HAL
open science

# JFPC 2019 - Actes des 15es Journees Francophones de Programmation par Contraintes

Xavier Lorca, Élise Vareilles

## To cite this version:

Xavier Lorca, Élise Vareilles. JFPC 2019 - Actes des 15es Journees Francophones de Programmation par Contraintes. Xavier Lorca; Elise Vareilles. JFPC 2019 - Actes des 15es Journees Francophones de Programmation par Contraintes, Jun 2019, Albi, France. IMT Mines Albi, xv - 190 p., 2019, 979-10-91526-07-4. hal-02159866

**HAL Id: hal-02159866**

**https://hal-mines-albi.archives-ouvertes.fr/hal-02159866**

Submitted on 20 Jun 2019

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

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

# Actes des 15es Journées Francophones de Programmation par Contraintes

## JFPC 2019

12-14 juin 2019 – Albi, France

Rédacteur en chef : Xavier Lorca — Juin 2019

ISBN 979-10-91526-07-4 IMT Mines Albi

# Sommaire


Sommaire . . . . . v

Éditorial . . . . . vii

Comité scientifique 2019 . . . . . ix

Comité d'organisation 2019 . . . . . xi

Organisateurs et soutiens . . . . . xiii

Conférences invitées . . . . . xv

1 – « *Clustering descriptif : formulations en PLNE et en PPC et applications* » par Thi-Bich-Hanh Dao, Chia-Tung Kuo, S. S. Ravi, Christel Vrain, Ian Davidson *(2 pages)* . . . . . 1

2 – « *Décompositions structurelles et sémantiques, rapport préliminaire* » par Philippe Jégou, Cyril Terrioux *(4 pages)* . . . . . 3

3 – « *Préserver la confidentialité pour l'algorithme Generalized Distributed Breakout* » par Julien Vion, René Mandiau, Sylvain Piechowiak, Marius Silaghi *(10 pages)* . . . . . 7

4 – « *Impact de la granularité spatio-temporelle des données sur l'optimisation des tournées de livraison en ville* » par Omar Rifki, Nicolas Chiabaut, Christine Solnon *(10 pages)* . . . . . 17

5 – « *Unifier les stratégies de sélection de réserve avec la programmation par contraintes et les graphes* » par Dimitri Justeau-Allaire, Philippe Birnbaum, Xavier Lorca *(3 pages)* . . . . . 27

6 – « *Une approche de Programmation par Contraintes pour résoudre le Problème de Transport de Patients* » par Charles Thomas, Quentin Cappart, Pierre Schaus, Louis-Martin Rousseau *(2 pages)* 31

7 – « *Octogones entiers pour le problème RCPSP* » par Pierre Talbot, David Cachera, Éric Monfroy, Charlotte Truchet *(10 pages)* . . . . . 33

8 – « *De la pertinence des littéraux dans les contraintes pseudo-booléennes apprises* » par Daniel Le Berre, Pierre Marquis, Stefan Mengel, Romain Wallon *(10 pages)* . . . . . 43

9 – « *Contraintes de cardinalité cachées dans les preuves d'insatisfaisabilité* » par Valentin Montmirail, Marie Pelleau, Jean-Charles Régin, Laurent Simon *(10 pages)* . . . . . 53

10 – « *Utilisation d'algorithmes d'approximation en programmation par contraintes* » par Arthur Godet, Xavier Lorca, Gilles Simonin *(4 pages)* . . . . . 63

11 – « *Une approche SAT incrémentale pour raisonner efficacement sur les réseaux de contraintes qualitatives* » par Gaël Glorian, Jean-Marie Lagniez, Valentin Montmirail, Michael Sioutis *(10 pages)* . . . . . 67

12 – « *Heuristiques exploitant la relaxation linéaire pour l'optimisation dans les réseaux de fonction de coût* » par Fulya Trösser, Simon de Givry, George Katsirelos *(10 pages)* . . . . . 77

13 – « *Introduction de contraintes structurelles pour la résolution du problème du voyageur de commerce* » par Nicolas Isoart, Jean-Charles Régin *(9 pages)* . . . . . 87

14 – « *Contrainte globale abstractXOR : résultats de complexité et algorithmes de propagation* » par Loïc Rouquette, Christine Solnon *(11 pages)* . . . . . 97

15 – « *PyCSP³ : modéliser des problèmes combinatoires sous contraintes en Python* » par Christophe Lecoutre, Nicolas Szczepanski *(8 pages)* . . . . . 109

16 – « *Transformation de modèles et programmation par contraintes avec ATL$^c$* » par Théo Le Calvar, Fabien Chheln Frédéric Jouault, Frédéric Saubion *(4 pages)* . . . . . 117

17 – « *Compact-Diagram Propagateur efficace pour la contrainte (s)MDD* » par Hélène Verhaeghe, Christophe Lecoutre, Pierre Schaus *(2 pages)* . . . . . 121

18 – « *Extension de Compact-Diagram aux smart MVD* » par Hélène Verhaeghe, Christophe Lecoutre, Pierre Schaus *(2 pages)* . . . . . 123

19 – « *Gestion robuste des opérations sur Mars* » par Michael Saint-Guillain *(2 pages)* . . . . . 125

20 – « *Testé comme jamais...* » par Xavier Gillard, Pierre Schaus *(6 pages)* . . . . . 127

21 – « *Estimer le nombre de solutions des contraintes de cardinalité grâce à leur décomposition* `range` *et* `roots` » par Giovanni Lo Bianco, Xavier Lorca, Charlotte Truchet *(9 pages)* . . . . . 133

22 – « *Qualité et diversité garanties dans les réseaux de fonctions de coût* » par M. Ruffini, J. Vucinic, S. de Givry, G. Katsirelos, S. Barbe, T. Schiex *(10 pages)* . . . . . 143

23 – « *Une heuristique basée sur l'historique des conflits pour les problèmes de satisfaction de contraintes* » par Djamal Habet, Cyril Terrioux *(2 pages)* . . . . . 153

24 – « *Une autre règle de séparation pour des codages de CSP vers SAT* » par Richard Ostrowski, Lionel Paris, Adrien Varet *(9 pages)* . . . . . 155

25 – « *Sur la pertinence des décompositions arborescentes optimales pour la résolution de CSP* » par Philippe Jégou, Hélène Kanso, Cyril Terrioux *(2 pages)* . . . . . 165

26 – « *Heuristiques de recherche : un bandit pour les gouverner toutes* » par Hugues Wattez, Frédéric Koriche, Christophe Lecoutre, Anastasia Paparrizou, Sébastien Tabary *(9 pages)* . . . . . 167

27 – « *Sur l'UP-résilience des k-UCSs binaires* » par Mohamed Sami Cherif, Djamal Habet *(4 pages)* 177

28 – « *Améliorations de l'hybridation PPC et PLNE pour la somme coloration* » par Samba Ndojh Ndiaye *(10 pages)* . . . . . 181

# Éditorial

2005, première année des *Journées Francophones de Programmation par Contraintes*, suite à la création de *l'Association Française de Programmation par Contrainte* un an auparavant, c'est aussi cette année là que je débutais ma thèse de doctorat !

2019 ! Quatorze années se sont écoulées depuis et l'on me fait l'honneur de me proposer de présider le comité de programme de ces JFPC, quelle émotion ! Elles sont pour moi le fruit d'une longue histoire et d'une dynamique forte, très tôt donnée par la communauté francophone de programmation par contraintes. Elles prennent naissance de la fusion des conférences JFPLC (*Journées Francophones de la Programmation Logique avec Contraintes*) nées en 1992 et JNPC (*Journées Nationales sur la Résolution Pratique de Problèmes NP-Complets*) nées en 1994. Ainsi, les JFPC sont un évènement annuel qui rythme la vie des chercheurs francophones du domaine et leur permettent de suivre l'évolution et les tendances des recherches présentes et futures.

Point de nostalgie dans ces mots mais plutôt une interrogation personnelle sur ce qui m'a marqué durant cette période et la portée du travail accompli par notre communauté. Cette vue rétrospective est forcément biaisée par mes affinités thématiques autour des outils de résolution en contraintes, je retiendrais donc un élément lié à l'évolution du paysage des outils (les fameux « *solveurs* ») : qu'ils ont la vie dure au milieu de la PLNE, de SAT et de méta-heuristiques ! Bien entendu l'évolution notable de cette période concerne les « *lazy close generation* », fruit d'une intelligente et fructueuse hybridation de la PPC avec SAT. Une autre évolution notable concerne l'hybridation avec les méthodes de décomposition issue de la PLNE (lagrangienne en particulier). Là encore, une hybridation intelligente de ces décompositions par le biais des contraintes globales a permis de s'attaquer à des problèmes d'optimisation pour lesquels la PPC seule était bien mal armée. Et voilà donc un questionnement, un brin provocateur, qui me démange : la PPC n'est-elle qu'un bon catalyseur, un bon cadre, pour valoriser les qualités d'autres techniques ? N'y voyez pas d'ironie mais juste une vraie question après avoir travaillé dix ans autours des outils de contraintes, défendu leur généricité, leur flexibilité et quelques fois leur efficacité !

Je ne sais ni quand, ni comment, les outils de contraintes feront leur révolution comme l'ont connu les outils de PLNE et de SAT mais je suis convaincu qu'il nous reste de belles choses à faire. En effet, plus je découvre de nouvelles opportunités d'applications de la PPC, plus je me rends compte que nous ne valorisons pas assez la cible originelle : les problèmes hétérogènes dans leur forme et fortement combinatoire de « *satisfaction* » (existe-t-il une solution à... ?). Cela ne veut pas dire qu'il ne faut plus, par exemple, faire de l'optimisation avec des contraintes mais qu'il faut probablement s'interroger sur la nature du problème de décision sous-jacent. Cela veut probablement aussi dire que notre communauté doit promouvoir dans son travail des éléments valorisant les forces du paradigme plus que de tenter de démontrer à grand coup d'expérimentations que nous pouvons être « *proche* » ou tout juste meilleurs que d'autres techniques. Je ne critique pas ici le fait d'expérimenter mais plutôt une tendance qui m'inquiète beaucoup à trouver le bon problème qui justifie une (bonne ou fausse) idée, plutôt que de se concentrer sur une étude théorique plus poussée de celle-ci. J'espère que nous n'avons pas déjà sanctionné une de ces vraies bonnes idées

par ce comportement, mais surtout, j'espère que nous apprendrons à valoriser toutes les idées qui font progresser notre compréhension du domaine, incluant des résultats parfois négatifs !

J'ai observé, beaucoup, et participé, un peu, au travail plein d'attention et de bienveillance du comité de programme de ces quinzièmes JFPC. Dure période pour nous que le printemps, coincés entre IJCAI et CP, je remercie chaleureusement l'action de ce comité qui a pris le temps de fournir des retours constructifs aux articles en cours de maturation. Souhaitons que parmi eux, une belle et grande idée émerge ! Trente papiers soumis cette année, 28 retenus. Soyons clair même si la période est peu propice, c'est trop peu. À chacun de se mobiliser pour systématiser le réflexe de soumissions aux JFPC et venir discuter de son point de vue. À nous, membres de l'AFPC [1], de trouver le ou les formats qui permettront de mobiliser plus et avec plus de facilité.

Passons maintenant au programme des réjouissances pour cette édition 2019 ! Trois invités de renom sont venus nous raconter chacun une histoire :

— Jean-Charles Régin qui nous a montré comment la PPC, augmentée de structures de données originales, permet de proposer des solutions simples à des problèmes complexes de génération automatique de texte ou de musique dans le style de l'auteur.

— Joao Marques-Silva qui est venu présenter un sujet très actif pour les méthodes d'apprentissage, à savoir « *l'explicabilité* ». Il nous a montré comment les approches de contraintes peuvent faciliter ce travail.

— Jean-Marc Astesana nous a présenté des approches centrées SAT et échantillonnage pour résoudre des problèmes de création d'ensembles de véhicules respectant des contraintes de gamme et de taux d'équipements fixés par des prévisionnistes.

Les sessions de la journée du jeudi ont été en grande partie consacrées aux « *jeunes chercheurs* ». Chaque participant a pu voter pour la présentation qu'il a jugée la plus pertinente. L'objectif était de pouvoir valoriser et récompenser un travail prometteur tant sur le fond que sur la forme.

Je terminerai ces quelques mots par un grand merci à tout le comité d'organisation présidé par Elise Vareilles. D'une redoutable efficacité, ce comité me ferait presque dire qu'organiser une conférence est une affaire facile, ou une affaire d'expérience ! Plus sérieusement, le travail effectué a été d'une précision chirurgicale sous le bistouri d'Elise épaulée par l'implication sans faille de Paul Gaborit. IMT Mines Albi est heureuse et fière de vous accueillir dans son beau pays, il ne vous reste qu'à venir profiter de ces quinzièmes Journées Francophones de la Programmation par Contraintes !

Xavier Lorca
Juin 2019

1. Association Française pour la Programmation par Contraintes : http://afpc.greyc.fr/web/

# Comité scientifique 2018

## Président

**Xavier Lorca**
IMT Mines Albi

## Membres du comité

| | |
|---|---|
| David Allouche | INRA Toulouse |
| Gilles Audemard | CRIL |
| Rémi Coletta | TellMePlus |
| Simon De Givry | INRA Toulouse |
| Andres Felipe Barco Santa | Pontificia Univ. Colombie |
| Paul Gaborit | IMT Mines Albi |
| Gaël Glorian | CRIL |
| Arthur Godet | Cosling |
| Emmanuel Hébrard | LAAS-CNRS |
| Ratheil Houndji | Université d'Abomey-Calavi, Bénin |
| Marie-José Huguet | INSA Toulouse, LAAS-CNRS |
| Narendra Jussien | IMT Mines Albi |
| Dimitri Justeau | CIRAD AMAP |
| Roger Kameugne | Université de Louvain |
| Arnaud Lallouet | Huawei |
| Nadjib Lazaar | LIRMM |
| Christophe Lecoutre | CRIL |
| Giovanni Lo Bianco | IMT Atlantique |
| Samir Loudni | Université de Caen |
| Margaux Nattaf | INP Grenoble |
| Marie Pelleau | I3S |
| Cédric Piette | CRIL |
| Charles Prud'homme | IMT Atlantique |
| Frédéric Saubion | Université d'Angers |
| Pierre Schaus | Université de Louvain |
| Mohamed Siala | LAAS-CNRS |
| Laurent Simon | LABRI |
| Christine Solnon | INSA Lyon |
| Pierre Talbot | Université de Nantes |
| Élise Vareilles | IMT Mines Albi |
| Hélène Verhaeghe | Université de Louvain |
| Philippe Vismara | LIRMM |

# Comité d'organisation 2019

## Présidente

**Élise Vareilles**
IMT Mines Albi

## Co-présidents

Michel Aldanondo ..... IMT Mines Albi
Paul Gaborit ..... IMT Mines Albi
Narendra Jussien ..... IMT Mines Albi
Xavier Lorca ..... IMT Mines Albi

## Membres du comité

David Allouche ..... INRA Toulouse
Frédérick Benaben ..... IMT Mines Albi
Marlène Boval ..... IMT Mines Albi
Hélène Fargier ..... IRIT
Isabelle Fournier ..... IMT Mines Albi
Simon de Givry ..... INRA Toulouse
Didier Gourc ..... IMT Mines Albi
Delphine Guillon ..... IMT Mines Albi
Emmanuel Hébrard ..... LAAS-CNRS
Marie-José Huguet ..... LAAS-CNRS, INSA
Claude Laffore ..... IMT Mines Albi
Matthieu Lauras ..... IMT Mines Albi
Pierre Lopez ..... LAAS-CNRS
Frédéric Maris ..... IRIT, Université Toulouse 3 - Paul Sabatier
François Marmier ..... IMT Mines Albi
Aurélie Montarnal ..... IMT Mines Albi
Thomas Schiex ..... INRA Toulouse
Mohamed Siala ..... LAAS-CNRS, INSA

# Organisateurs et soutiens

# Conférences invitées

## Génération automatique de texte et de musique

Dans cet exposé nous présenterons des résultats récents sur la génération automatique de texte et de musique. Nous commencerons par expliquer quelques mécanismes de base en IA pour engendrer des textes/musique dans la style d'un auteur, puis nous montrerons comment le développement de structures de données envisageables grâce à l'augmentation de la mémoire disponible, comme les Multi-Valued Decision Diagram, permettent de proposer des solutions simples à des problèmes complexes. Nous montrerons clairement l'intérêt de la programmation par contraintes dans cette approche.

**Jean-Charles Régin**

## eXplainable AI: An Emerging Application of Constraint Programming

The practical successes of Machine Learning (ML) in different settings motivates the ability of computing small explanations for predictions made. Small explanations are generally accepted as easier for human decision makers to understand. Existing work on computing explanations is based on heuristic approaches, providing no guarantees of quality, in terms of how close such solutions are from cardinality- or subset-minimal explanations. This talk describes a novel constraint-agnostic approach for computing explanations for any Machine Learning (ML) model. The proposed solution exploits abductive reasoning, and imposes the requirement that the ML model be represented as sets of constraints using some target constraint reasoning system, for which the decision problem can be answered with some oracle. The talk also overviews recent work on exploiting constraint-based solutions for assessing the quality of heuristic explanation approaches.

**Joao Marques-Silva**

## Renault-Nissan-Mitsubishi : SAT et échantillonnage

La gamme de véhicules vendables par l'alliance Renault-Nissan-Mitsubishi n'est pas énumérable. Elle est donc représentée en intention et de nombreux problèmes (configuration, contrôle de cohérence documentaire, etc.) se ramènent alors au problème SAT.
Certains problèmes liés à l'activité prévisionnelle nécessitent la création d'ensembles de véhicules respectant les contraintes de la gamme et les taux d'équipements fixés par les prévisionnistes. Cette génération nécessite une approche plus originale.
La présentation abordera les problèmes de type SAT ainsi que les techniques utilisées pour la génération d'échantillons prévisionnels, avec la mise en lumière de leurs forces et leurs faiblesses.

**Jean-Marc Astesana**

# Clustering descriptif : formulations en PLNE et en PPC et applications

**Thi-Bich-Hanh Dao[1], Chia-Tung Kuo[2], S. S. Ravi[34], Christel Vrain[1], Ian Davidson[2]**

[1] LIFO, University of Orléans, France
[2] University of California, Davis
[3] Virginia Tech [4] University at Albany – SUNY

**Résumé**

Cet article résume le travail publié dans [3] et introduisant la notion de clustering descriptif. Nous explorons ici un cadre où les données sont décrites non seulement par des attributs numériques mais aussi par un ensemble de tags discrets et interprétables. Nous formulons le problème de **clustering descriptif** comme un problème d'optimisation bi-objectif : obtenir des clusters compacts relativement aux attributs numériques, associés à une description pertinente en termes de tags. Nous présentons des formulations en PLNE et en PPC et montrons qu'elles peuvent être intégrées dans un algorithme itératif standard permettant de calculer le front de Pareto. Les résultats préliminaires sur des données réelles montrent l'intérêt de notre approche par rapport aux méthodes standards de clustering.

## 1 Introduction

Le clustering est une tâche essentielle en fouille de données qui vise à organiser les objets en classes (clusters). Souvent utilisée dans une phase exploratoire, il est important de pouvoir décrire les clusters obtenus afin de faciliter leur interprétation. Lorsque les attributs décrivant les données sont binaires, des méthodes de clustering conceptuel permettant d'apprendre des concepts (classe et description associée) peuvent s'appliquer. Cependant dans de nombreux domaines qui impliquent des images, des graphes ou d'autres données complexes, les attributs sont par nature numériques et l'utilisation seule d'attributs interprétables n'est plus suffisante. Dans ce papier, nous explorons le scénario où nous cherchons un clustering de données décrites par des attributs numériques (ou une matrice de distance) et étiquetées par des descripteurs (étiquettes, tags) discrets. Les clusters devront donc être compacts du point de vue de la distance mais aussi interprétables en terme de tags. Par exemple on peut souhaiter trouver non seulement des communautés dans un réseau social, mais aussi décrire chaque communauté par les propriétés (étiquettes) de ses membres. A ces fins, la méthode la plus classique est de trouver un clustering puis de décrire les clusters par les étiquettes communes aux objets d'un même cluster. Cependant puisque ces dernières ne sont pas prises en compte lors de la phase de clustering, le résultat peut être pauvre.

