette stratégie hybride est synthétisée par l'algorithme 1. Elle permet de résoudre la totalité des benchmarks et est simple à implanter. Cette stratégie est plus rapide que la stratégie de référence par un ordre de grandeur, mais reste plus lente d'un ordre de grandeur que le `VBS`. Elle est également 2.5 fois plus rapide que la meilleure stratégie (colonne 1). Cette approche est un bon compromis entre simplicité, et efficacité.

5.3.3 Benchmarks insatisfiables

Une analyse similaire a été effectuée pour les benchmarks insatisfiables. La stratégie hybride n'est pas suffisante pour résoudre ces benchmarks. Néanmoins, plusieurs stratégies résolvent autant de benchmarks que le VBS. Par exemple, la stratégie qui utilise un `maxAbs` comme choix de variable, un `3B-10` comme stratégie

Algorithm 1 Stratégie Hybride

```
procedure HybridSearch(vars)
    nb ← count(for v in vars : if !bound(v))
    abs ← computeNBAbs(for v in vars : if !bound(v))
    nbVarsGT ← count(for a in abs : if a > 0)
    if (nbVarsGT/nb * 100) > 5 then
        maxAbsorption(vars,3B-10,6split,3B(5),n)
    else
        lex(6split,None,2B(5),y)
```

de sélection de sous-domaines et sous-coupe, une 3B à 5% comme filtrage et qui autorise la resélection ne fait que trois timeouts. Ces trois benchmarks en question ne sont résolus par aucune stratégie. À la suite de quelques expérimentations, on a pu déterminer que pour que ces benchmarks soient résolus dans un temps raisonnable, un effort supplémentaire doit être fait au niveau du solveur pour supprimer les variables inutiles au niveau du modèle.

6 Conclusion et perspectives

Dans Un article précédent, nous avons proposé un ensemble de stratégies de sélection de variables basées sur les spécificités des flottants pour guider la recherche. Ces stratégies de sélection de variables améliorent la recherche d'un contre-exemple illustrant la violation d'une propriété dans un programme à vérifier, mais ne suffisent pas pour passer à l'échelle et résoudre des benchmarks plus difficiles et réalistes. Des stratégies de sélection de sous-domaines dédiés aux flottants sont nécessaires. Les contributions de cet article sont un ensemble de stratégies de sélection de sous-domaines sur des flottants. La première, exploite les phénomènes d'absorption, alors que la seconde est fondée sur des idées dérivées de consistances fortes. Ces stratégies sont comparées sur un ensemble significatif de benchmarks. Plusieurs stratégies présentées, sont efficaces et permettent d'obtenir de bien meilleurs résultats que la stratégie standard utilisée pour résoudre ce genre de problème. Néanmoins, aucune de ces stratégies ne résout tous ces benchmarks en un temps raisonnable. Deux approches ont été proposées pour contourner ce problème : un portfolio de stratégies et une stratégie adaptative. En raison de sa simplicité et de ses bonnes performances, la stratégie adaptative présentée est plus pertinente que le portfolio.

Benchmarks	**% de vars. avec de l'abs.**
PID_diff_opt	72.73
sine_1	50.00
sine_2	50.00
sine_3	50.00
PID_diff_opt_5	74.07
PID_diff_opt_6	75.00
PID_diff_opt_7	75.68
PID_diff_opt_8	76.19
PID_diff_opt_9	76.60
PID_diff_opt_10	77.88
square_1	0.00
square_2	0.00
square_3	0.00
square_4	0.00
square_5	0.00
square_6	0.00
square_7	0.00
square_8	0.00
newton_1_4	25.00
newton_1_5	25.00
newton_1_6	25.00
newton_1_7	25.00
newton_1_8	25.00
newton_2_6	28.57
newton_2_7	28.57
newton_2_8	28.57
newton_3_6	30.00
newton_3_7	30.00
newton_3_8	30.00
e1_2	0.00
slope26+10	0.00
heron156	80.00
heron	80.00
slope26-1	0.00
solve_quadratic	0.00
solve_cubic	**0.00**
MullerKahan	0.00
optimized_heron	75.00
heron10-8	80.00
Odometrie_1	10.00
Odometrie_10	**4.11**
Odometrie_50	**1.42**
Odometrie_150	**0.47**
Odometrie_200	**0.36**
runge_kutta_1_1	28.57
runge_kutta_1_2	27.27
runge_kutta_1_3	0.00
runge_kutta_1_4	0.00
runge_kutta_1_5	0.00

Table 4 – Pourcentage de variable pouvant mener à une absorption

Références

[1] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2016.

[2] Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting systematic search by weighting constraints. *ECAI'04*, pages 146–150, 2004.