Nous proposons une approche qui cherche simultanément des clusters compacts relativement à la distance (par exemple distance dans un graphe pour des réseaux sociaux ou calculée à partir d'attributs SIFT pour des images) et descriptifs par rapport aux tags. Le problème de clustering descriptif est formulé par un problème d'optimisation bi-objectif, avec deux types de critères, ceux définissant la qualité du clustering en terme de distance et et ceux définissant la qualité de la description en tags des clusters.

## 2 Formulation du clustering descriptif

Considérons $n$ données représentées par une matrice $X$ d'attributs de taille $n \times f$ et une matrice $D$ de descripteurs booléens de taille $n \times r$. Nous souhaitons une partition des données en $k$ clusters. Nous définissons deux matrices de variables booléennes : $Z$ de taille $n \times k$, avec $Z_{ic} = 1$ si $i$ est dans le cluster $c$, et $S$ de taille $k \times r$, où $S_{cp} = 1$ signifie que l'étiquette $p$ est présente dans la description de $c$.

**Contraintes.** Nous définissons des contraintes pour exprimer que les clusters doivent former une partition

et expliciter leur description. Par exemple on peut poser les contraintes suivantes sur la description attendue des clusters ($\alpha$, $\beta$ paramètres donnés) :

$$\begin{array}{ll} \text{(C6)} & \forall c = 1, \ldots, k,\ \forall i = 1, \ldots, n, \\ & Z_{ic} = 1 \Longrightarrow \sum_{p=1}^{r} S_{cp}(1 - D_{ip}) \leq \alpha \\ \text{(C7)} & \forall c = 1, \ldots, k,\ \forall p = 1, \ldots, r, \\ & S_{cp} = 1 \iff \sum_{i=1}^{n} Z_{ic}(1 - D_{ip}) \leq \beta \end{array}$$

(C6) indique que chaque instance d'un cluster doit satisfaire *la plupart* des tags le décrivant (jusqu'à $\alpha$ exceptions). (C7) demande qu'un tag soit dans la description d'un cluster si et seulement si *la plupart* de ses instances (jusqu'à $\beta$ exceptions) vérifient ce tag. Ces contraintes tolèrent un désaccord entre objets et propriétés, utiles lorsque les étiquettes sont éparses ou bruitées. Pour les données denses on peut choisir $\alpha = \beta = 0$.

**Objectifs.** La compacité peut être définie par des critères classiques (par exemple le diamètre maximal des clusters ou la somme des distances intra-cluster). En terme de description, nous avons introduit trois objectifs correspondant à différents niveaux de relaxation suivant que les étiquettes soient denses ou bruitées :

- Max-Min Complete Tag Agreement (MMCTA) : maximiser la taille minimale des descriptions des clusters en imposant que les instances d'un même cluster partagent tous les tags de sa description ;
- Minimize Tag $\alpha$-$\beta$ Disagreement (MTD) : des exceptions sont autorisées mais le nombre d'exceptions doit être minimisé ;
- Max-Min Neighborhood Agreement (MMNA) : chaque instance partage au moins $q$ tags avec une autre instance du même cluster et le paramètre $q$ est à maximiser.

**Le problème de clustering descriptif** est formulé comme un problème d'optimisation bi-objectif, avec un objectif sur la compacité et un objectif sur la description du clustering, sous les contraintes de partition et de description. Ces objectifs ne sont pas compatibles et nous cherchons donc le front de Pareto qui réalise un compromis entre eux. Pour chercher ces solutions, nous utilisons un algorithme qui optimise successivement un objectif sous une contrainte exprimée sur l'autre objectif [2].

**Formulations en PLNE et en PPC.** Le problème peut être présenté par un programme linéaire en nombres entiers, car la formulation des contraintes et des fonctions objectif est ou peut-être transformée en contraintes linéaires. Nous présentons ici une formulation en programmation par contraintes. On définit en plus pour chaque instance $i$ une variable entière $G_i$ qui indique le cluster contenant $i$. Le modèle utilise la contrainte *precede* [4] pour casser les symétries entre clusters et les contraintes développées dans [2] pour modéliser le critère de compacité du clustering. Les contraintes sur les descriptions, comme (C6-C7) sont exprimées par des contraintes réifiées.

Pour les critères MMCTA et MMNA, nous pouvons remarquer que lorsque le domaine de la fonction objectif $q$ est $[\underline{q}, \overline{q})$, deux instances partageant moins de $\underline{q}$ tags ne peuvent pas être dans le même cluster. Nous avons ainsi développé une contrainte qui assure la relation :

$$\forall i, j = 1, .., n,\ G_i = G_j \Longrightarrow \sum_{p=1}^{r} D_{ip} D_{jp} \geq q$$

Cette contrainte révise la borne supérieure $\overline{q}$ en utilisant les instances déjà affectées aux clusters, et aussi propage $G_i \neq G_j$ pour toute paire d'instances $i, j$ qui partagent moins de $\underline{q}$ tags. On maintient une consistance de borne pour $q$ et une consistance de domaine partielle pour les variables d'affectation des instances.

Notre formulation diffère des approches récentes utilisant la PLNE [5] ou la PPC [1] pour le clustering conceptuel. Elles reposent sur une recherche préalable des concepts au sens de l'analyse formelle des concepts, et ne peuvent donc pas s'appliquer à notre cadre où nous disposons de deux niveaux de représentation des objets (distance et étiquettes) et où nous relâchons la contrainte qu'une description soit un concept.

**Évaluation.** Des évaluations ont été menées sur des données réelles. Elles montrent que les solutions de Pareto fournissent des clusters à différents niveaux de descriptions, ce qui semble plus pertinent par rapport aux méthodes classiques de clustering mono-objectif. Elles montrent également que la capacité à renforcer les contraintes et des stratégies de recherche adaptées permettent d'améliorer l'efficacité de la PPC.

## Références

[1] M. Chabert et C. Solnon : Constraint programming for multi-criteria conceptual clustering. *In CP 2017*, pages 460–476, 2017.

[2] T.-B.-H. Dao, K.-C. Duong et C. Vrain : Constrained clustering by constraint programming. *Artificial Intelligence*, 244:70–94, 2017.

[3] T.-B.-H. Dao, C.-T. Kuo, S. S. Ravi, C. Vrain et I. Davidson : Descriptive clustering : ILP and CP formulations with applications. *In IJCAI 2018*, pages 1263–1269, 2018.

[4] Y. C. Law et J. H.-M. Lee : Global constraints for integer and set value precedence. *In CP 2004*, pages 362–376, 2004.

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

# Décompositions structurelles et sémantiques Rapport préliminaire*

**Philippe Jégou** **Cyril Terrioux**
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
{philippe.jegou, cyril.terrioux}@lis-lab.fr

**Résumé**

Dans le cas des modèles graphiques, même les plus simples comme les CSP (réseaux de contraintes), les décompositions généralement proposées dans la littérature s'intéressent pour l'essentiel à la structure. Cela étant, afin de mieux appréhender un problème dans sa globalité, il serait souhaitable aussi de prendre en compte sa sémantique, au sens des domaines des variables, et des relations de compatibilité qui leur sont associées via les contraintes. Dans cette note, il est proposé une nouvelle approche de la problématique qui prend justement en compte ces deux aspects pour définir une nouvelle forme de décomposition et son paramètre associé. L'une des ambitions d'une telle démarche et qui peut d'ailleurs constituer un réel défi pour la communauté, est de dépasser les limites actuellement posées par les décompositions arborescentes de graphes de *tree-width* (ou largeur) bornée. On montre ainsi que le paramètre que nous proposons minore systématiquement la tree-width d'un réseau de contraintes binaires (cf. théorème 1) qui constitue "la" borne fondamentale actuelle pour le traitement de nombreux problèmes combinatoires, dès lors qu'ils s'expriment en termes de graphes et qu'il s'agit de les résoudre par des approches fondées sur leur structure. Toutefois, si ce premier résultat semble démontrer la pertinence d'une telle approche, cette contribution doit être considérée comme une contribution très préliminaire à une éventuelle nouvelle voie de recherches.

## 1 Introduction et notations

L'objectif ici est de proposer une nouvelle forme de décomposition qui prend en compte, outre la structure, les caractéristiques sémantiques des modèles graphiques considérés. Dans ce cadre, l'aspect structurel repose sur une décomposition de graphe (mais aussi d'hypergraphe) qui n'est pas nécessairement arborescente. Cette décomposition permet de définir un graphe de clusters (un cluster étant un sous-ensemble de variables et définissant ainsi un sous-problème). Sur cette base, en considérant la résolution de chaque sous-problème induit par chaque cluster, on peut définir une instance de CSP binaire (ou de méta-CSP binaire) dont chaque variable correspond à un cluster dont le domaine est défini par les solutions locales associées à ce cluster. L'aspect sémantique va également considérer l'agencement des clusters, non pas au niveau structurel, mais au niveau sémantique. L'idée de base est d'exploiter au niveau du méta-CSP binaire les classes polynomiales de CSP définies en termes sémantiques. Il en existe un certain nombre [2], mais nous illustrons cette possibilité avec la classe polynomiale hybride $\mathcal{BTP}$ [3], ce qui permet déjà de proposer une décomposition dont la largeur sera toujours inférieure ou égale à la tree-width. Cela étant, toute classe polynomiale de nature sémantique est *a priori* envisageable, en repoussant la question de la reconnaissance, i.e. du méta-problème qui peut ouvrir sur de nombreuses difficultés.

Nous rappelons la définition d'un CSP d'arité quelconque, et donc les notations utilisées ici. Un CSP $P$ sera noté par un triplet $(X, D, C)$ avec $X$ l'ensemble des variables que l'on note $\{x_1, ..., x_n\}$ avec donc $n$ le nombre de variables. $D = \{D_{x_1}, ..., D_{x_n}\}$ représente l'ensemble des domaines avec $D_{x_i}$ le domaine de la variable $x_i$ ($d$ notera la taille maximum des domaines). $C$ désigne l'ensemble des contraintes $\{c_1, ..., c_e\}$ avec $e$ le nombre des contraintes. Chaque contrainte porte sur un ensemble de $r$ variables au maximum. Cet ensemble est appelé scope (ou portée) d'une contrainte et il est noté $S(c_i)$. Si l'arité d'une contrainte est égale à 2, la contrainte est dite

*Ce travail est soutenu par l'Agence Nationale de la Recherche dans le cadre du projet DEMOGRAPH (ANR-16-C40-0028)

binaire et on notera par $c_{ij}$ la contrainte qui porte sur $x_i$ et $x_j$. Un CSP dont toutes les contraintes sont binaires est dit binaire. À chaque contrainte $c_i$, on associe une relation $R(c_i)$ qui définit l'ensemble des valeurs, des tuples donc, qui peuvent être affectées simultanément aux variables de $S(c_i)$ pour la satisfaire.

## 2 Une décomposition alliant structure et propriétés sémantiques

Nous donnons maintenant la définition de la nouvelle décomposition. Cette définition s'appuie sur une décomposition structurelle en clusters qui n'est pas nécessairement arborescente (comme dans [5]) couplée avec une classe polynomiale de CSP notée $\mathcal{C}$ (pour *tractable Class*). Cela étant, autoriser une décomposition en clusters peut conduire à une explosion combinatoire car le nombre de ceux-ci peut potentiellement être exponentiel en le nombre de sommets du graphe considéré (jusqu'au nombre d'éléments figurant dans une frontière du treillis des parties de l'ensemble des sommets). Aussi, contrairement à [5], et pour conserver la capacité à définir des méthodes de résolution de complexité polynomiale, le nombre de clusters sera paramétré par une constante $k$ correspondant au degré du polynôme associé à ce nombre. Nous parlerons donc ainsi de $\mathcal{C}$-$k$-décompositions. Nous verrons qu'il existe d'autres points de divergence d'avec l'approche proposée dans [5], tant au niveau de la nature de la connectivité dans la décomposition qu'au niveau de la connexité interne imposée dans [5] au niveau des clusters mais non requise ici.

La suite de cette note considère aussi bien les CSP binaires (graphes de contraintes) que les CSP d'arité quelconque (hypergraphes de contraintes). Cela étant, même dans le cas des hypergraphes de contraintes, il est possible de considérer des décompositons de graphe (cf. par exemple [5]). Pour cela, il suffit de considérer la notion de *2-section* [1] également appelée *graphe primal* dans la communauté CSP.

Puisque l'objectif est de définir des décompositions de modèles graphiques, ici simplement de CSP, les décompositions devront prendre en compte les solutions associées aux instances. Cela passe par la conservation, dans l'objet décomposé, de l'ensemble des solutions de l'objet originel. Pour garantir cette condition, nous rappelons la définition de problème *dual* :

**Définition 1 (Dual d'une instance CSP)** *Le* dual d'une instance de CSP $P = (X, D, C)$ *est un CSP binaire* $Dual(P) = (X^{Du}, D^{Du}, C^{Du})$ *où chaque variable* $X_i^{Du} \in X^{Du}$ *correspond à* $S(c_i)$ *et son domaine est défini par les tuples de la relation* $R(c_i)$, *et une contrainte* $c_{ij}^{Du} \in C^{Du}$ *relie deux variables* $X_i^{Du}$ *et* $X_j^{Du}$ *si celles-ci partagent au moins une variable d'origine, i.e. si* $S(c_i) \cap S(c_j) \neq \emptyset$. *La relation de compatibilité* $R(c_{ij}^{Du})$ *est définie par les couples de tuples* $(t_i, t_j) \in D_i^{Du} \times D_j^{Du}$ *tels que* $t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]$ *(la notation* $t[\ldots]$ *désigne l'opérateur de projection).*

On peut démontrer très facilement qu'il existe une bijection entre l'ensemble des solutions du CSP originel et son dual car l'affectation des variables d'une solution correspond tout simplement à une collection de tuples (un par contrainte et donc relation) qui sont tous compatibles deux à deux (pour la satisfaction des contraintes binaires de $Dual(P)$) [4]. Dans [4], il est également démontré que ce CSP dual peut posséder des contraintes (binaires) redondantes et que celles-ci peuvent donc être supprimées, tout en conservant une instance équivalente (au sens de l'existence d'une bijection entre ensembles de solutions). On parle alors d'*intergraphe de contraintes* associé à $P$, ou plutôt ici de *dual partiel* d'une instance de CSP :

**Définition 2 (Dual partiel d'une instance CSP)** *Étant donné un CSP d'arité quelconque* $P = (X, D, C)$ *et son dual* $Dual(P) = (X^{Du}, D^{Du}, C^{Du})$, *un* Dual Partiel *de* $P$ *est un CSP binaire noté* $Dual_p(P) = (X^{Du}, D^{Du}, C^{Du_p})$ *où* $C^{Du_p} \subseteq C^{Du}$ *et pour tout couple* $X_i^{Du}$ *et* $X_j^{Du}$ *tel que* $S(c_i) \cap S(c_j) \neq \emptyset$, *il doit exister un chemin dans le graphe de contraintes associé à* $Dual_p(P)$, *tel que pour toute variable* $X_\ell^{Du}$ *figurant sur ce chemin de* $X_i^{Du}$ *à* $X_j^{Du}$, *alors* $X_i^{Du} \cap X_j^{Du} \subseteq X_\ell^{Du}$.

Dans [4], il est montré que le calcul d'un dual partiel d'une instance de CSP dont le nombre de contraintes est minimum est réalisable en temps polynomial.

Avant de définir les $\mathcal{C}$-$k$-décompositions, il nous faut introduire une autre notion, celle de *Sur-Dual* d'une instance de CSP. Intuitivement, étant donné un CSP d'arité quelconque, un Sur-Dual va consister en un CSP binaire qui recouvre l'ensemble de variables d'un CSP par des clusters (il s'agit donc d'une décomposition), et dont l'ensemble de solutions est le même (à une bijection près).

**Définition 3 (Sur-Dual d'une instance CSP)** *Un* Sur-Dual d'une instance de CSP $P = (X, D, C)$ *est un CSP binaire* $SD(P) = (X^{SD}, D^{SD}, C^{SD})$ *où :*

- $X^{SD}$ *est une famille de sous-ensembles de* $X$ *que l'on appelle clusters*
- $\forall i \neq j$, *avec* $X_i^{SD}, X_j^{SD} \in X^{SD}$, *on a* $X_i^{SD} \not\subseteq X_j^{SD}$ *et* $X_j^{SD} \not\subseteq X_i^{SD}$
- $\forall c_j \in C, \exists X_i^{SD} \in X^{SD}$ *tel que* $S(c_j) \subseteq X_i^{SD}$

- *tout domaine* $D_i^{SD} \in D^{SD}$ *est l'ensemble des solutions* $s_i \in \times_{x_j \in X_i^{SD}} D_{x_j}$ *du sous-problème de* $P$ *induit par les variables* $x_j$ *de* $P$ *telles que* $x_j \in X_i^{SD}$ *et par les contraintes* $c_j$ *de* $P$ *telles que* $c_j \subseteq X_i^{SD}$.
- *les contraintes* $c_{ij}^{SD} \in C^{SD}$ *relient deux variables* $X_i^{SD}$ *et* $X_j^{SD}$ *si celles-ci partagent au moins une variable de l'instance du CSP d'origine, i.e. si* $X_i^{SD} \cap X_j^{SD} \neq \emptyset$.
- *les relations de compatibilité* $R(c_{ij}^{SD})$ *sont définies par les couples* $(s_i, s_j) \in D_i^{SD} \times D_j^{SD}$ *tels que* $s_i[X_i^{SD} \cap X_j^{SD}] = s_j[X_i^{SD} \cap X_j^{SD}]$

On peut constater d'une part que tout dual est un sur-dual, et d'autre part, avec le même procédé que dans [4], qu'il existe une bijection entre l'ensemble des solutions du CSP originel et tout sur-dual qui peut lui être associé. De plus, comme pour le cas d'un dual, on peut définir la notion de *Sur-Dual Partiel* d'une instance de CSP, en supprimant les arêtes redondantes, sous la réserve de conserver les mêmes propriétés de connectivité dans le graphe que pour le cas d'un dual partiel i.e. l'existence d'un chemin dans le graphe de contraintes associé tel que pour toute variable $X_\ell^{SD}$ figurant sur ce chemin de $X_i^{SD}$ à $X_j^{SD}$, alors $X_i^{SD} \cap X_j^{SD} \subseteq X_\ell^{SD}$. Un Sur-Dual Partiel de $P$ sera noté $SD_p(P)$ et il sera de la forme $SD_p(P) = (X^{SD}, D^{SD}, C^{SD_p})$ avec $C^{SD_p} \subseteq C^{SD}$ si le Sur-Dual dont il est issu est $SD(P) = (X^{SD}, D^{SD}, C^{SD})$.

On récapitule les liens (i.e. équivalences) entre ensembles de solutions de ces différentes représentations d'instances dans la propriété suivante qui est très simple à démontrer :

**Proposition 1** *Étant donné un CSP d'arité quelconque* $P = (X, D, C)$ *:*
- *les ensembles de solutions de Dual*$(P)$ *et de tout dual partiel de* $P$ *sont égaux*
- *les ensembles de solutions de tout Sur-Dual de* $P$ *et de tout Sur-Dual Partiel de* $P$*sont égaux*
- *les ensembles de solutions de* $P$ *et de Dual*$(P)$ *sont en bijection*
- *les ensembles de solutions de* $P$ *et de tout Sur-Dual (éventuellement Partiel) de* $P$ *sont en bijection*

Cette question d'équivalence entre l'ensemble des solutions d'une instance originelle $P$ et celui d'un Sur-Dual (éventuellement Partiel) de $P$ est requise pour s'assurer de la validité de la décomposition associée à ce Sur-Dual (éventuellement Partiel). On peut remarquer que si l'on considère une décomposition en clusters comme celle proposée dans [5], il n'existe alors plus de garantie, quand bien même ces décompositions incluent les décompositions arborescentes. En effet, comme pour les décompositions arborescentes, la connectivité requise pour disposer d'une décomposition au sens de [5] impose qu'il existe au moins un chemin dans le graphe des clusters, tel que si deux clusters partagent une variable du graphe de départ (ici le CSP originel), alors cette variable figure sur tous les clusters de ce chemin, comme c'est d'ailleurs le cas pour les décompositions arborescentes. Cela étant, pour les décompositions arborescentes, cette condition est équivalente à l'existence d'un chemin contenant toutes les variables appartenant à l'intersection des deux clusters puisque le graphe considéré par la décomposition est un arbre. Par contre, si on prend par exemple $X = \{x_1, x_2, x_3, x_4, x_5, x_6\}$ avec 4 clusters $E_1 = \{x_1, x_2, x_3\}$, $E_2 = \{x_1, x_2, x_4\}$, $E_3 = \{x_1, x_5\}$ et $E_4 = \{x_2, x_6\}$ et une décomposition dont les arêtes sont $\{E_1, E_3\}$, $\{E_1, E_4\}$, $\{E_2, E_3\}$ et $\{E_2, E_4\}$, il existera bien un chemin de $E_1$ à $E_2$ contenant $x_1$, celui passant par $E_3$, et un chemin de $E_1$ à $E_2$ contenant $x_2$, celui passant par $E_4$. Mais il n'en existera aucun contenant simultanément $x_1$ et $x_2$ alors que $E_1 \cap E_2 = \{x_1, x_2\}$. L'absence de ce chemin ne permet alors pas la conservation de la cohérence au niveau CSP, entre les clusters $E_1$ et $E_2$. Mais elle serait garantie dans une décomposition arborescente car l'arête $\{E_1, E_2\}$ serait imposée par la structure d'arbre.

La définition des $\mathcal{C}$-$k$-décompositions peut maintenant être présentée :

**Définition 4** *Étant donnée une instance CSP d'arité quelconque* $P = (X, D, C)$*, une constante entière* $k$*, et une classe polynomiale de CSP notée* $\mathcal{C}$*, une* $\mathcal{C}$-$k$-décomposition *de* $P$ *est un graphe* $G = (X^{SD}, F)$ *associé à* $SD_p(P) = (X^{SD}, D^{SD}, C^{SD_p})$ *qui est un sur-dual partiel de* $P$ *et qui vérifie :*

1. $|X^{SD}| \leq n^k$*, avec* $n = |X|$*,*
2. $F$ *est l'ensemble d'arêtes associées aux portées des contraintes binaires figurant dans* $C^{SD_p}$*,*
3. $SD_p(P) = (X^{SD}, D^{SD}, C^{SD_p})$ *appartient à la classe* $\mathcal{C}$*.*

*La* largeur *d'une* $\mathcal{C}$-$k$-*décomposition est égale à* $max_{X_i^{SD} \in X^{SD}} |X_i^{SD}| - 1$*. La* $\mathcal{C}$-$k$-largeur $b$ *de* $P$ *est la largeur minimum pour toutes les* $\mathcal{C}$-$k$-*décompositions de* $P$*.*

On peut remarquer que si $G = (X^{SD}, F)$ est un arbre, il s'agit tout simplement d'une décomposition arborescente du graphe 2-section associé au CSP $P$.

Si on considère une $\mathcal{C}$-$k$-décomposition d'une instance $P$ de CSP, sa résolution est réalisable en un temps de l'ordre de $n^k d^{b+1}$ pour la première phase pour le calcul de $SD_p(P)$ et donc notamment la résolution des clusters de $X^{SD}$ produisant $D^{SD}$, suivi d'un temps polynomial en $d^{b+1}$ pour la phase consistant en l'exploitation de la classe polynomiale $\mathcal{C}$.

## 3 Une illustration avec la classe BTP

Pour illustrer cette décomposition, on peut considérer la classe polynomiale *BTP*. Ainsi, si on a $\mathcal{C} = \mathcal{BTP}$, nous montrons que l'on dispose d'une décomposition dont la largeur est inférieure à la treewidth du CSP. Nous rappelons d'abord la définition de la classe $\mathcal{BTP}$ et ses propriétés utiles ici.

**Définition 5 (Broken-Triangle Property [3])** *Un CSP binaire $P$ satisfait la Broken Triangle Property (BTP) par rapport à un ordre sur les variables $<$ si, pour tout triplet de variables $(x_i, x_j, x_k)$ tel que $x_i < x_j < x_k$, si $(v_i, v_j) \in R(c_{ij})$, $(v_i, v_k) \in R(c_{ik})$ et $(v_j, v'_k) \in R(c_{jk})$, alors soit $(v_i, v'_k) \in R(c_{ik})$, soit $(v_j, v_k) \in R(c_{jk})$.*

La propriété BTP permet de définir une classe polynomiale sur les instances de CSP binaires, classe que nous noterons $\mathcal{BTP}$. D'une part, les instances de cette classe peuvent être reconnues en temps polynomial en $O(ned^3)$ (cf. Corollaire 7.5 de [3]), et d'autre part, elle peuvent être résolues en temps polynomial en $O(nd^2)$ (cf. Théorème 3.1 de [3]).

Notons que BTP englobe un certain nombre de classes polynomiales existantes, certaines de nature sémantiques, et une autre, d'origine structurelle. En particulier, la proposition 4.5 de [3] montre que pour tout CSP binaire arborescent, il existe un ordre qui vérifie BTP.

Cette propriété permet de justifier l'approche proposée par la $\mathcal{C}$-$k$-décomposition. En effet, dans le théoreme qui suit, nous montrons qu'elle offre une décomposition qui est systématiquement meilleure que la décomposition arborescente, en ce sens que si la classe polynomiale considérée est $\mathcal{BTP}$, i.e. si $\mathcal{C} = \mathcal{BTP}$, on dispose alors d'une décomposition dont la largeur est toujours inférieure à la treewidth du CSP :

**Théorème 1** *Pour tout CSP binaire $P = (X, D, C)$, on a alors $b \leq w$ avec $w$ la treewidth du CSP binaire, et $b$, la $\mathcal{C}$-1-largeur de $P$ avec $\mathcal{C} = \mathcal{BTP}$.*

**Preuve 1** *On remarque au préalable que pour le cas des décompositions arborescentes, le nombre de clusters peut être majoré par $n$ (sous l'hypothèse où aucun cluster n'est inclus dans un autre cluster). Ainsi, on peut fixer $k = 1$. Ensuite, il suffit de constater que pour toute instance de CSP binaire $P$, on dispose d'une $\mathcal{C}$-1-décomposition pour laquelle le graphe $G = (X^{SD}, F)$ est une décomposition arborescente de l'instance $P$, car tout CSP binaire arborescent appartient à la classe $\mathcal{BTP}$. On en déduit que la $\mathcal{C}$-1-largeur de $P$ est au plus égale à $w$.*

## 4 Discussion et conclusion

Si le théorème 1 peut inciter à l'optimisme dans la démarche qui fonde ce travail, cette proposition de décomposition alliant structure et propriétés sémantiques que constitue la $\mathcal{C}$-$k$-décomposition conduit cependant à plusieurs questions ouvertes.

Déjà concernant la complexité du calcul de ces décompositions, la tâche est conséquente car il s'agit d'analyser les différentes classes polynomiales de CSP de nature sémantique, pour étudier la question en profondeur et il en existe un certain nombre [2]. Cela étant, en se limitant à $\mathcal{BTP}$, il a été montré que pour une instance de CSP binaire, calculer un sous-ensemble de variables $S \subseteq X$ de taille maximum dont le sous-problème induit vérifie BTP est NP-difficile (cf. Théorème 9.1 de [3]). Ce résultat semble offrir un premier éclairage sur la complexité de cette question, déjà pour $\mathcal{BTP}$, éclairage qui peut conduire au pessimisme. En effet, si vérifier l'appartenance d'un sur-dual partiel $SD_p(P)$ d'une instance $P$ à la classe $\mathcal{BTP}$ est réalisable en temps polynomial, la question de la détermination, pour une instance $P$, d'un sur-dual partiel idoine peut s'avérer extrêmement combinatoire. Une voie pour contourner cette difficulté peut cependant consister en l'étude de cette question en s'appuyant sur l'hypothèse d'un paramètre $k$ constant.

Deux autres interrogations sont aussi légitimes. Ce type de décomposition peut-il admettre des mises en œuvre effectives, et donc opérationnelles en pratique ? La difficulté qu'il y a eu déjà pour rendre efficaces en pratique les approches basées sur la décomposition arborescente pose ici question. Et au-delà, concernant des modèles graphiques plus riches que les seuls CSP, comme les WCSP par exemple, ce type d'approche est-il seulement envisageable ?

**Remerciements** Nous remercions Mamadou Kanté qui nous a informé de l'existence de l'article [5].

## Références

[1] C. Berge : *Graphes et Hypergraphes.* Dunod-France, 1970.

[2] C. Carbonnel et M. C. Cooper : Tractability in constraint satisfaction problems : a survey. *Constraints*, 21(2):115–144, 2016.

[3] M. Cooper, P. Jeavons et A. Salamon : Generalizing constraint satisfaction on trees : hybrid tractability and variable elimination. *Artificial Intelligence*, 174:570–584, 2010.

[4] P. Jégou : *Contribution à l'étude des problèmes de satisfaction de contraintes : Algorithmes de propagation et de résolution – Propagation de contraintes dans les réseau dynamiques.* Thèse de doctorat, Université des Sciences et Techniques du Languedoc, January 1991.

[5] I. Schiering : A Hierarchical Approach to Monadic Second-Order Logic over Graphs. *In Proceedings of CSL*, pages 424–440, 1997.

# Préserver la confidentialité pour l'algorithme Generalized Distributed Breakout

**Julien Vion[1] René Mandiau[1] Sylvain Piechowiak[1] Marius Silaghi[2]**

1. LAMIH CNRS UMR 8201, Université Polytechnique Hauts de France
2. Florida Institute of Technology

1. {prénom.nom}@uphf.fr 2. msilaghi@fit.edu

**Résumé**

Protéger la vie privée des utilisateurs est l'un des enjeux des systèmes distribués. Dans cet article, nous mesurons la fuite d'informations lors de la résolution de problèmes d'optimisation de contraintes distribués (DCOP). Récemment, Okamoto, Zivan et Nahon [19] ont proposé l'algorithme de Breakout distribué généralisé (GDBA pour Generalized Distributed Breakout Algorithm) pour traiter ce type de problème. Cependant, la mesure et le contrôle des données échangées n'ont jamais été réalisés pour cet algorithme.

Nous étudions le comportement de GDBA lors de la résolution de DCOP et de DCOP utilitaristes, une variante dans laquelle les agents cherchent une solution de qualité tout en protégeant les informations qu'ils possèdent. Nous montrons que GDBA peut être amélioré pour préserver la plupart des informations sur les contraintes et réduire la fuite de données sur les domaines d'un facteur 2 à 3 sans impact significatif sur la qualité des solutions.

**Abstract**

Privacy has traditionally been a major motivation of distributed problem solving. In this paper, we focus on privacy issues when solving Distributed Constraint Optimization Problems (DCOPs). Recently, Okamoto, Zivan et Nahon [19] promoted the Generalized Distributed Breakout Algorithm (GDBA) as a very efficient heuristic strategy to find good solutions to DCOPs.

We study how GDBA behaves when solving Utilitarian DCOPs, where utilitarian agents want to reach an agreement while reducing the privacy loss. We show that GDBA can be improved to allow for extensive handling of constraint privacy, and reduce domain privacy loss by a factor 2~3 with no significant impact on the quality of the solution.

Le problème d'optimisation de contraintes distribué (DCOP pour Distributed Constraint Optimization Problem) est un modèle général en programmation par contraintes (PPC) utilisé pour modéliser et résoudre des problèmes distribués NP-difficiles. Pour résoudre un DCOP, les agents négocient afin de trouver une solution qui satisfait au mieux un ensemble de contraintes. Le respect de la vie privée est un enjeu important dans de nombreuses applications distribuées. En conséquence, en plus de limiter le coût engendré par des contraintes non satisfaites, le coût lié à la fuite d'information devrait également être prise en compte [9, 13, 26]. Par exemple, dans un problème d'ordonnancement distribué, les participants peuvent souhaiter ne pas révéler toutes leurs contraintes et leurs disponibilités. Certaines d'entre elles peuvent être liées à leur vie privée.

Il y a perte de confidentialité dès lors que les agents échangent des données. L'affectation des créneaux horaires peut être difficile si les participants ne révèlent pas leurs contraintes [3, 5, 7]. En pratique, si un agent souhaite contrôler les fuites de données, il peut associer un coût à la révélation de chaque information liée à son problème local. Ce coût peut être pris en compte par un raisonnement orienté utilités [4, 21, 22].

Des algorithmes stochastiques de recherche locale ont été introduits pour traiter les problèmes distribués, notamment Distributed Stochastic Algorithm (DSA) [28] et Generalized Distributed Breakout Algorithm [15, 19, 27] (GDBA). Ils permettent de prendre en compte les utilités relativement facilement, i. e., sans impact sur leur correction. Bien qu'incomplets, ces algorithmes peuvent trouver des solutions sous-optimales de bonne qualité relativement rapidement, dans un contexte *anytime* [25, 29].

Dans cet article, nous étudions les propriétés de GDBA du point de vue de la confidentialité. L'objectif de ce travail est d'améliorer GDBA pour préserver la vie privée de l'agent. Dans les sections suivantes, après

des rappels de notations et de contexte, nous montrons comment la confidentialité des contraintes (section 2) et des domaines (section 3) peuvent être améliorées pour GDBA. Dans la section 4, nous montrons expérimentalement l'impact des coûts des contraintes et de confidentialité des domaines sur des problèmes académiques et réalistes.

## 1 Contexte

Nous traitons des problèmes discrets ; les domaines sont des sous-ensembles finis de $\mathbb{Z}$, bien que n'importe quel ensemble fini dénombrable puisse être utilisé. Nous faisons quelques suppositions simplificatrices supplémentaires, sans perte de généralité : un agent n'encapsule qu'une seule variable, une contrainte implique au plus deux variables, les coûts des contraintes sont dans $\mathbb{N}$ (on pourrait utiliser $\mathbb{R}$ sans autre difficulté que leur représentation informatique, mais les coûts négatifs ne sont pas intuitifs).

**Définition 1.1.** Un problème d'optimisation de contraintes distribué (DCOP) est défini par un quadruplet $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, tel que :

$\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$ est un ensemble de $n$ agents.

$\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ est un ensemble de $n$ variables. Chaque agent $A_i$ encapsule la variable $x_i$.

$\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n\}$ est un ensemble de $n$ domaines, tels que $\forall i, |\mathcal{D}_i| \leq d$. Chaque variable $x_i$ peut être instanciée à une valeur $v \in \mathcal{D}_i$. Une affectation de $X \subseteq \mathcal{X}$ est un ensemble d'instanciations pour chaque variable de $X$.

$\mathcal{C} = \{C_1, C_2, \ldots, C_e\}$ est un ensemble de $e$ constraintes valuées. Chaque contrainte $C_i$ implique les variables $\mathcal{X}(C_i) \subseteq \mathcal{X}$, et définit un coût pour chaque affectation de ses variables. Nous notons $C_i(X)$ le coût de l'affectation de $\mathcal{X}(C_i) \subseteq X$ pour $C_i$. Nous notons également $\mathcal{C}(A_i)$ l'ensemble des contraintes qui impliquent $x_i$, i. e., $\mathcal{C}(A_i) = \{C_j \in \mathcal{C} \mid x_i \in \mathcal{X}(C_j)\}$.

L'objectif du problème est de trouver une solution optimale $S$, i. e., une affectation de $\mathcal{X}$ qui minimise le coût des contraintes $\sum_{i=1}^{e} C_i(S)$.

Les contraintes peuvent être unaires et n'impliquer qu'un seul agent (on parle de contrainte intra-agent), ou binaires et mettre ainsi en jeu deux agents. Ces dernières sont appelées contraintes inter-agent et les deux agents impliqués sont alors voisins. Chaque agent $A_i$ a exactement $N_i$ voisins. Nécessairement, $\sum_{i=1}^{n} N_i$ est égal à deux fois le nombre de contraintes inter-agent du problème.

La *confidentialité* est la propriété qu'ont les agents de garder leurs informations secrètes. C'est un concept assez flou que plusieurs auteurs ont tenté de catégoriser [8, 10, 12]. Pour Grinshpoun [10], la confidentialité des agents peut concerner les domaines, les contraintes, les affectations et les algorithmes. Parmi les trois articles cités ci-dessus, c'est le seul à mentionner la confidentialité des domaines, bien que l'auteur indique qu'il n'est pas possible de la conserver dans le modèle DCOP traditionnel. En effet, Grinshpoun considère que les coûts des contraintes sont représentés en extension, par des matrices qui couvrent tout le produit cartésien des domaines des variables impliquées. Cela implique que les agents ont connaissance des domaines complets de leur voisins avant même que la résolution ne commence. Bien qu'il semble difficile d'implémenter un algorithme de filtrage sans cette connaissance complète des domaines voisins, le modèle de PPC « ouverte » [6] donne des pistes pour le rendre réalisable. Nous rendons possible la non-révélation des domaines lors de l'initialisation par :

— l'utilisation de de contraintes définies *en intention* par une fonction $f_i : \mathbb{Z}^{|\mathcal{X}(C_i)|} \mapsto \mathbb{N}$, i. e., définie sur un sur-ensemble de $\prod_{x_j \in \mathcal{X}(C_i)} \mathcal{D}_j$,

— l'utilisation d'algorithmes stochastiques qui ne révèlent au plus qu'une valeur de domaine par cycle.

*GDBA* by Okamoto, Zivan et Nahon [19] est une généralisation de DBA [15, 27] aux DCOP valués [1]. GDBA est un algorithme stochastique incomplet conçu pour trouver rapidement de bonnes solutions à un DCOP. Tout comme la plupart des algorithmes incomplets, GDBA ne permet pas de prendre en compte les contraintes « dures », i. e., les coûts infinis [2], et ne peuvent pas prouver qu'une solution est optimale. Cependant, déterminer si une solution est optimale reste un problème NP-difficile et GDBA traite bien mieux les problèmes de grande taille que les algorithmes complets à complexité exponentielle comme ABT, DPOP ou Sync-BB [12, 16, 26].

La plupart des travaux sur la confidentialité pour les DCOP se sont focalisés sur la confidentialité des affectations. Mettre en place une confidentialité complète à ce niveau nécessite l'utilisation d'agents médiateurs ou des protocoles cryptés sécurisés sophistiqués [23, 24]. À notre connaissance, aucun de ces travaux ne traite la confidentialité des domaines. Notre approche est donc différente : au lieu de vouloir garder secrètes les affectations, nous considérons les domaines comme secrets. Au cours de la résolution du problème, un agent essaie d'affecter une valeur du domaine à sa variable, et révèle cette affectation. Cela entraine une perte irré-

1. Pour éviter les confusions avec les variantes que nous introduisons dans cet article, nous nommerons parfois cet algorithme "Vanilla GDBA".

2. En pratique, on peut utiliser une grande constante $K$ pour représenter un coût "infini".

médiable de confidentialité. Notre objectif est de minimiser ces fuites d'information.

GDBA est purement décentralisé, chaque agent raisonne localement sans passer par des médiateurs ni échange de sous-problèmes. L'algorithme fonctionne de manière synchrone : chaque agent commence par choisir un valeur aléatoirement, l'affecte à sa variable et envoie cette information à ses voisins *via* un message de type « ok ? ». À chaque étape, chaque agent choisit de manière « gloutonne » la meilleure valeur à affecter du point de vue des coûts engendrés par ses contraintes. Il calcule alors la différence des coûts des contraintes entre la précédente et la nouvelle affectation. Il transmet ce *delta* à ses voisins *via* un message de type « improve ». Une fois qu'un agent a reçu les deltas de tous ses voisins, il ne change sa valeur que si son propre delta est strictement négatif (c'est-à-dire, dans le cadre d'un problème de minimisation, que le changement de valeur entraine une amélioration de la solution) et le meilleur parmi son voisinage, avec une résolution déterministe des égalités. Si aucun agent ne peut améliorer la solution partielle de son voisinage, i. e., aucun delta n'est négatif, alors l'agent considère qu'il est dans un *quasi minimum local*. Il réalise alors un *breakout*, c'est-à-dire que toutes les contraintes non-satisfaites [3] voient leur pondération incrémentée d'une unité. Cette stratégie présente deux propriétés intéressantes : d'une part, en ne permettant qu'à l'agent présentant le meilleur delta de son voisinage de changer sa valeur, elle interdit les « oscillations » qui pourraient survenir si deux voisins changent de valeur simultanément (cf section 2). D'autre part, la pondération des contraintes permet aux agents de sortir des minima locaux [18].