[3] Edmund Clarke, Daniel Kroening, and Flavio Lerda. A tool for checking ansi-c programs. In Kurt Jensen and Andreas Podelski, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 168–176, 2004.

[4] Hélène Collavizza, Claude Michel, and Michel Rueher. Searching critical values for floating-point programs. In *Testing Software and Systems - 28th IFIP WG 6.1 International Conference, ICTSS 2016, Graz, Austria, October 17-19, 2016, Proceedings*, pages 209–217, 2016.

[5] Hélène Collavizza, Michel Rueher, and Pascal Van Hentenryck. Cpbpv : A constraint-programming framework for bounded program verification. *Constraints*, 15(2) :238–264, 2010.

[6] Hélène Collavizza, Nguyen Le Vinh, Michel Rueher, Samuel Devulder, and Thierry Gueguen. A dynamic constraint-based BMC strategy for generating counterexamples. *26th ACM Symposium On Applied Computing*, 2011.

[7] Nasrine Damouche, Matthieu Martel, Pavel Panchekha, Chen Qiu, Alexander Sanchez-Stern, and Zachary Tatlock. Toward a standard benchmark format and suite for floating-point analysis. In Sergiy Bogomolov, Matthieu Martel, and Pavithra Prabhakar, editors, *Numerical Software Verification*, pages 63–77, Cham, 2017. Springer International Publishing.

[8] Vijay D'Silva, Leopold Haller, Daniel Kroening, and Michael Tautschnig. Numeric bounds analysis with conflict-driven learning. In Cormac Flanagan and Barbara König, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 48–63, 2012.

[9] Matteo Gagliolo and Jürgen Schmidhuber. Learning dynamic algorithm portfolios. *Annals of Mathematics and Artificial Intelligence*, 2006.

[10] Steven Gay, Renaud Hartert, Christophe Lecoutre, and Pierre Schaus. Conflict ordering search for scheduling problems. In *Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings*, pages 140–148, 2015.

[11] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surv.*, 23(1) :5–48, 1991.

[12] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf. Support vector machines. *IEEE Intelligent Systems and their Applications*, 13(4) :18–28, July 1998.

[13] IEEE. IEEE standard for binary floating-point arithmetic. *ANSI/IEEE Standard*, 754, 2008.

[14] Narendra Jussien and Olivier Lhomme. Dynamic domain splitting for numeric csps. *ECAI*, pages 224–228, 1998.

[15] R. Baker Kearfott. Some tests of generalized bisection. *ACM Trans. Math. Softw.*, 13(3) :197–220, September 1987.

[16] J. Kiefer and J. Wolfowitz. Stochastic estimation of the maximum of a regression function. *Ann. Math. Statist.*, 23(3) :462–466, 09 1952.

[17] Olivier Lhomme. Consistency techniques for numeric csps. In *Proceedings of the 13th International Joint Conference on Artifical Intelligence - Volume 1*, IJCAI'93, pages 232–238, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[18] Jeff T. Linderoth and Martin W. P. Savelsbergh. A computational study of search strategies for mixed integer programming. *INFORMS Journal on Computing*, 11(2) :173–187, 1999.

[19] Manuel Loth, Michèle Sebag, Youssef Hamadi, and Marc Schoenauer. Bandit-based search for constraint programming. In Christian Schulte, editor, *Principles and Practice of Constraint Programming*. Springer Berlin Heidelberg, 2013.

[20] Laurent Michel and Pascal Van Hentenryck. Activity-based search for black-box constraint programming solvers. In Nicolas Beldiceanu, Narendra Jussien, and Éric Pinson, editors, *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems : 9th International Conference, CPAIOR 2012, Nantes, France, May 28 – June1, 2012. Proceedings*, pages 228–243, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[21] Laurent D. Michel and Pascal Van Hentenryck. A microkernel architecture for constraint programming. *Constraints*, 22(2) :107–151, 2017.

[22] Anthony Palmieri, Jean-Charles Régin, and Pierre Schaus. Parallel strategies selection. In *Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings*, pages 388–404, 2016.

[23] Philippe Refalo. Impact-based search strategies for constraint programming. *CP 2004*, 3258 :557–571, 2004.

[24] P.H. Sterbenz. *Floating-point computation.* Prentice-Hall series in automatic computation. Prentice-Hall, 1973.

[25] Pascal Van Hentenryck and Laurent Michel. The objective-cp optimization system. In *Principles and Practice of Constraint Programming : 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*, pages 8–29, Berlin, Heidelberg, 2013.

[26] Heytem Zitoun, Claude Michel, Michel Rueher, and Laurent Michel. Search strategies for floating point constraint systems. In *Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, pages 707–722, 2017.

Index des auteurs