Si nous revenons aux catégories de Grinshpoun [10] : nous ne considèrerons pas la confidentialité des algorithmes dans cet article, et nous supposons que tous les agents suivent le même algorithme. Cependant, ces travaux n'ont de sens qu'en l'existence d'agents « malveillants » collectant des données pour une utilisation inappropriée. Comme expliqué précédemment, nous traitons la confidentialité des domaines plutôt que des affectations. Notons que ces deux propriétés sont loin d'être indépendantes. Enfin, certaines catégorisations évoquent la confidentialité topologique, c'est-à-dire de la structure des problèmes. GDBA implique que deux agents sont voisins si et seulement si ils partageant une même contrainte. Une confidentialité complète de la topologie n'est pas réalisable, mais reste contrôlée. Dans les sections suivantes, nous traitons la confidentialité au niveau des *coûts* des contraintes, et des domaines.

## 2 Confidentialité des contraintes dans GDBA

Nous souhaitons modéliser et résoudre un problème où un agent A veut organiser un rendez-vous avec les agents B et C simultanément, si possible. Si un créneau commun ne peut être trouvé, A préfèrerait rencontrer B plutôt que C, mais il ne souhaite pas que cette préférence soit révélée, afin de ne pas contrarier C. Avec les variables $x_A$, $x_B$ et $x_C$ représentant l'heure du rendez-vous du point de vue de chaque agent, on peut modéliser ce problème par un ensemble de contraintes d'égalité ($x_A = x_B$ et $x_A = x_C$) pour exprimer le fait que les participants doivent se réunir à la même heure. On peut modéliser ce problème simple en utilisant des contraintes valuées : le coût est nul si la contrainte est satisfaite, et strictement positif si elle ne l'est pas. On peut traduire les préférences de A en "payant" un coût élevé si la première contrainte n'est pas satisfaite, et plus faible pour l'autre, mais à aucun moment C ne doit être mis au courant des coûts associés à la contrainte $x_A = x_B$ [4].

Comme expliqué dans la section précédente, GDBA nécessite à chaque étape que tous les agents envoient un message de type « improve » contenant le delta. Ce dernier est la différence entre le coût de l'affectation précédente et le meilleur coût qu'il puisse obtenir en changeant d'affectation. D'une part, cela peut donner des indices sur le coût des contraintes de l'agent, et d'autre part, si on considère un coût lié à la perte d'une information, on peut se demander s'il faut incorporer ce coût de confidentialité dans le calcul du delta. Et dans ce cas, s'il faut aussi contrôler la perte de confidentialité liée aux coûts de confidentialité eux-mêmes (et, pourquoi pas, récursivement).

Nous proposons une variante de GDBA qui résoud ces deux problèmes en supprimant la phase « improve » de l'algorithme. Il reste à trouver des mécanismes alternatifs pour éviter les oscillations, et sortir des minima locaux.

**Contribution 1 : prévention des oscillations en l'absence de message « improve ».** L'exemple suivant montre le problème lié aux oscillations : considérons par exemple deux agents $A_1$ et $A_2$ encapsulant les variables respectives $x_1$ et $x_2$, dont les domaines sont tous les deux $\{1, 2\}$, associées par la contrainte $x_1 = x_2$.

3. La notion de « contrainte non-satisfaite » dans le cadre des COP valués n'est pas triviale et l'essentiel de l'article de Okamoto, Zivan et Nahon [19] est consacré à cette problématique. Nous y reviendrons dans la section 2 de cet article.

4. Le modèle DCOP défini dans la section précédente suppose que la fonction de coût de chaque contrainte est connue des deux agents. Il serait souhaitable de réduire encore la publication de la fonction de coût, par exemple avec un modèle de type PKC [2, 11]. C'est une des perspectives de ce travail.

Initialement, $x_1 = 1$ et $x_2 = 2$, ce qui contredit la contrainte. Individuellement, pour chaque agent, la meilleure action est de changer d'affectation. À l'étape suivante, on aura $x_1 = 2$ et $x_2 = 1$, et la contrainte n'est toujours pas satisfaite. Si aucun mécanisme ne vient prévenir ce changement simultané, les deux agents vont continuer à osciller sans jamais parvenir à la solution.

L'algorithme DSA introduit un paramètre $p$ qui définit la probabilité avec laquelle un agent va changer d'affectation entre deux cycles [28]. Autrement dit, avec une probabilité $1-p$, chaque agent peut « sauter son tour », ce qui permet à l'agent voisin de changer seul d'affectation. Nous avons introduit ce paramètre $p$ dans notre variante de GDBA, que nous appelons dorénavant GDBA-$p$.

Il faut alors trouver une « bonne » valeur de $p$ pour chaque problème. Si $p$ est trop faible, les agents vont avoir tendance à garder leurs affectations trop longtemps, ce qui rend la recherche lente ; si $p$ est trop élevé, le phénomène d'oscillation réapparait. Cependant, la pratique montre qu'un $p$ relativement élevé, sans pour autant tendre vers 1, convient à une grande variété de problèmes. Dans nos conditions d'expérimentation, $p = 0{,}95$ est très satisfaisant pour l'ensemble des problèmes traités (cf section 4).

**Contribution 2 : *Breakout* en l'absence de message « improve ».** Nous avons réintroduit le mécanisme de *Breakout* pour échapper aux minima locaux comme suit : quand le delta d'un agent est positif ou nul, il conserve logiquement son affectation précédente, mais il envoie dans ce cas un message « stalled » à ses voisins. Si tous les voisins d'un agent sont dans ce cas, ce dernier détecte un quasi minimum local et peut réaliser le *breakout* en pondérant les contraintes non-satisfaites.

En résumé, un agent $A_i$ exécutant GDBA-$p$ ne réalise qu'une seule action à chaque étape :

— s'il existe une valeur $v' \in \mathcal{D}_i$ qui améliore la fonction de coût, $A_i$ peut *affecter* $x_i = v'$ avec une probabilité $p$, ou

— si $v'$ existe, avec une probabilité $1-p$, $A_i$ ne change pas sont affectation (il *passe*). Quoi qu'il en soit, il envoie un message « ok ? » à ses voisins pour faire part de la valeur finalement choisie.

— Si $v'$ n'existe pas, $A_i$ est en *plateau* et envoie un message « stalled » à ses voisins. Si tous ses voisins sont dans un tel plateau, un quasi minimum local est détecté et les contraintes non-satisfaites de $A_i$ sont pondérées. Les messages « stalled » permettent de distinguer les *« passes »* des *« plateaux »*.

**Algorithme 1 :** GDBA-$p$ exécuté par l'agent $A_i$

1 Initialiser les poids des contraintes à 1
2 $v \leftarrow \arg\min_{j \in D_i} \text{COST}(j, \emptyset, \emptyset)$
3 Transmettre "ok ?($v$)" à tous les voisins
4 **tant que** *condition d'arrêt non rencontrée* **faire**
5 $\quad$ $stallCount \leftarrow 0$
6 $\quad$ *Recevoir les messages de tous les voisins :*
7 $\quad\quad$ **when** *"ok ?"* : mettre à jour *agentView* et *minCosts*
8 $\quad\quad$ **when** *"stalled"* : incrémenter *stallCount*
9 $\quad$ $v' \leftarrow \arg\min_{j \in D_i} \text{COST}(j, agentView, \mathcal{R}_i)$
10 $\quad$ **si** $\text{COST}(v', agentView, \mathcal{R}_i) < \text{COST}(v, agentView, \mathcal{R}_i)$ **alors**
11 $\quad\quad$ Avec une probabilité $p$ : $v \leftarrow v'$
12 $\quad\quad$ $\mathcal{R}_i \leftarrow \mathcal{R}_i \cup \{v\}$
13 $\quad\quad$ Envoyer "ok ?($v$)" à tous les voisins
14 $\quad$ **sinon**
15 $\quad\quad$ **si** $stallCount = N_i$ **alors** // quasi minimum local
16 $\quad\quad\quad$ **pour chaque** $C_j \in \mathcal{C}(A_i) \mid C_j(agentView) > minCosts(C_j)$ **faire**
17 $\quad\quad\quad\quad$ Incrémenter le poids de $C_j$
18 $\quad\quad$ Envoyer "stalled" à tous les voisins
19 Affecter la meilleure $v$ à $x_i$

**Contribution 3 : non-minimalité dans un contexte de PPC ouverte.** Le travail de Okamoto, Zivan et Nahon [19] avait consisté à généraliser l'algorithme legacy DBA de Yokoo et Hirayama [27] aux contraintes valuées. Dans ce cadre, la notion de contrainte « non-satisfaite » se révèle insuffisante pour déterminer les contraintes à pondérer. Yokoo et Hirayama ont mesuré les performances de plusieurs variantes de « non-satisfaction ». Parmi celles-ci, la plus prometteuse est celle de non-minimalité (NM). Une contrainte est NM-insatisfaite si le coût de la solution actuelle est plus elevé que le coût minimal pouvant être obtenu par cette contrainte (i. e., pas forcément 0). Cependant, dans le cadre de PPC ouverte, si les domaines des voisins sont inconnus et la fonction de coût définie sur des ensembles infinis, il nous est impossible de connaitre le coût minimal. Nous avons traité ce problème en maintenant une structure de données *minCosts*, qui associe à chaque contrainte $C_i$ le coût le plus bas *connu* pour cette contrainte. Quand un voisin révèle une nouvelle valeur de son domaine, tous les coûts de chaque contrainte incluant cette nouvelle valeur sont évalués et *minCost* est mise à jour en conséquence. Pour maintenir cette structure, chaque affectation n'est évaluée qu'une fois pour chaque contrainte tout au long du processus de recherche, c'est-à-dire qu'il faut effectuer au plus $d^{|\mathcal{X}(C_i)|}$ tests de la contrainte. Comme nous nous sommes restreints au cas des contraintes binaires, nous n'avons pas rencontré d'explosion combinatoire. Généraliser cette technique aux contraintes non-binaires nécessitera probablement la mise en place de fonctions ad-hoc (comme pour les contraintes globales en PPC clas-

sique) ou d'approximations.

La valeur de $minCosts(C_i)$ ne peut que décroitre au cours de la recherche, mais cela signifie que $C_i$ peut être considérée comme minimalement satisfaite au début de la recherche, mais non-minimalement satisfaite plus tard par la même affectation. Cette propriété, bien que contre-intuitive, ne pose pas de difficulté à l'algorithme. La complexité spatiale de *minCosts* est $\Theta(|\mathcal{C}(A_i)|)$ pour chaque agent, soit globalement $\Theta(e)$. Il faut également tenir compte de l'ensemble des valeurs révélées par chaque voisin de chaque agent, ce qui nécessite pour chaque $A_i$ une structure de données de taille $\Theta(N_i \cdot d)$, soit globalement $\Theta(ed)$. GDBA-$p$ a les mêmes complexités temporelle et spatiale que Vanilla GDBA. L'algorithme tourne jusqu'à ce qu'une condition d'arrêt soit rencontrée, généralement une limite sur le nombre de cycles.

GDBA-$p$ est décrit extensivement ci-après (algorithme 1) : $v$ représente l'instantiation actuelle de la variable de l'agent. *agentView* contient une affectation représentant les instantiations connues des voisins. Une de nos contributions mineures consiste à affecter initialement $v$ non pas aléatoirement, mais à la meilleure valeur possible (ligne 2). Même si on ne peut pas à ce stade prendre en compte les contraintes inter-agent, on minimise les coûts des contraintes unaires et de confidentialité décrits dans la section suivante. Cette valeur est transmise aux voisins (ligne 3).

Les lignes 6 à 8 traitent les messages entrants. Les agents GDBA-$p$ envoient et reçoivent exactement un message par voisin à chaque étape (contre deux pour Vanilla GDBA). La ligne 9 sélectionne la meilleure valeur $v'$ pour la fonction COST. Cette fonction prend en compte l'*agentView*, et l'ensemble des valeurs précédemment révélées $\mathcal{R}_i$ (cf section suivante). La ligne 10 vérifie si $v'$ améliore la soluton courante. Si c'est le cas, $v'$ est affectée, avec une probabilité $p$ (ligne 11). C'est ici la principale différence entre GDBA-$p$ et Vanilla GDBA : dans ce dernier, il y a une étape supplémentaire où le delta entre les coûts associés à $v$ et $v'$ est envoyé à tous les voisins, et $v'$ n'est affectée que si l'agent courant permet la meilleure amélioration. Quoi qu'il en soit, la valeur choisie est ajoutée à $\mathcal{R}_i$ (ligne 12) et envoyée aux voisins (ligne 13).

Si $v'$ n'améliore pas la solution courante, la ligne 15 détecte la présence d'un quasi minimum local. Dans ce cas, les lignes 16 et 17 pondèrent les contraintes non-minimalement satisfaites.

## 3 Confidentialité des domaines dans GDBA

Pour illustrer notre travail, nous voulons modéliser le problème suivant : un agent A organise une réunion avec plusieurs collègues. Il est disponible toute la journée, de 8 h à 20 h, avec des préférences variables, mais ne souhaite par avouer qu'il a autant de disponiblités, afin de ne pas être submergé de demandes de réunions. Il est possible d'obtenir ce résultat en gardant les domaines privés, et en révélant aussi peu de valeurs que possible au cours de la recherche.

**Contribution 4 : contrôle des données révélées *via* des utilités.** Comme décrit dans la section 1, l'utilisation d'algorithmes stochastiques ainsi que de contraintes définies en intention sur des sur-ensembles des domaines permettent de ne pas dévoiler systématiquement tous les domaines lors de la phase d'initialisation. Cela n'interdit pas pour autant l'existence de contraintes à base de tables : on peut les définir sur des sur-ensembles des domaines, utiliser des valeurs par défaut si la table n'est pas exhaustive, ou encore des représentations compressées (e. g., [17]).

Quand un agent affecte une valeur à sa variable, il transmet le message « ok ? » à ses voisins. Il s'agit d'une perte de confidentialité. Le modèle de DCOP Utilitariste permet d'affecter un coût à cette fuite d'information [4, 21, 22]. Pour contrôler la confidentialité des domaines, nous considérons que la recherche est réalisée par des agents utilitaristes qui prennent leurs décisions pour minimiser leur propre fonction d'utilité. Celle-ci implique de minimiser d'une part le coût global, et d'autre part la perte de confidentialité. Ceci peut être formalisé comme suit :

**Définition 3.1.** Un DCOP utilitariste (UDCOP) est un quintuplet $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{U} \rangle$, i. e., un DCOP avec une matrice supplémentaire $\mathcal{U}$ de coûts de confidentialité sur les domaines. $\mathcal{U}_{i,j}$ est le coût, positif ou nul, pour $A_i$ de révéler si $j \in \mathcal{D}_i$.

L'objectif est de trouver une solution optimale tout en minimisant la perte de confidentialité $\sum_{i=1}^{n} \sum_{j \in \mathcal{R}_i} \mathcal{U}_{i,j}$, où $\mathcal{R}_i \subseteq \mathcal{D}_i$ est l'ensemble des valeurs que $A_i$ à révélées dans le processus de résolution.

L'objectif est maintenant double. Cependant, la perte de confidentialité dépend du processus de recherche lui-même et pas uniquement du problème et de sa solution. On ne peut donc pas utiliser les techniques traditionnelles de résolution multi-objectif. Les deux objectifs sont contradictoires : la perte de confidentialité ne peut que s'accroitre au cours de la recherche, au fur et à mesure que de nouvelles valeurs sont révélées, alors que les coûts sur les contraintes vont descendre alors que de nouvelles solutions sont découvertes. Nous voulons contrôler le processus de recherche pour parvenir à un compromis. Dans des travaux précédents et dans le cadre d'algorithmes complets, nous avons utilisé des éléments de théorie des jeux [20]. Ici, nous

**Algorithme 2 :** $\text{cost}(v, agentView, \mathcal{R}_i)$

1 $constraintCost \leftarrow \sum_{C_j \in \mathcal{C}(A_i)} weight(C_j) \times C_j(agentView \cup \{x_i = v\})$

2 $privacyCost \leftarrow \left( \max_{C_j \in \mathcal{C}(A_i)} weight(C_j) \right) \times \sum_{j \in R_i \cup \{v\}} \mathcal{U}_{i,j}$

3 **retourner** $constraintCost + privacyCost$

avons choisi de nous baser sur la *somme* entre les coûts des contraintes et la perte de confidentialité.

Cette technique peut être appliquée à DSA, Vanilla GDBA ou GDBA-$p$. Une des difficultés avec GDBA et ses variantes est que la pondération des contraintes peut, à terme, rendre les coûts de confidentialité négligeables. Nous avons observé expérimentalement que donner autant de poids aux coûts de confidentialité qu'à la contrainte la plus pondérée donnait de très bons résultats. L'algorithme 2 résume la fonction COST finale que nous avons utilisée. La fonction renvoie le coût (contraintes et confidentialité) qu'entrainerait l'affectation de la valeur $v$ à l'étape courante, étant donnés *agentView* et $\mathcal{R}_i$. Pour prendre en compte les contraintes unaires et des coûts de confidentialité lors de l'initialisation de l'algorithme, un coût par défaut (0) est renvoyé si un *agentView* incomplet ne permet pas de déterminer le coût d'une contrainte.

La valeur *privacyCost* calculée à la ligne 2 représente ce que nous appelons le *contrôle de la confidentialité des domaines* (DPC pour Domain Privacy Cost). Cette valeur peut être mémorisée entre deux appels à la fonction et mise à jour (en temps constant) quand le poids de la contrainte la plus pondérée est incrémenté, ou qu'une nouvelle valeur est révélée. Ainsi, la DPC n'a pas d'impact sur la complexité temporelle de GDBA.

## 4 Expérimentations

Nous avons implémenté DSA, GDBA et GDBA-$p$ avec et sans DPC sur la plateforme Akka 2.5 [1]. Akka est une implémentation moderne et performante du modèle Acteurs [14] pour la machine virtuelle Java. Elle convient parfaitement pour simuler des agents DCOP dans notre contexte expérimental.

Nous avons testé les algorithmes sur 150 instances des benchmarks suivants, et avons mesuré la moyenne des coûts *anytime* à chaque cycle. Les valeurs aléatoires non-négatives ont été générées, sauf exception, d'après une distribution log-normale de moyenne $\mu$ et écart-type $\sigma$ donnés, arrondie à l'entier le plus proche.

**MultiDMS** est un problème d'ordonnancement de réunions distribué : 40 réunions doivent être programmées sur $d = 30$ crénaux horaires. 100 individus sont impliqués dans ces réunions : pour chacune d'elles, chaque personne peut être sélectionnée pour participer avec une probabilité de 5 %. Chaque réunion a une durée ($\mu = 2{,}5; \sigma = 1{,}12$). De plus, les participants on besoin de temps ($\mu = 1{,}5; \sigma = 0{,}5$) pour passer d'une réunion à la suivante. Si le planning d'une personne nécessite d'être présent à deux réunions à la fois, ou que la contrainte de distance n'est pas respectée, une pénalité de $K = 100$ est imposée. Une contrainte unaire est générée pour chaque variable, donnant à chaque valeur un coût ($\mu = 20; \sigma = 6{,}06$). Un coût de confidentialité est généré pour chaque valeur ($\mu = 10; \sigma = 3$).

**RDO** est un problème binaire aléatoire de $n = 200$ variables avec un domaine de $d = 30$ valeurs. Un graphe de contraintes aléatoire d'une densité de 2 % est généré ($e = 398$). Chaque contrainte impose un coût ($\mu = 50; \sigma = 50$) pour chaque affectation possible de ses variables. Un coût de confidentialité est généré pour chaque valeur ($\mu = 10; \sigma = 10$).

**WGC** est une variante de coloration de graphe pondérée sur $n = 200$ variables avec $d = 30$ couleurs. Un graphe de contraintes aléatoire est généré avec une densité de 20 % ($e = 3\,980$). Ces contraintes binaires sont des contraintes d'inégalité "dures" ($K = 1\,000$). Une contrainte unaire de coût est générée pour chaque variable, imposant un coût à chaque valeur ($\mu = 50; \sigma = 50$). Un coût de confidentialité est généré pour chaque valeur ($\mu = 10; \sigma = 10$).

Nous avons choisi les paramètres de MultiDMS pour correspondre grossièrement à notre application cible, i. e., générer un planning hebdomadaire pour notre établissement. Le nombre de participants à une réunion suit une distribution binomiale de moyenne 5. Chaque paire (réunion, participant) nécessite une variable, soit 200 variables en moyenne pour chaque instance. Pour ce problème, notre implémentation peut exécuter environ 500 cycles de GDBA par seconde (temps *wall-clock*) sur une VM Serveur Java OpenJDK 11 64-Bit exécutée sur un processeur à 4 cœurs Intel i7-5600U @ 2,6 GHz avec 4 GiB de mémoire de tas allouée. Pour les deux autres classes de problèmes, plus académiques, nous avons choisi les paramètres pour obtenir des problèmes de taille similaire (variables et domaines) et un temps d'exécution similaire.

**Une première expérimentation compare DSA-0.95, GDBA et GDBA-$p$ pour plusieurs valeurs de $p$.** DPC est désactivé en supprimant la ligne 2 de l'algorithme 2, seul les coûts de contrainte sont observés. La figure 1 présente les résultats. Ces graphiques

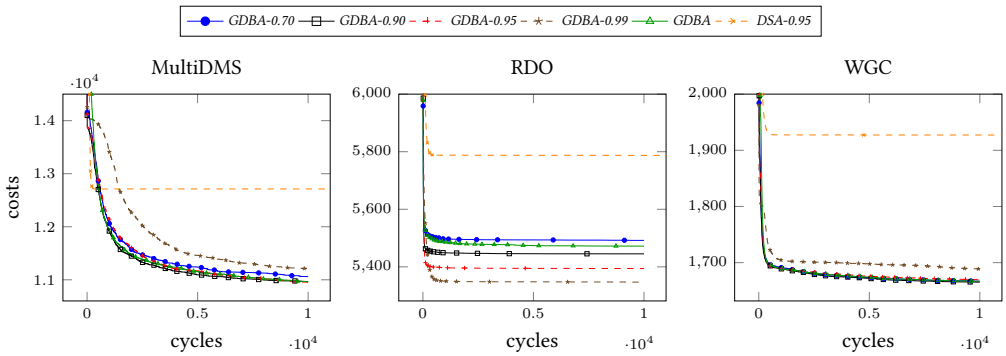



FIG. 1 – Coûts de contraintes *anytime* pour chaque cycle avec GDBA et GDBA-$p$

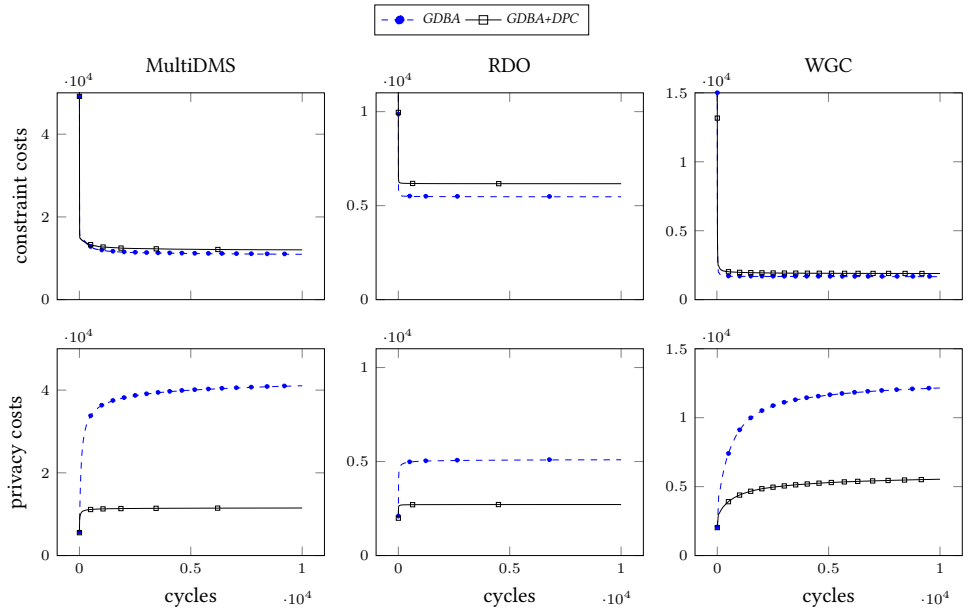



FIG. 2 – Impact de DPC sur les coûts de contrainte et de confidentialité *anytime* pour GDBA

montrent que GDBA-0.90 est toujours meilleur que Vanilla GDBA sur ces benchmarks. Cependant, GDBA-0.95 est nettement meilleur que GDBA-0.90 sur RDO, et la différence est presque impercetible sur les autres. GDBA-0.99 fonctionne particulièrement bien sur les problèmes RDO : les oscillations ne surviennent probablement pas sur un problème si homogène. Pour DSA, nous avons réalisé une expérimentation préliminaire pour déterminer le meilleur paramètre, qui s'est révélé être 0.95 également. DSA ne bénéficie d'aucun mécanisme pour échapper aux minima locaux et bloque rapidement sur une solution de mauvaise qualité. Notons que cela implique que peu de valeurs sont révélées.

**Une deuxième expérimentation compare les coûts des contraintes et de confidentialité pour Vanilla GDBA et GDBA-$p$, avec et sans DPC.** Nous avons choisi ici d'*inclure* les coûts de confidentialité dans le calcul des deltas pour les message "improve". Cela signifie que de l'information sur les coûts de confidentialité a pu être transmise, mais les prendre en compte réduit les coûts de confidentialité eux-mêmes. Les résultats sont présentés sur la figure 2. On peut observer que DPC permet de réduire les coûts de confidentialité d'un facteur 2 à 3 en moyenne, tout en ayant un impact relativement faible sur les coûts des contraintes. Des résultats similaires sont obtenus avec GDBA-0.95 (figure 3). On remarque également que GDBA-0.95 tend à révéler plus de valeurs que Vanilla GDBA. Pour mieux observer ce phénomène avec plusieurs valeurs de $p$, nous avons ajouté l'expérimentation suivante.

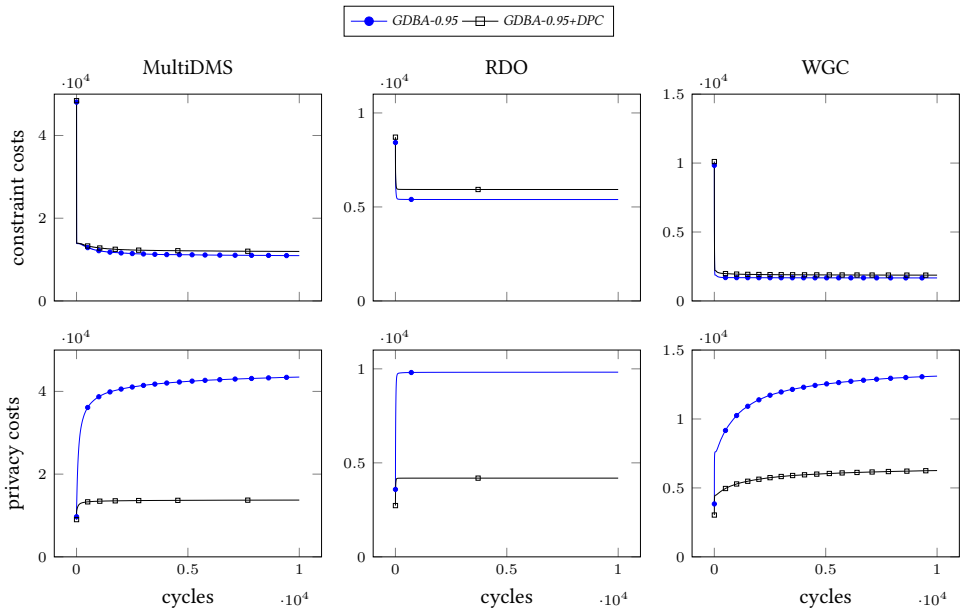



FIG. 3 – Impact de DPC sur les coûts de contrainte et de confidentialité *anytime* pour GDBA-0.95

**La troisème expérimentation compare les coûts de confidentialité pour Vanilla GDBA et GDBA-$p$ pour différents $p$, avec DPC désactivé ou activé.** La figure 4 montre que Vanilla GDBA ou GDBA-$p$ avec de petites valeurs de $p$ réduit le nombre de valeurs révélées, ce qui fait sens dans la mesure où un $p$ élevé implique des réaffectations plus aggressives. Ce comportement s'observe également quand DPC est activé, mais dans une moindre mesure. Pour WGC avec DPC désactivé et $p = 0{,}99$, i. e., une valeur trop élevée, l'impact des oscillations est tellement négatif sur la qualité de la résolution que peu de valeurs sont révélées.

Bien que, sous certaines conditions, Vanilla GDBA soit parfois plus efficace que GDBA-0.95, nous rappelons ici que contrairement à Vanilla GDBA, GDBA-$p$ ne transmet aucune information sur les coûts des contraintes, ce qui n'apparait pas sur les graphiques.

## 5 Conclusion et perspectives

Dans cet article, nous avons proposé une variante de l'algorithme de Breakout distribué généralisé (GDBA), que nous appelons GDBA-$p$. Cette variante supprime les messages « improve » de l'algorithme original. Ceci a pour double avantage de réduire le nombre de messages échangés d'un facteur 2, et de supprimer toute transmission d'information sur les coûts des contraintes. GDBA-$p$ met en place des stratégies alternatives pour limiter le phénomène d'oscillation et de pièges dans des minima locaux.

Une deuxième série de contributions concerne la mise en place d'un contrôle de la confidentialité des domaines (DPC) pour DSA, GDBA et GDBA-$p$, en exploitant le modèle de DCOP utilitariste proposé par DOSHI et al. [4] et SAVAUX et al. [21].

Nos résultats montrent que la DPC a un impact relativement faible sur la qualité des solutions, mais permet de réduire la fuite de données sur les domaines d'un facteur deux à trois.

Des perspectives ont été évoquées au cours de l'article : en premier lieu, il sera nécessaire d'expérimenter nos approches sur des problèmes présentant plusieurs variables par agent, des contraintes non-binaires, ou aux coûts exprimés dans $\mathbb{R}$. Les problèmes multi-variables sont particulièrement intéressants dans la mesure où il n'y a pas de perte de confidentialité lorsqu'un agent utilise des valeurs pour ses raisonnements locaux. D'autre part, aucune généralisation de DSA et GDBA aux problèmes multi-variables n'a encore été proposée.

Il serait également intéressant de comparer la qualité des solutions obtenues par les algorithmes stochastiques utilisés ici aux solutions optimales des problèmes. En travaillant sur des instances de plus petite taille, on pourra trouver des solutions optimales et observer la marge de progression à trouver pour améliorer ces algorithmes. La littérature sur les algorithmes incomplets (méta-heuristiques) est très riche dans le cas des problèmes centralisés, et il y a sans aucun doute beaucoup de choses à expérimenter sur des problèmes distribués.

Du point de vue de la confidentialité, on a évoqué au cours de l'article l'idée d'améliorer la confidentia-

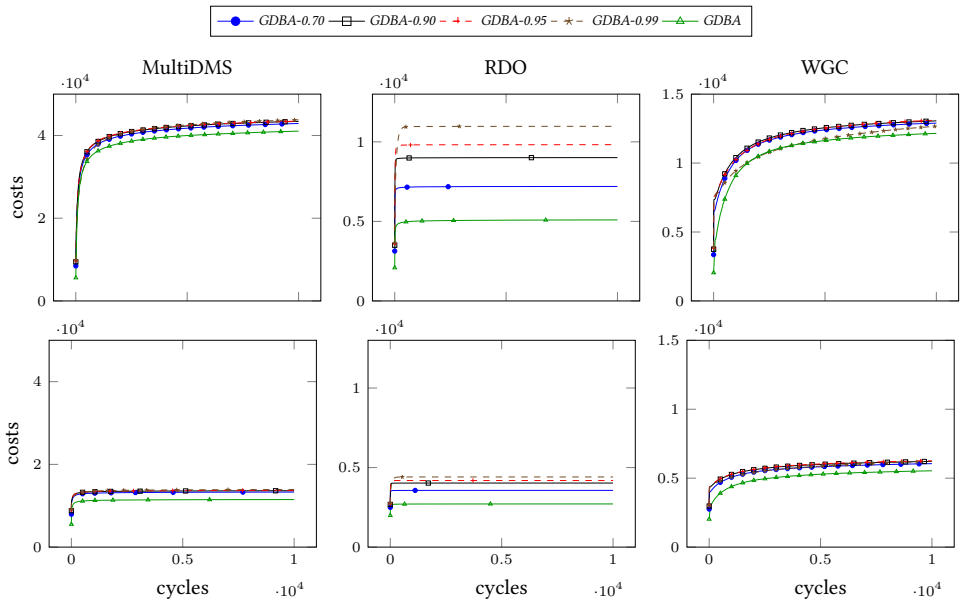



FIG. 4 – Coûts de confidentialité pour GDBA et GDBA-$p$ : DPC désactivé (au dessus) et activé (en dessous)

lité topologique par l'utilisation des contraintes asymétriques/PKC [2, 11]. De plus, si la fonction d'évaluation d'une contrainte n'est connue que d'un agent, il n'a pas besoin de transmettre son affectation à son voisin. Cela réintroduit une forme de confidentialité des affectations.

Nous avons évoqué le fait que le problème UDCOP est un problème à deux objectifs. Nous avons fait le choix d'utiliser la somme des coûts des contraintes et des coûts de confidentialité dans cet article, mais d'autres associations doivent être envisagés. Il n'est pas possible d'utiliser un front de Pareto puisque les coûts de confidentialité ne peuvent que croître au cours de la recherche, mais on peut envisager des produits, rapports, pondérations, des jeux, etc.

Enfin, une autre approche pour tester les performances des algorithmes pour la confidentialité serait de concevoir des agents malveillants qui ont pour stratégie de collecter le maximum de données de leurs voisins. Quelles garanties permettent DSA, GDBA, GDBA-$p$ et les utilités dans ce cadre ?

## Références

[1] J. BONÉR et THE AKKA TEAM AT LIGHTBEND. *Akka : Build powerful reactive, concurrent, and distributed applications more easily.* akka.io. 2009.

[2] I. BRITO, A. MEISELS, P. MESEGUER et R. ZIVAN. "Distributed constraint satisfaction with partially known constraints". In : *Constraints* 14.2 (2009), p. 199-234.

[3] L. CRÉPIN, Y. DEMAZEAU, O. BOISSIER et F. JACQUENET. "Privacy preservation in a decentralized calendar system". In : *Proc. 7th Itl Conf on Practical Applications of Agents and Multi-Agent Systems (PAAMS)*. T. 55. Advances in Intelligent and Soft Computing. 2009, p. 529-537.

[4] P. DOSHI, T. MATSUI, M. SILAGHI, M. YOKOO et M. ZANKER. "Distributed private constraint optimization". In : *Proc. IEEE/WIC/ACM Itl Conf on Intelligent Agent Technology*. IEEE Computer Society, 2008, p. 277-281.

[5] B. FALTINGS, T. LÉAUTÉ et A. PETCU. "Privacy guarantees through distributed constraint satisfaction". In : *Proc. IEEE/WIC/ACM Itl Conf on Intelligent Agent Technology*. T. 2. IEEE Computer Society. 2008, p. 350-358.

[6] B. FALTINGS et S. MACHO-GONZALEZ. "Open constraint programming". In : *Artificial Intelligence* 161.1 (2005). Distributed Constraint Satisfaction, p. 181-208. ISSN : 0004-3702.

[7] E. C. FREUDER, M. MINCA et R. J. WALLACE. "Privacy/efficiency tradeoffs in distributed meeting scheduling by constraint-based agents". In : *IJCAI Workshop on Distributed Constraint Reasoning*. 2001, p. 63-72.

[8] R. Greenstadt, B. Grosz et M. D Smith. "SSDPOP : improving the privacy of DCOP with secret sharing". In : *Proc. of the 6th Itl joint conference on Autonomous agents and multiagent systems.* IFAAMAS, 2007, p. 171.

[9] R. Greenstadt, J. P. Pearce et M. Tambe. "Analysis of privacy loss in distributed constraint optimization". In : *Proc 21st Nat. Conf. on Artificial Intelligence and the 18th Innovative Applications of Artificial Intelligence Conf.* AAAI Press, 2006, p. 647-653.

[10] T. Grinshpoun. "When You Say (DCOP) Privacy, What do You Mean ? - Categorization of DCOP Privacy and Insights on Internal Constraint Privacy". In : *Proc. of the 4th Itl Conference on Agents and Artificial Intelligence (ICAART).* Sous la dir. de Joaquim Filipe et Ana L. N. Fred. SciTePress, 2012, p. 380-386.

[11] T. Grinshpoun, A. Grubshtein, R. Zivan, A. Netzer et A. Meisels. "Asymmetric distributed constraint optimization problems". In : *Journal of Artificial Intelligence Research* 47 (2013), p. 613-647.

[12] T. Grinshpoun et T. Tassa. "P-SyncBB : A Privacy Preserving Branch and Bound DCOP Algorithm". In : *J. Artif. Intell. Res. (JAIR)* 57 (2016), p. 621-660.

[13] Y. Hamadi, C. Bessiere et J. Quinqueton. "Backtracking in distributed Constraint Networks". In : *Proc. of 13th European Conf. on Artificial Intelligence (ECAI).* Brighton, UK, 1998, p. 219-223.

[14] C. Hewitt, P. Bishop et R. Steiger. "A Universal Modular ACTOR Formalism for Artificial Intelligence". In : *Proc. 3rd IJCAI.* Stanford, USA, 1973, p. 235-245.

[15] K. Hirayama et M. Yokoo. "The distributed breakout algorithms". In : *Artificial Intelligence* 161.1-2 (2005), p. 89-115.

[16] T. Léauté et B. Faltings. "Protecting Privacy through Distributed Computation in Multiagent Decision Making". In : *Journal Artificial Intelligence Research (JAIR)* 47 (2013), p. 649-695.

[17] J.-B. Mairy, Y. Deville et C. Lecoutre. "The Smart Table Constraint". In : *Proc. 12th CPAIOR.* Sous la dir. de L. Michel. Springer International Publishing, 2015, p. 271-287. isbn : 978-3-319-18008-3.

[18] P. Morris. "The breakout method for escaping from local minima". In : *Proceedings of AAAI'93.* 1993, p. 40-45.

[19] S. Okamoto, R. Zivan et A. Nahon. "Distributed Breakout : Beyond Satisfaction". In : *Proc. 25th IJCAI.* 2016, p. 447-453.

[20] J. Savaux, J. Vion, S. Piechowiak, R. Mandiau, T. Matsui, K. Hirayama, M. Yokoo, S. Elmane et M. Silaghi. "Privacy Stochastic Games in Distributed Constraint Reasoning". In : *Annals of Mathematics and Artificial Intelligence* (to appear) (2019).

[21] J. Savaux, J. Vion, S. Piechowiak, R. Mandiau, T. Matsui, K. Hirayama, M. Yokoo, S. Elmane et M. Silaghi. "Utilitarian Approach to Privacy in Distributed Constraint Optimization Problems". In : *Proc. 30th FLAIRS.* 2017, p. 454-459.

[22] M. Silaghi et D. Mitra. "Distributed constraint satisfaction and optimization with privacy enforcement". In : *Intelligent Agent Technology (IAT).* IEEE. 2004, p. 531-535.

[23] T. Tassa, R. Zivan et T. Grinshpoun. "Preserving Privacy in Region Optimal DCOP Algorithms". In : *Proc. 25th IJCAI.* 2016, p. 496-502.

[24] T. Tassa, R. Zivan et T. Grinshpoun. "Privacy preserving implementation of the Max-Sum algorithm and its variances". In : *JAIR* 59 (2017), p. 311-349.

[25] R. J. Wallace et E. C. Freuder. "Anytime algorithms for constraint satisfaction and SAT problems". In : *ACM SIGART Bulletin* 7.2 (1996), p. 7-10.

[26] M. Yokoo, E. H. Durfee, T. Ishida et K. Kuwabara. "The distributed constraint satisfaction problem : Formalization and algorithms". In : *IEEE Transactions on knowledge and data engineering* 10.5 (1998), p. 673-685.

[27] M. Yokoo et K. Hirayama. "Distributed breakout algorithm for solving distributed constraint satisfaction problems". In : *Proc. of the 2nd Itl Conf on Multi-Agent Systems.* AAAI Press, 1996, p. 401-408.

[28] W. Zhang, G. Wang et L. Wittenburg. "Distributed stochastic search for constraint satisfaction and optimization : Parallelism, phase transitions and performance". In : *Proceedings of AAAI Workshop on Probabilistic Approaches in Search.* Alberta, Canada, juil. 2002.

[29] R. Zivan, S. Okamoto et H. Peled. "Explorative anytime local search for distributed constraint optimization". In : *Artificial Intelligence* 212 (juil. 2014), p. 1-26.

# Impact de la granularité spatio-temporelle des données sur l'optimisation des tournées de livraison en ville*

**Omar Rifki**[1,2] **Nicolas Chiabaut**[1] **Christine Solnon**[2]

[1] Université de Lyon, ENTPE / IFSTTAR, LICIT, UMR _T 9401, F-69518, Lyon, France
[2] Université de Lyon, INSA-Lyon, CNRS, LIRIS, UMR5205, F-69621, France
nicolas.chiabaut@entpe.fr {omar.rifki, christine.solnon}@insa-lyon.com

**Résumé**

L'optimisation des tournées de livraisons se ramène essentiellement à un problème de voyageur de commerce (TSP) dans un graphe où les sommets correspondent aux points à livrer, et les coûts des arcs aux temps de trajet entre deux points. Le TSP dépendant du temps (TD-TSP) est une extension du TSP dans laquelle le coût d'un arc dépend de l'heure de départ du nœud d'origine. Cette extension est particulièrement pertinente dans un contexte urbain, car les vitesses de déplacement varient en fonction de l'heure de la journée. Les fonctions de coût dépendantes du temps sont construites à partir de données issues de capteurs, et nous pouvons considérer pour cela différents niveaux de granularité temporelle (fréquence des mesures considérée) et spatiale (nombre et positionnement des capteurs dans le réseau routier). L'objectif de cet article est d'étudier l'impact de la granularité spatio-temporelle des fonctions de coût sur la qualité des tournées calculées ainsi que sur les temps de calcul. Pour cela, nous avons généré différents jeux de données en utilisant un logiciel de micro-simulation du trafic urbain, permettant d'obtenir des données très réalistes à différents niveaux de granularité spatio-temporelle. Nous considérons également différentes approches de résolution exactes (programmation linéaire en nombres entiers, *Branch&Bound* et programmation dynamique) et heuristiques (programmation dynamique restreinte et *limited discrepancy search*).

## 1 Introduction

Nous considérons un problème très général d'optimisation de tournées de livraison consistant à minimiser le temps nécessaire pour visiter un ensemble de sites de livraison ou de reprise de marchandises. Ce problème est généralement modélisé par un graphe orienté, où les sommets correspondent aux points à visiter, les arcs aux plus courts chemins entre sommets, et les coûts des arcs aux durées des plus courts chemins. La tournée optimale correspond alors au plus court circuit visitant chaque sommet du graphe exactement une fois, *i.e.*, la solution du problème du voyageur de commerce (*Traveling Salesman Problem* ou *TSP*).

Dans le TSP classique, les coûts des arcs sont supposés constants, *i.e.*, les temps de trajet sont les mêmes quelle que soit l'heure de la journée. Cette supposition n'est pas réaliste car les conditions de circulation varient du fait des embouteillages aux heures de pointe. En pratique, les plus courts chemins (*i.e.*, successions de tronçons routiers) entre les sites peuvent changer au cours de la journée, et leur durée également. Pour combler ce manque de réalisme, les fonctions définissant les coûts des arcs doivent être *dépendantes du temps (Time-Dependent ou TD)*, *i.e.*, le coût d'un arc doit dépendre de l'heure à laquelle il est traversé. Le TSP avec des fonctions de coût dépendantes du temps est appelé TD-TSP [9, 12, 13].

Dans cet article, nous supposons que les données de temps de trajet dépendantes du temps sont définies à l'aide de fonctions constantes par morceaux, avec des pas de temps de longueurs égales : si l'horizon temporel est de longueur totale $H$ et si la longueur du pas de temps est $s$, alors il y a $H/s$ pas de temps et, pour chacun de ces pas de temps et chaque tronçon de route possible, la durée de traversée du tronçon est supposée constante pendant ce pas de temps. Dans les faits, ce modèle est bien adapté aux données de trafic, car il

*Travail soutenu par le projet ELUD du Labex IMU. Les remerciements sont dus aussi à Matthis Manthe pour un travail précédent sur les algorithmes de résolution.

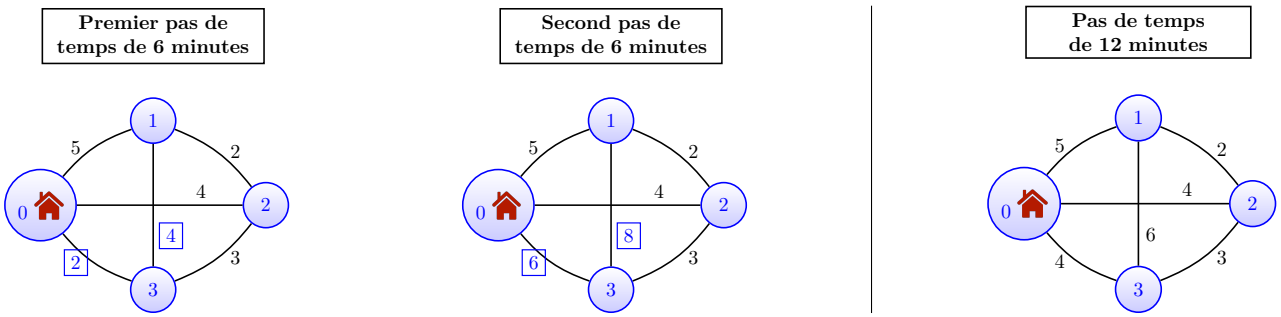



FIGURE 1 – Illustration de l'impact de la granularité des pas de temps sur les tournées calculées. Le réseau considéré a 4 sommets : le dépôt noté 0 et trois adresses de livraison $\{1, 2, 3\}$. Les arêtes représentent les plus courts chemins. À gauche : deux pas de temps de 6 minutes chacun, où les coûts des arcs $(0, 3)$ et $(1, 3)$ augmentent entre le premier et le deuxième pas ; la meilleur tournée est $\langle 0, 3, 1, 2, 0\rangle$ (afin de traverser les arcs $(0, 3)$ et $(3, 1)$ avant qu'ils ne soient congestionnés). À droite : un seul pas de temps de 12 minutes tel que les coûts des arcs $(0, 3)$ et $(1, 3)$ sont des moyennes des deux pas de gauche ; la meilleure tournée devient $\langle 0, 1, 2, 3, 0\rangle$.

correspond au schéma habituel d'estimation des durées de trajet. La figure 1 montre, à travers un exemple simple, comment la modification de la longueur $s$ des pas de temps de 6 à 12 minutes peut impacter la qualité d'une tournée. Elle peut également avoir un impact sur l'efficacité de résolution des algorithmes utilisés pour calculer la solution optimale. Par ailleurs, les fonctions de coût sont construites à partir de données issues de capteurs, et les caractéristiques de ces fonctions diffèrent selon le nombre de capteurs et leur positionnement dans le réseau routier.

Notre objectif est d'étudier l'impact de la granularité spatio-temporelle des données sur la qualité des solutions calculées d'une part et les temps de calcul d'autre part. Dans la section 2, nous décrivons le TD-TSP et les approches de résolution considérées : exactes (programmation linéaire en nombres entiers, *Branch&Bound* et programmation dynamique), et heuristiques (programmation dynamique restreinte et *limited discrepancy search*). Dans la section 3, nous décrivons les jeux de données que nous avons générés. Nous avons utilisé pour cela un logiciel de micro-simulation du trafic urbain, permettant de générer des jeux réalistes où nous pouvons faire varier d'une part la taille des pas de temps (pour la dimension temporelle) et d'autre part le nombre de capteurs utilisés pour évaluer les vitesses (pour la dimension spatiale). Dans la section 4, nous évaluons l'impact de la granularité spatio-temporelle des données sur la qualité des tournées calculées et sur les temps de calcul des algorithmes.

## 2 Le problème du TD-TSP

### 2.1 Description du problème

Nous considérons des graphes orientés et complets dont les sommets correspondent aux adresses de livraison (nous montrons dans la section 3 comment ces graphes sont construits à partir d'un réseau de transport routier). On note $n$ le nombre de sommets, $V$ l'ensemble des sommets et $0 \in V$ le dépôt à partir duquel le tour commence.

Introduit en 1992 par Malandraki et Daskin [22], le TD-TSP est une généralisation du TSP qui vise à trouver le circuit hamiltonien de durée minimale dans le cas où le coût de chaque arc $(i, j)$ varie dans le temps et dépend de l'heure de départ $t$ du sommet $i$. Ce coût sera noté $c^t_{ij}$. Étant donné un chemin $P = \langle v_1, \ldots, v_k\rangle$, une heure de départ $t_0$ et un sommet $v_i \in P$, l'heure d'arrivée sur $v_i$ est notée $at(v_i, t_0, P)$ et est définie récursivement par :

$$\begin{aligned} at(v_1, t_0, P) &= t_0 \\ \forall i \in [2, k],\ at(v_i, t_0, P) &= at(v_{i-1}, t_0, P) + d(v_{i-1}) \\ &\quad + c^{at(v_{i-1},t_0,P)+d(v_{i-1})}_{v_{i-1},v_i}, \end{aligned}$$

où $d(v_i)$ est la durée d'arrêt associée au sommet $v_i$. La durée de parcours associée au chemin $P$ et à une heure de départ $t_0$ est notée $ts(t_0, P)$ et est définie par :

$$ts(t_0, P) = at(v_k, t_0, P) - \sum_{i=1}^{k} d(v_i) - t_0.$$

L'objectif du TD-TSP est de trouver un circuit $P = \langle v_0, \ldots, v_n\rangle$ tel que $P$ commence et termine sur le sommet 0 (*i.e.* , $v_0 = v_n = 0$) ; $\{v_0, \ldots, v_{n-1}\}$ est une permutation de $V$ ; et la durée de parcours de $P$ en partant de $v_0$ à $t_0$ (*i.e.*, $ts(t_0, P)$) est minimale.

### 2.2 Approches de résolution

Différentes approches ont été proposées pour résoudre le TD-TSP, depuis son introduction par [22]. Certaines sont basées sur des méta-heuristiques telles

que la recherche taboue [31], le recuit simulé [30] ou les algorithmes de colonies de fourmis [10]. Dans cette section, nous nous focalisons sur les approches exactes, *i.e.*, la programmation par contraintes (*Constraint Programming - CP*), la programmation linéaire en nombres entiers (*Integer Linear Programming - ILP*), *Branch&Bound* (*B&B*) et la programmation dynamique (*Dynamic Programming - DP*), ainsi que sur les approches heuristiques dérivées de ces méthodes exactes, *i.e.*, *Limited Discrepancy Search* (*LDS*), et la programmation dynamique restreinte (*Restricted DP - RDP*).

**CP.** Melgarejo *et al.* ont introduit dans [24] la contrainte globale *TDNoOverlap*, qui garantit qu'un ensemble de tâches ne se chevauchent pas lorsque les temps de transition entre les tâches dépendent du temps. Cette contrainte peut être utilisée pour résoudre le TD-TSP. Elle est bien plus efficace que les modèles CP classiques pour le TD-TSP (basés sur les contraintes *allDifferent*) puisqu'elle propage les relations de précédence entre les tâches. Cependant, *TDNoOverlap* ne passe pas bien à l'échelle en l'absence de fenêtres temporelles (*i.e.*, contraintes relatives aux heures d'arrivée et de départ). Par exemple, elle n'est pas toujours en mesure de résoudre des instances avec $n = 20$ sommets dans un délai de 900 secondes. Par conséquent, nous ne considérons pas CP dans notre étude expérimentale.

**ILP.** La première formulation linéaire en nombres entiers du problème est due à Malandraki et Daskin [22]. Par la suite, plusieurs approches ont été conçues. Cordeau *et al.* [8] ont notamment proposé une méthode permettant de calculer une borne inférieure du problème et de résoudre ainsi le TD-TSP à l'aide d'un algorithme *Branch & Cut* (B&C). Cependant certaines instances ayant 40 sommets ne sont pas résolues en un temps raisonnable, même lorsque les fonctions de coût ne comportent que trois pas de temps. Arigliano *et al.* [1] ont étendu le modèle de [8] aux fenêtres temporelles (TD-TSPTW), et Montero *et al.* [25] ont mis au point un algorithme B&C pour résoudre un problème similaire. Récemment, Vu *et al.* [32] et Boland *et al.* [4] ont proposé une méthode de résolution basée sur des graphes étendus dans le temps. A notre connaissance, cette méthode représente actuellement l'état de l'art pour le TD-TSPTW : elle est capable de résoudre en quelques secondes des instances ayant jusqu'à $n = 40$ sommets, même lorsque le nombre de pas de temps est égal à 73, alors que l'approche de [25] ne passe pas à l'échelle dans ce cas. Cependant, l'approche de [32] ne passe pas à l'échelle lorsqu'on considère des instances sans fenêtres temporelles. Nous l'avons testé [1] sur nos instances (qui n'ont pas de fenêtres temporelles) et avons constaté qu'elles ont besoin de plusieurs heures pour converger, même dans le cas de petites instances.

1. Nous remercions Vu *et al.* d'avoir partagé leur code source.

**Algorithme 1 :** B&B($P$)

**Input :** Un chemin $P = \langle v_0, \ldots, v_i \rangle$ allant du dépôt $v_0 = 0$ à un sommet $v_i$

1 Soit $t_0$ l'heure de départ du dépôt 0
2 Soit $t*$ l'heure d'arrivée de la meilleure tournée trouvée
3 Soit $C = V \setminus \{v_0, \ldots, v_i\}$ l'ensemble des sommets qui ne sont pas encore visités par $P$
4 **si** $C = \emptyset$ **alors**
5  **si** $at(v_i, t_0, P) + d(v_i) + c_{v_i,v_0}^{at(v_i,t_0,P)+d(v_i)} < t*$ **alors**
6  mettre à jour $t*$
7 **sinon**
8  **pour** *chaque sommet* $v_j \in C$ **faire**
9  $h_{v_j} \leftarrow at(v_j, t_0, P.\langle v_j \rangle) + d(v_j) + bound(v_j, C)$
10  **pour** *chaque sommet* $v_j \in C$ *considéré par valeur croissante de* $h_{v_j}$ **faire**
11  **si** $h_{v_j} < t*$ **alors** B&B($P.\langle v_j \rangle$);

Par conséquent, étant donné que les approches ILP existantes ne sont pas capables de résoudre en un temps raisonnable des instances ayant plus de 3 pas de temps et pas de fenêtres temporelles, nous ne les considérons pas dans notre étude expérimentale (sauf pour le cas statique avec un seul pas de temps).

**B&B.** Nous considérons une approche B&B classique décrite dans l'algorithme 1. Le paramètre en entrée $P$ est un chemin du dépôt au sommet actuel $v_i$ (initialement, $P = \langle 0 \rangle$). Si $P$ contient tous les sommets et si l'heure d'arrivée au dépôt est inférieure à la meilleure solution trouvée jusqu'à présent, la meilleure solution est mise à jour (lignes 4-6). Sinon, pour chaque sommet non visité $v_j \in C$, B&B calcule une borne inférieure $h_{v_j}$ de l'heure d'arrivée de la meilleure tournée qui visite successivement $v_0, \ldots, v_i$, puis $v_j$, puis tous les sommets restants de $C \setminus \{v_j\}$, et retourne enfin au dépôt (lignes 8-9). Le calcul de cette borne inférieure est détaillé ci-dessous. Enfin, l'algorithme itère sur chaque sommet non visité $v_j \in C$ par ordre croissant de la borne inférieure $h_{v_j}$. Si $h_{v_j}$ est inférieure à la meilleure solution trouvée jusqu'à présent, l'algorithme appelle récursivement B&B avec $v_j$ ajouté à la fin du chemin $P$ (lignes 10-11).

La fonction $bound(v_j, C)$ calcule une borne inférieure de la durée du plus court chemin qui part de $v_j$ à l'heure $t_j = at(v_j, t_0, P.\langle v_j \rangle) + d(v_j)$, puis visite exactement une fois tous les sommets de $C \setminus \{v_j\}$, et termine enfin au dépôt. Ce type de borne est généralement calculé en résolvant des relaxations du problème initial. Pour le TSP asymétrique (ATSP), le problème d'affectation (*Assignment Problem - AP*) est un problème classique

qui relâche la contrainte de connectivité du chemin : l'objectif de AP est de sélectionner un ensemble d'arcs dans $\{(v_k, v_l) : v_k \in C, v_l \in \{0\} \cup C \setminus \{v_j\}\}$ tel que (i) $v_j$ possède exactement un arc sortant ; (ii) 0 a exactement un arc entrant ; (iii) chaque sommet de $C \setminus \{v_j\}$ a exactement un arc entrant et un arc sortant ; et (iv) la somme des coûts des arcs sélectionnés est minimale. Cette relaxation peut être résolue en $\mathcal{O}(n^3)$ en utilisant des variantes améliorées de l'algorithme Hongrois [6].

Cependant, pour le TD-TSP, le coût d'un arc $(v_k, v_l)$ dépend de l'heure de départ de $v_k$ qui n'est pas connue (sauf pour $v_j$). Pour assurer que le coût du AP est une borne inférieure du chemin optimal, nous devons considérer le plus bas coût possible pour chaque arc $(v_k, v_l)$, *i.e.*, $c^{min}_{v_k v_l} = \min_{l_k \leq t \leq u_k} c^t_{v_k v_l}$ où $l_k$ (resp. $u_k$) est l'heure de départ possible la plus petite (resp. la plus grande) à partir de $v_k$. Bien sûr, plus $u_k - l_k$ est restreinte, plus $c^{min}_{v_k v_l}$ est élevé, et meilleure est la borne calculée par AP. Dans cet article, nous calculons $l_k$ et $u_k$ comme suit : $l_k$ correspond au temps d'arrivée à $v_k$ si $v_k$ est visité juste après $v_j$ (*i.e.*, $l_k = at(v_j, t_0, P.\langle v_j \rangle) + d(v_j) + c^{at(v_j, t_0, P.\langle v_j \rangle) + d(v_j)}_{v_j v_k}$) ; et $u_k$ est l'heure de départ au plus tard de $v_k$ qui permet de retourner au dépôt à $t*$ (*i.e.*, $u_k = t* - c^{t* - c^{min}_{v_k v_0}}_{v_k v_0}$), où $t*$ est l'heure d'arrivée de la meilleure tournée trouvée. Notons que nous pourrions encore réduire l'intervalle $[l_k, u_k]$ en inférant des relations de précédence entre les sommets, comme proposé dans [24]. Cependant, cela n'a pas (encore) été implémenté.

Comme nous devons résoudre AP pour chaque sommet candidat $v_j \in C$, et que les différentes instances de AP à résoudre ne diffèrent que d'un seul sommet, nous utilisons la version incrémentale de l'algorithme Hongrois qui résout plus efficacement AP en réparant une affectation existante [11].

**LDS.** La recherche à divergence limitée LDS [15] n'explore que partiellement l'arbre de recherche construit par B&B. Elle suppose qu'il existe une heuristique pouvant être utilisée à chaque sommet de l'arbre afin de classer toutes les décisions possibles, de la meilleure à la plus mauvaise : pour chaque nœud la divergence d'une décision est égale à son rang (où la meilleure décision a le rang 0) ; et la divergence d'une branche de l'arbre de recherche (de la racine à un nœud donné) est égale à la somme des divergences des décisions prises à chacun de ses nœuds. L'idée de LDS est d'explorer les branches en augmentant la divergence : LDS explore d'abord la branche sans divergence (la meilleure décision est choisie à chaque nœud) ; puis toutes les branches dont la divergence est égale à 1 ; puis celles dont la divergence est égal à 2, etc.

LDS est connue pour explorer certains nœuds plusieurs fois. Pour surmonter ce problème, nous considérons une variante de LDS introduite par Beck et Perron dans [2], appelée *Discrepancy-Bounded Depth First Search* (*DBDFS*) : l'idée est d'effectuer un parcours en profondeur sur tous les nœuds avec des chemins qui ont jusqu'à un certain nombre borné de divergences.

Pour le TD-TSP, les décisions possibles à chaque nœud de l'arbre de recherche sont les sélections d'un sommet non visité $v_j \in C$, et l'heuristique utilisée pour ordonner les décisions est $h_{v_j}$ (calculée à la ligne 9 de l'algorithme 1). Cette heuristique renvoie une borne inférieure de la tournée optimale commençant par $P.\langle v_j \rangle$. Au lieu de définir la divergence de $v_j$ simplement par son rang, nous la définissons comme suit : $h_{v_j} - \min_{v_k \in C} h_{v_k}$ (comme celle proposé dans [5], par exemple). En d'autres termes, plus $h_{v_j}$ est proche de la meilleure valeur heuristique, plus la divergence est petite. Par conséquent, notre algorithme LDS est obtenu à partir de l'algorithme 1 en modifiant la condition pour l'appel récursif de $B\&B$ (ligne 11) comme suit :

$$h_{v_j} < t* \text{ et } d_{tot} + h_{v_j} - \min_{v_k \in C} h_{v_k} < d_{max}$$

où $d_{tot}$ est la divergence totale du chemin $P$ (elle est incrémentalement calculée en ajoutant $h_{v_j} - \min_{v_k \in C} h_{v_k}$ à chaque appel récursif), et $d_{max}$ est un paramètre de l'algorithme qui définit la divergence maximal.

**DP.** La méthode de programmation dynamique proposée par Held et Karp pour résoudre le TSP [16] peut être facilement étendue au TD-TSP. Plus précisément, pour chaque sommet $v_i \in V$ et chaque sous-ensemble de sommets $S \subseteq V \setminus \{0, v_i\}$, soit $p(v_i, S)$ le temps d'arrivée au plus tôt d'un chemin qui commence au sommet 0 à $t_0$, visite chaque sommet de $S$ exactement une fois, et se termine au sommet $v_i$. Les équations de Bellman qui définissent récursivement $p(v_i, S)$ sont :

$$\begin{aligned} \text{si } S = \emptyset,\ p(v_i, S) &= c^{t_0}_{0 v_i} \\ \text{sinon, } p(v_i, S) &= \min_{v_j \in S} p(v_j, S \setminus \{v_j\}) + d(v_j) + \\ &\quad c^{p(v_j, S \setminus \{v_j\}) + d(v_j)}_{v_j v_i} \end{aligned}$$

La solution du TD-TSP est donnée par $p(0, V \setminus \{0\})$, et peut être calculée itérativement en considérant des sous-ensembles $S$ de cardinalités croissantes : nous calculons d'abord $p(v_i, \emptyset)$ pour chaque sommet $v_i$ ; puis, à chaque niveau $k$ (avec $k$ allant de 1 à $n-1$), nous calculons $p(v_i, S)$ pour chaque sommet $v_i$ et chaque sous-ensemble $S$ de cardinalité $k$. Nous utilisons un tableau de $n$ bits pour représenter chaque sous-ensemble $S$, de sorte que le $j$ème bit soit égal à 1 ssi $j \in S$. La complexité temporelle de cet algorithme est $\mathcal{O}(n^2 \cdot 2^n)$ et sa complexité spatiale est $\mathcal{O}(n \cdot 2^n)$.

**RDP.** Pour éviter une explosion de DP, à la fois en termes de temps et de mémoire, RDP limite le nombre de résultats mémorisés à chaque niveau $k$. Cette idée est utilisée pour résoudre le TD-TSP dans [23]. Elle est également utilisée dans [3] pour calculer des solutions approchées de divers problèmes d'optimisation avec des diagrammes de décision restreints.

RDP a un seul paramètre $H$ qui correspond au nombre maximal de résultats mémorisés à chaque niveau $k$ : pour chaque résultat $p(v_i, S)$ mémorisé au niveau précédent $k-1$ et chaque sommet $v_j \in V \setminus (S \cup \{v_i\})$, RDP calcule $p(v_j, S \cup \{v_i\})$ et, si le nombre total de résultats calculés dépasse $H$, il mémorise les $H$ meilleurs résultats. Comme pour DP, nous utilisons des vecteurs de bits pour représenter les sous-ensembles. Nous utilisons également une file de priorité pour mémoriser les $H$ meilleurs résultats et une table de hachage pour un accès en temps constant à l'index d'un résultat dans la file de priorité. La complexité spatiale de cet algorithme est $\mathcal{O}(H)$ et sa complexité temporelle est $\mathcal{O}(n^2 H \cdot \log H)$ (il y a $n$ itérations et, à chaque itération, il y a au plus $H$ résultats et pour chacun de ces résultats, il y a au plus $n$ successeurs possibles ; le facteur $\log H$ provient du fait que nous utilisons une file de priorité pour stocker les résultats).

## 3 Description des données

### 3.1 Limitations des données existantes

Pour comprendre et évaluer les effets des conditions de trafic et de la granularité des coûts dépendants du temps sur les tournées optimales, il est primordial de disposer de données réalistes. En effet, les algorithmes du TD-TSP sont généralement évalués sur des données artificielles, et dans la plupart des cas, ils ne prennent en compte que peu de pas de temps.

Par exemple, les données introduites dans [8], qui sont largement utilisées pour évaluer les algorithmes du TD-TSP, ne considèrent que trois pas de temps : le premier et le troisième pas correspondent respectivement aux heures de pointe du matin et du soir, tandis que le deuxième pas correspond au milieu de la journée, lorsque la densité du trafic est plus faible. De plus, ces données sont générées d'une manière aléatoire et, même si des hypothèses réalistes ont été émises pour générer des temps de parcours (en considérant trois zones concentriques, avec un réglage différent pour chaque zone et des scénarios différents correspondant à des modèles de trafic différents), il est probable que les conclusions tirées de ces données artificielles ne seraient plus valables sur des données réelles. En particulier, une question importante abordée dans notre étude expérimentale est la suivante : pouvons-nous calculer de meilleurs tours si nous considérons des données de trafic en entrée définies à un niveau de granularité plus fin ? Pour répondre à cette question, nous avons besoin de données réalistes, à différents niveaux de granularité temporelle et spatiale avec des variations réalistes pour les temps de parcours.

Un benchmark plus réaliste est introduit dans [24]. Il a été généré à l'aide de données de trafic réelles provenant de capteurs (boucles magnétiques) qui mesurent les conditions de circulation sur 630 tronçons routiers de la ville de Lyon. Cependant, ce benchmark présente trois principales faiblesses.

Premièrement, la majorité des tronçons ne sont pas équipés de capteurs et, dans ce cas, la vitesse a été interpolée en fonction des capteurs les plus proches. De plus, les capteurs considérés (boucles magnétiques) ont des faiblesses bien connues en théorie du trafic, en particulier pour estimer les temps de trajet [21]. En effet, ils enregistrent des conditions de trafic de manière ponctuelle et isolée. Par conséquent, si une file d'attente se produit dans une rue mais n'atteint pas le capteur, elle ne sera pas enregistrée, ce qui entraînera une surestimation de la vitesse. Au contraire, si une file d'attente dépasse les capteurs, elle ne mesurera que la congestion, ce qui n'est pas totalement représentatif des conditions de circulation dans la rue et conduit à une sous-estimation de la vitesse.

Deuxièmement, la fonction de coût dépendante du temps entre deux adresses de livraison a été obtenue en considérant que la durée d'un chemin est égale à la somme des temps de parcours de ses tronçons de route. Cela sous-estime le temps de parcours réel car le temps passé sur les sommets (correspondant aux carrefours entre tronçons) n'est pas pris en compte, alors que le temps nécessaire pour traverser un carrefour ou pour virer à gauche est un facteur important d'augmentation des temps de parcours aux heures de pointe.

Enfin, le benchmark de [24] a été généré à l'aide de données mesurées entre 6h00 et 12h30. Si cela est suffisant pour les petites tournées, avec un maximum de 30 points de livraison, des tournées avec 50 ou 100 adresses de livraisons ont généralement des heures d'arrivée dépassant 12h30, même en supposant que la tournée commence à 6h00.

Un objectif de cette étude est de fournir à la communauté de chercheurs un jeu de données pour évaluer l'impact de la granularité spatio-temporelle des données sur la qualité des solutions trouvées d'une part et sur les performances des algorithmes de résolution du TD-TSP d'autre part.

### 3.2 Estimation de la durée d'un tronçon

Pour obtenir un accès complet à des données réalistes, nous utilisons un simulateur dynamique et mi-

FIGURE 2 – Le réseau routier lyonnais considéré dans cette étude et les positions réelles des capteurs placés par la collectivité de la Métropole de Lyon (en jaune).

croscopique du trafic, appelé SYMUVIA [7], sur une partie du réseau de transport de Lyon, visible en figure 2. SYMUVIA simule toute la complexité du trafic en tenant compte des différentes classes de véhicules, du comportement de conduite individuelle, des phénomènes de changement de voie, etc. Il utilise une loi de suivi de voiture basée sur le modèle de transport de Newell [26] et ses extensions [19, 20]. Même si la simulation n'est qu'une approximation du monde réel, elle permet d'avoir un accès aux détails les plus fins et d'émuler toutes les mesures possibles de la dynamique du trafic : temps de parcours individuels, vitesses des liaisons, données mesurées par les capteurs (boucles magnétiques), etc. Afin d'obtenir une simulation réaliste, les données utilisées par la simulation (nombre de véhicules, etc) ont été calculées à partir des données mesurées par les capteurs réels (dont les positions sont données dans la figure 2). La simulation permet d'obtenir, pour chaque tronçon de route $r$ du réseau et chaque instant $t$ de la journée, deux valeurs correspondant à ce qui aurait été mesuré à l'instant $t$ par un capteur réel qui serait positionné le long du tronçon $r$, à savoir la concentration K (véh./m) et le débit Q (véh./s), avec des conditions de trafic identiques à la simulation. Les deux valeurs sont le taux d'occupation et le débit observé sur le tronçon, et elles permettent d'en déduire la durée de traversée du tronçon à l'instant $t$.

### 3.3 Sources d'imprécision des données

Bien évidemment, au moment où un transporteur planifie une tournée de livraison, il ne connaît pas les conditions exactes de circulation qui seront observées pendant la réalisation de la tournée, et il doit utiliser un modèle prédictif pour estimer ces conditions de circulation. La conception de ces modèles prédictifs en milieu urbain est un sujet de recherche très actif que nous n'abordons pas ici (voir, par exemple, [29] pour une comparaison de différents modèles prédictifs utilisant les données issues des capteurs visualisés dans la figure 2). Nous supposons dans cette étude que nous disposons d'un modèle prédictif parfait pour chaque capteur, et nous considérons deux cas : le cas (optimiste) où chaque tronçon du réseau est équipé d'un capteur, et le cas (réaliste) où les capteurs sont positionnés exactement aux mêmes endroits que dans le réseau lyonnais actuel (cf figure 2), ce qui correspond à une couverture de 7,35% des tronçons. Dans ce cas, les données des tronçons non couverts par un capteur sont générées par interpolation en considérant le capteur le plus proche (en terme de distance Euclidienne).

Nous obtenons les deux jeux de données suivants : $D^+$ est le jeu obtenu avec une couverture totale des tronçons par les capteurs, et $D^-$ est le jeu obtenu avec une couverture partielle de $7,35\%$ des tronçons. Chacun de ces jeux donne, pour chaque tronçon de route et chaque période de 30 secondes, la durée de traversée du tronçon à cette période.

Enfin, pour chaque jeu $D^* \in \{D^+, D^-\}$, nous avons généré 5 fonctions de coût notées $D^*Sl$ où $l \in \{6, 12, 24, 60, 720\}$ est la longueur du pas de temps (en minutes), définissant la fréquence à laquelle les durées de traversée sont calculées : la fonction $D^*Sl(r)$ définissant la durée de traversée d'un tronçon $r$ est une fonction constante par morceaux, chaque morceau étant un intervalle de $l$ minutes, et la durée de traversée de $r$ à l'intervalle $[t, t+l[$ est obtenue à travers les moyennes des concentrations K et des débits Q des $2*l$ périodes de 30 secondes de $[t, t+l[$ dans le jeu $D^*$. Le cas où $l = 720$ correspond à des coûts constants, puisque notre horizon temporel [7h00,19h00] est de 12 heures = 720 minutes.

### 3.4 Calcul de la durée d'un plus court chemin

La durée de parcours d'un chemin est obtenue en faisant la somme des durées des tronçons qui le composent. Afin de mieux intégrer les durées de traversée des carrefours, les tronçons ont été définis de telle sorte que les carrefours ne sont pas positionnés aux extrémités des tronçons, mais au milieu des tronçons.

Une propriété importante pour pouvoir calculer efficacement un plus court chemin avec des données dépendantes du temps est la propriété FIFO : la fonction de coût d'un tronçon $(i, j)$ est FIFO si pour tout couple d'heures $(t_1, t_2)$ tel que $t1 < t2$, on a $t_1 + c_{i,j}^{t_1} < t_2 + c_{i,j}^{t_2}$ (autrement dit, on ne peut pas arriver plus tôt sur $j$ en partant plus tard de $i$). Si les données sont FIFO, alors les plus courts chemins peuvent être calculés en

| | | | B&B | | LDS | | | | | | | | DP | RDP | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $n$ | $\frac{D^-}{D^+}$ | | | $d = 0.1$ | | $d = 0.2$ | | $d = 0.5$ | | $d = 1.0$ | | | $H = 10^3$ | | $H = 10^4$ | | $H = 10^5$ | |
| | | | $t1$ | $t2$ | $t1$ | % | $t1$ | % | $t1$ | % | $t1$ | % | $t2$ | $t1$ | % | $t1$ | % | $t1$ | % |
| $D^+$ | 10 | | **0.0** | **0.0** | **0.0** | 3.9 | **0.0** | 2.7 | **0.0** | 0.3 | **0.0** | 0.0 | **0.0** | **0.0** | 0.7 | **0.0** | 0.1 | **0.4** | 0.0 |
| | 20 | | **46.1** | **111.8** | **0.0** | 8.8 | **0.1** | 5.3 | **2.5** | 1.3 | **39.2** | 0.0 | **0.4** | **0.0** | 12.7 | **0.4** | 10.7 | **12.7** | 6.6 |
| | 30 | | - | - | **0.1** | 8.6 | **4.0** | 5.5 | **569.4** | 1.5 | **5230.8** | 2.8 | **997.8** | **0.1** | 16.5 | **1.8** | 13.1 | **83.9** | 9.4 |
| | 50 | | - | - | **1.4** | 7.7 | **22.3** | 4.1 | **3073.8** | 0.4 | **5433.7** | 3.9 | - | **0.1** | 14.8 | **5.1** | 10.8 | **241.0** | 9.5 |
| $D^-$ | 10 | 6.8 | **0.0** | **0.0** | **0.0** | 9.5 | **0.0** | 7.2 | **0.0** | 7.6 | **0.0** | 6.9 | **0.0** | **0.0** | 7.1 | **0.0** | 6.8 | **0.3** | 6.8 |
| | 20 | 13.1 | **40.7** | **105.8** | **0.0** | 20.9 | **0.0** | 17.2 | **0.9** | 14.3 | **28.1** | 13.5 | **0.4** | **0.0** | 23.6 | **0.5** | 19.1 | **15.8** | 15.7 |
| | 30 | 14.4 | - | - | **0.1** | 21.4 | **1.6** | 19.1 | **325.3** | 14.2 | **5083.6** | 16.4 | **998.3** | **0.1** | 29.6 | **1.9** | 25.7 | **72.9** | 22.7 |
| | 50 | - | - | - | **1.14** | 17.9 | **16.1** | 15.8 | **1849.1** | 11.1 | **5637.4** | 13.5 | - | **0.1** | 27.7 | **5.8** | 24.0 | **303.0** | 20.9 |

TABLE 1 – Comparaison de B&B, LDS (avec $d \in \{0.1, 0.2, 0.5, 1.0\}$), DP et RDP (avec $H \in \{10^3, 10^4, 10^5\}$), pour $l = 6$min, $n \in \{10, 20, 30, 50\}$ et $D^* \in \{D^+, D^-\}$. $t1$ est le temps pour trouver le meilleur tour et $t2$ le temps pour prouver l'optimalité. Pour DP, $t1 = t2$. Pour LDS et RDP, le meilleur tour trouvé $T$ n'est pas nécessairement optimal et $\% = \textit{écart}(T, T^{D^+S6})$. Les tours optimaux calculés avec $D^+$ et $D^-$ sont différents et $\frac{D^-}{D^+} = \textit{écart}(T^{D^-S6}, T^{D^+S6})$. Les temps sont en secondes et '-' indique que le temps est supérieur à 3h.

adaptant l'algorithme de Disjkstra, sinon le problème devient $\mathcal{NP}$-difficile [18].

Si les données réelles de circulation sont naturellement FIFO, le fait de les discrétiser par pas de temps constants peut les rendre non FIFO. Ainsi, nous utilisons les algorithmes de [17,24] pour rendre nos fonctions de coût FIFO avant de calculer les plus courts chemins.

### 3.5 Description des instances du TD-TSP

Nous considérons quatre tailles $n \in \{10, 20, 30, 50\}$. Pour chaque valeur de $n$, nous avons généré 30 instances en sélectionnant au hasard pour chaque instance, $n$ adresses géographiques de la ville de Lyon. Pour chaque couple d'adresses $(u, v)$, chaque jeu de donnée $D^*Sl$ (avec $* \in \{+, -\}$ et $l \in \{6, 12, 24, 60, 720\}$), et chaque heure de départ possible $t$, nous avons calculé la durée de plus court chemin pour aller du $u$ à $v$ en partant de $u$ à l'instant $t$. Nous considérons que toutes les tournées commencent à $t_0 = 7h00$ et ne peuvent pas terminer après $19h00$. Une durée d'arrêt fixe de 6 (resp. 3) minutes est associée à chaque adresse de livraison si $n \in \{10, 20, 30\}$ (resp. $n = 50$).

## 4 Évaluation expérimentale

Dans cette section, nous présentons un aperçu des premiers résultats obtenus (qualité des solutions et temps de calcul) en fonction du jeu de données $D^*Sl$ considéré, avec $* \in \{+, -\}$ et $l \in \{6, 12, 24, 60, 720\}$.

**Conditions expérimentales.** Tous les algorithmes ont été implémentés en `C` et compilés avec `gcc -O3`, à l'exception de l'approche ILP, qui a été codée en `C ++` et a utilisé `Gurobi 8.1.0` [14]. Ce dernier a été exécuté avec ses paramètres par défaut avec un seul thread. Toutes les expérimentations ont été exécutées sur un processeur `Intel (R) Xeon (MD) Platinum 8175M à 2,50 GHz` et `32 Go` de mémoire. Les exécutions sont toutes limitées à trois heures de temps CPU.

**Mesure de performance.** Nous notons $T^{D^*Sl}$ le tour optimal calculé avec le jeu $D^*Sl$ tel que $* \in \{+, -\}$ et $l \in \{6, 12, 24, 60, 720\}$. Pour pouvoir comparer les durées de tours optimaux calculés avec des jeux de données différents, nous évaluons la durée de chaque tour optimal $T^{D^*Sl}$ en utilisant les données de $D^+S6$, et nous notons $ts^{D^+S6}(T^{D^*Sl})$ la durée du tour $T^{D^*Sl}$ évaluée avec les données de $D^+S6$. Cela nous permet de comparer nos tournées dans la même base et avec la meilleure approximation possible des conditions de trafic réelles, puisque $D^+$ est le jeu le plus précis, et que $l = 6$ min est le plus petit pas de temps considéré.

Considérons par exemple la fonction de coût définie en partie droite de la figure 1 (avec $l = 12$), et supposons que cette fonction de coût corresponde au jeu $D^+$. Dans ce cas, le meilleur tour est $T^{D^+S12} = \langle 0, 1, 2, 3, 0 \rangle$, et sa durée est 14. Si on évalue la durée de ce tour avec la fonction de coût définie en partie gauche (avec $l = 6$), on obtient $ts^{D^+S6}(T^{D^+S12}) = 16$. Le meilleur tour avec $l = 6$ est $T^{D^+S6} = \langle 0, 3, 1, 2, 0 \rangle$, et sa durée est $ts^{D^+S6}(T^{D^+S6}) = 12$.

Pour évaluer la qualité d'un tour, nous évaluons son écart en pourcentage au tour optimal sur les données les plus précises, *i.e.*, $T^{D^+S6}$. L'écart en pourcentage entre deux tours $T_1$ et $T_2$ est :

$$\textit{écart}(T_1,T_2) = \frac{ts^{D^+S6}(T_1) - ts^{D^+S6}(T_2)}{ts^{D^+S6}(T_2)} \times 100. \quad (1)$$

Les tours optimaux $T^{D^+S6}$ sont connus pour $n \leq 30$. Pour $n = 50$, aucun algorithme exact ne termine dans la limite de trois heures. Dans ce cas, nous considérons à la place de $T^{D^+S6}$ une solution de référence, qui est la meilleure solution trouvée par toutes les approches dans la limite de 3 heures de temps CPU.

**Passage à l'échelle par rapport à $n$ quand $l = 6$mn.** Le tableau 1 compare les résultats expérimentaux de B&B, LDS, DP, et RDP quand on considère la plus petite granularité temporelle $l = 6$mn. Le paramètre de divergence $d$ du LDS est choisi selon une approche de mise au point parmi les valeurs de 0.1 à 1.5, avec un intervalle régulier de 0.1. DP prouve l'optimalité en moins d'une seconde (resp. 1000 secondes) pour $n \leq 20$ (resp. $n = 30$). Pour $n = 50$, DP ne peut plus être utilisé à cause de sa complexité spatiale.

B&B est moins performant que DP, et nécessite plus de trois heures pour $n = 30$. En revanche, LDS calcule rapidement de bonnes approximations. Considérons tout d'abord les résultats obtenus avec $D^+$. Pour $n = 30$ (resp. $n = 50$) et pour une divergence $d = 0.2$, nous obtenons des tours à 5.5% de l'optimum en 4 secondes (resp. à 4.1% du meilleur tour en 22 secondes). Les tours optimaux calculés avec les données spatialement dégradées, $D^-$, sont 6.8% plus longs que ceux calculés avec les données spatialement complètes quand $n = 10$, et cet écart augmente quand $n$ augmente pour atteindre 14.4% quand $n = 30$. Cela montre l'importance de disposer d'une information spatiale complète pour le modèle prédictif utilisé. Les solutions approchées calculées avec LDS ont des écarts supérieurs, et ces écarts tendent à augmenter quand on diminue $d$ (sauf pour les plus grosses instances). RDP n'est pas compétitif avec LDS : il met plus de temps pour calculer de moins bonnes solutions.

Dans la figure 3, nous comparons l'évolution en fonction du temps de la qualité des solutions calculées par LDS (avec $d \in \{.1, .2, .5, 1\}$) quand $n = 50$. Des solutions de bonne qualité sont obtenues rapidement avec $d = .5$, mais pour de plus longues limites de temps de meilleures solutions sont trouvées avec de plus grandes valeurs de $d$. A noter que dans la limite de 3 heures, LDS avec $d = 1$ est moins performant que LDS avec $d = .5$ car il met trop de temps à converger.

**Impact de la granularité spatio-temporelle.** Le TSP statique (quand $l = 720$) est plus facile à résoudre que le TD-TSP, et ILP peut résoudre efficacement des instances avec des centaines de noeuds [28]. Aussi convient-il d'examiner l'intérêt d'optimiser des tours avec des coûts dépendant du temps. En d'autres termes, pouvons-nous trouver de meilleures solutions plus rapidement en optimisant les tours avec des coûts constants ? Plus généralement, quel est l'impact de la granularité de $l$ sur l'évolution de la qualité des tours en fonction du temps de calcul ? Pour répondre à cette question, la figure 4 compare l'évolution de l'écart entre le meilleur tour trouvé et le tour optimal $T^{D^+S6}$ en fonction du temps, pour différentes granu-

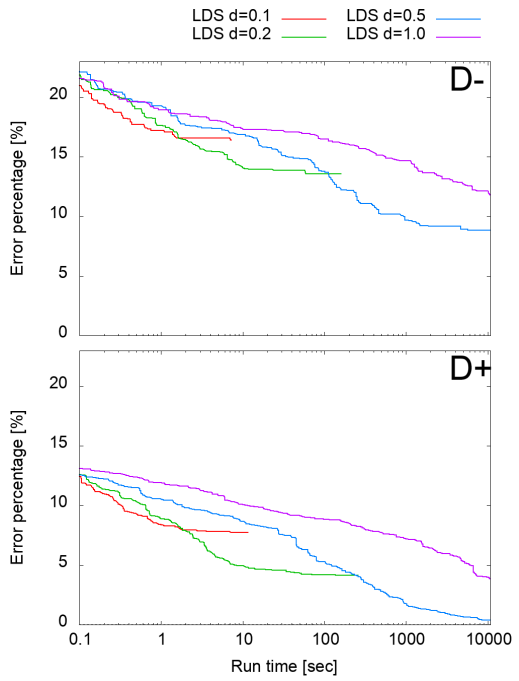



FIGURE 3 – Impact du paramètre $d$ de LDS quand $n = 50$. Pour chaque point $(x, y)$, $y = \textit{écart}(T_x, T^{D^+S6})$ où $T_x$ est le meilleur tour trouvé par LDS en $x$ secondes avec le jeu $D^-S6$ (en haut) et $D^+S6$ (en bas).

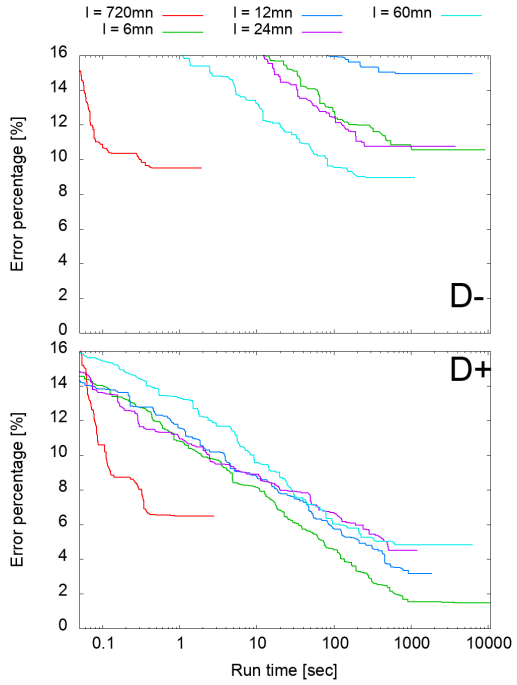



FIGURE 4 – Impact de la longueur $l$ du pas de temps quand $n = 30$. Pour chaque point $(x, y)$, $y = \textit{écart}(T_x, T^{D^+S6})$ où $T_x$ est le meilleur tour trouvé en $x$ secondes avec le jeu $D^-Sl$ (en haut) et $D^+Sl$ (en bas).

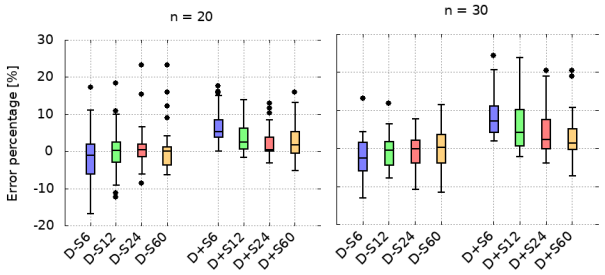



FIGURE 5 – Distribution de *écart*$(T^{D^*S720}, T^{D^*Sl})$ pour $l \in \{6, 12, 24, 60\}$, $* \in \{-, +\}$, et $n = \{20, 30\}$.

larités $l \in \{6, 12, 24, 60, 720\}$ et pour les deux jeux de données $D^+$ et $D^-$. Quand $l = 720$, le TSP est statique et nous utilisons une approche ILP *Branch&Cut* dans laquelle les sous-tours sont éliminés en commençant par ceux ayant la plus petite cardinalité [27]. Cette approche nous permet de trouver la solution optimale en moins de 0.4s en moyenne pour $n = 30$. Quand $l < 720$, nous utilisons l'approche LDS avec $d = 0.5$ qui est l'approche présentant le meilleur compromis entre qualité et temps de résolution.

ILP résolvant très rapidement le problème statique, les tours calculés avec $l = 720$ sont meilleurs que les tours calculés par LDS avec $l < 720$, pour des petites limites de temps. En revanche, pour des limites de temps supérieures à 100s et quand on utilise les données spatialement complètes $D^+$, les tours optimisés avec $l = 720$ sont 6% plus longs, en moyenne, que ceux optimisés avec $l = 6$. Cela montre l'intérêt d'optimiser avec des coûts dépendants du temps, même si cela rend le problème plus difficile. Cependant, cet intérêt diminue si on augmente $l$, et il devient nul si on considère les données spatialement incomplètes $D^-$.

Afin d'examiner plus en détails l'écart entre la solution statique optimale $T^{D^*S720}$, et celle calculée avec $l \in \{6, 12, 24, 60\}$, *i.e.*, $T^{D^*Sl}$, nous visualisons la distribution de *écart*$(T^{D^*S720}, T^{D^*Sl})$ sur la figure 5. Les valeurs supérieures à 0 correspondent à des instances où le tour calculé avec des données statiques est moins bon que le tour calculé avec des données dépendantes du temps. Considérons tout d'abord le cas où les données sont spatialement complètes ($D^+$). Dans ce cas, l'écart médian baisse quand $l$ augmente, mais il est toujours positif, et le fait d'utiliser des données dépendantes du temps peut faire gagner jusqu'à plus de 20% pour certaines instances. Quand $l \geq 12$ l'écart peut devenir négatif pour certaines instances, mais la perte est toujours inférieure à 8%. Quand on compare les distributions des écarts pour $n = 20$ avec celles pour $n = 30$, on remarque que les écarts ont tendance à augmenter quand on augmente $n$.

En revanche, quand les données sont spatialement incomplètes ($D^-$), l'écart médian est proche de 0, et pour certaines instances la perte peut être supérieure à 10%. Dans ce cas, il n'est pas intéressant d'optimiser les tours avec des données dépendantes du temps : le problème du TD-TSP est plus difficile à résoudre, et les tours ne sont pas nécessairement meilleurs lorsqu'on les évalue avec les données les plus proches possibles des conditions qui seront observées au moment de la réalisation du tour.

En étudiant la position des capteurs considérés dans le jeu $D^-$ (correspondant aux capteurs effectivement déployés sur le réseau routiers lyonnais), nous constatons que ces capteurs sont souvent placés sur les axes congestionnés du réseau, ce qui conduit à une surestimation des durées de traversée des tronçons non équippés de capteurs (puisque ces durées sont estimées par interpolation par rapport aux tronçons équippés de capteurs, généralement plus congestionnés).

## 5 Conclusion

Notre étude expérimentale montre à la fois la difficulté et l'intérêt d'évaluer des algorithmes sur des données réelles. Les données réelles de trafic sont généralement très incomplètes car de nombreux tronçons ne sont pas équipés de capteurs (et certains capteurs existants sont mal positionnés par rapport aux feux de circulation de sorte qu'ils ne donnent pas une bonne indication de la vitesse effective). Notons que les données issues de traceurs GPS embarqués dans des véhicules ne sont généralement pas plus complètes. La plateforme SYMUVIA nous a permis d'obtenir des données à la fois réalistes et complètes, et ces données nous ont permis de montrer qu'il est intéressant d'optimiser des tournées avec des données dépendantes du temps dans le cas où les données sont parfaites (*i.e.*, les prévisions sont identiques aux conditions qui seront observées au moment de réaliser la tournée), et dans le cas où on considère une granularité temporelle fine (*i.e.*, des pas de temps de 6 minutes). Dans ce cas, les tournées optimisées avec des données dépendantes du temps sont en moyenne 6.5% plus rapides que celles optimisées avec des données statiques, et le gain peut aller jusqu'à plus de 20% pour certaines instances quand $n = 30$. En revanche, l'intéret diminue lorsque la longueur du pas de temps $l$ augmente, et il devient nul lorsque les données sont spatialement incomplètes et que les données manquantes sont obtenues par interpolation. Nous étudions actuellement l'impact de la position des capteurs sur la qualité des tournées calculées. Notre objectif est de trouver les meilleures positions, permettant de capturer une image plus fidèle des conditions de trafic, et permettant de calculer de meilleures tournées avec les données issues de ces capteurs.

Notre étude expérimentale a également montré que

le TD-TSP est un problème difficile pour lequel les approches ILP et CP ne passent pas à l'échelle dans le cas où il n'y a pas de plages horaires associées aux livraisons. Dans ce cas, l'approche exacte la plus efficace est DP, mais cette approche est limitée aux instances ayant au plus 30 points de livraison. L'approche heuristique LDS permet d'obtenir plus rapidement de meilleures approximations que l'approche RDP, mais il reste encore de la marge pour améliorer ces approches. En particulier, nous étudions des pistes pour améliorer RDP en intégrant une information heuristique au moment de choisir les $H$ états intermédiaires mémorisés à chaque étape. Nous souhaitons également évaluer l'efficacité des approches basées sur la recherche locale, connues pour être efficaces pour le TSP classique.

Nous souhaitons par ailleurs étendre cette étude au TD-TSPTW, pour lequel il y a des plages horaires associées aux livraisons. Pour ce problème, les approches ILP récentes passent bien mieux à l'échelle et il serait intéressant d'évaluer leurs performances sur nos jeux de données.

## Références

[1] A. Arigliano, G. Ghiani, A. Grieco et E. Guerriero : Time dependent traveling salesman problem with time windows : Properties and an exact algorithm. 2015.

[2] J. C. Beck et L. Perron : Discrepancy-bounded depth first search. *In CPAIOR*, pages 8–10, 2000.

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

[4] N. Boland, M. Hewitt, D. Vu et M. Savelsbergh : Solving the traveling salesman problem with time windows through dynamically generated time-expanded networks. *In CPAIOR*, pages 254–262. Springer, 2017.

[5] E. Burns, K. Rose et W. Ruml : Best-first search for bounded-depth trees. *In Proceedings of SOCS.* AAAI, 2011.

[6] G. Carpaneto et P. Toth : Primal-dual algrorithms for the assignment problem. *Discrete Applied Mathematics*, 18(2):137 – 153, 1987.

[7] N. Chiabaut, M. Kung, M. Menendez et L. Leclercq : Perimeter control as an alternative to dedicated bus lanes : A case study. *Transportation Research Record*, 2018.

[8] J.-F. Cordeau, G. Ghiani et E. Guerriero : Analysis and branch-and-cut algorithm for the time-dependent travelling salesman problem. *Transportation Science*, 48:46–58, 2014.

[9] A. V. Donati, R. Montemanni, N. Casagrande, A. E. Rizzoli et L. M. Gambardella : Time dependent vehicle routing problem with a multi ant colony system. *EJOR*, 185(3):1174–1191, mar 2008.

[10] H. Kanoh et J. Ochiai : Solving time-dependent traveling salesman problems using ant colony optimization based on predicted traffic. *In DCAI*, volume 151, pages 25–32. Springer, 2012.

[11] A. Stentz G. Ayorkor Korsah et M Bernardine Dias : The dynamic hungarian algorithm for the assignment problem with changing costs. Rapport technique CMU-RI-TR-07-27, Carnegie Mellon University, Pittsburgh, PA, July 2007.

[12] M. Gendreau, G. Ghiani et E. Guerriero : Time-dependent routing problems : A review. *Computers & Operations Research*, 64:189–197, dec 2015.

[13] L. Gouveia et S. Voss : A classification of formulations for the (time-dependent) traveling salesman problem. *EJOR*, 83(1):69–82, may 1995.

[14] LLC Gurobi Optimization : Gurobi optimizer reference manual, 2018.

[15] W. D. Harvey et M. L. Ginsberg : Limited discrepancy search. *In IJCAI*, 1995.

[16] M. Held et R. M. Karp : A dynamic programming approach to sequencing problems. *In Proceedings of the 1961 16th ACM National Meeting*, ACM, pages 71.201–71.204, 1961.

[17] S. Ichoua, M. Gendreau et J-Y Potvin : Vehicle dispatching with time-dependent travel times. *EJOR*, 144(2):379–396, 2003.

[18] D. E. Kaufman et R. L. Smith : Fastest paths in time-dependent networks for intelligent vehicle-highway systems application. *Journal of Intelligent Transportation Systems*, 1:1–11, 1993.

[19] J. A. Laval et L. Leclercq : Microscopic modeling of the relaxation phenomenon using a macroscopic lane-changing model. *Transportation Research Part B : Methodological*, 42(6):511–522, 2008.

[20] L. Leclercq : Bounded acceleration close to fixed and moving bottlenecks. *Transportation Research Part B : Methodological*, 41(3):309–319, 2007.

[21] L. Leclercq, N. Chiabaut et B. Trinquier : Macroscopic fundamental diagrams : A cross-comparison of estimation methods. *Transportation Research Part B : Methodological*, 62:1 – 12, 2014.

[22] C. Malandraki et M. S. Daskin : Time dependent vehicle routing problems : Formulations, properties and heuristic algorithms. *Transportation Science*, 26(3):185–200, 1992.

[23] C. Malandraki et R. B. Dial : A restricted dynamic programming heuristic algorithm for the time dependent traveling salesman problem. *EJOR*, 90(1):45 – 55, 1996.

[24] P. Melgarejo, P. Laborie et C. Solnon : A time-dependent no-overlap constraint : Application to urban delivery problems. *In CPAIOR*, volume 9075 de *LNCS*, pages 1–17. Springer, 2015.

[25] A. Montero, I. Méndez-Díaz et J. Miranda-Bront : An integer programming approach for the time-dependent traveling salesman problem with time windows. *Computers & Operations Research*, 88:280–289, 2017.

[26] G. Newell : A simplified car-following theory : a lower order model. *Transportation Research Part B : Methodological*, 36(3):195–205, 2002.

[27] Staněk R. Pferschy U. : Generating subtour elimination constraints for the tsp from pure integer solutions. *Central European journal of operations research*, 25:231–260, 2017.

[28] R. Roberti et P. Toth : Models and algorithms for the asymmetric traveling salesman problem : an experimental comparison. *EURO J. Transportation and Logistics*, 1(1-2):113–133, 2012.

[29] J. Salotti, S. Fenet, R. Billot, N.-E. El Faouzi et C. Solnon : Comparison of traffic forecasting methods in urban and suburban context. *In ICTAI.* IEEE, 2018.

[30] J. Schneider : The time-dependent traveling salesman problem. *Physica A : Statistical Mechanics and its Applications*, 314(1):151–155, 2002.

[31] G. Stecco, J-F Cordeau et E. Moretti : A tabu search heuristic for a sequence-dependent and time-dependent scheduling problem on a single machine. *J. Scheduling*, 12(1):3–16, 2009.

[32] D. Vu, M. Hewitt, N. Boland et M. Savelsbergh : Solving time dependent traveling salesman problems with time windows, 2018.

# Unifier les stratégies de sélection de réserve avec la programmation par contraintes et les graphes.

**Dimitri Justeau-Allaire**[1,2,3*] **Philippe Birnbaum**[1,2,3] **Xavier Lorca**[4]

[1] CIRAD, UMR AMAP, F-34398 Montpellier, France
[2] Institut Agronomique néo-Calédonien (IAC), 98800 Noumea, New Caledonia
[3] AMAP, Univ Montpellier, CIRAD, CNRS, INRA, IRD, Montpellier, France
[4] ORKID, Centre de Génie Industriel, IMT Mines Albi
Campus Jarlard, 81013 Albi cedex 09, France

{dimitri.justeau-allaire,philippe.birnbaum}@cirad.fr xavier.lorca@mines-albi.fr

**Résumé**

Concilier la conservation des habitats naturels avec le développement socioéconomique constitue aujourd'hui un défi majeur. Le problème de sélection de réserve se propose d'y répondre avec une approche systématique. Cependant, les modèles existants se focalisent sur des variantes précises du problème et ne permettent pas de prendre en compte l'ensemble des contraintes auxquelles les gestionnaires peuvent être confrontés. Dans un article présenté à CP 2018 [4], nous avons montré comment la programmation par contraintes (CP) et la modélisation basée sur les graphes permet d'agréger les différents aspects du problème dans un modèle unique et flexible, puis nous avons validé ce modèle sur un cas d'utilisation en Nouvelle-Calédonie. Nous résumons ici cet article, et introduisons les perspectives qu'il ouvre dans le domaine de l'aide à la décision pour la conservation de la nature.

## 1 Le problème de sélection de réserve

Le problème de *sélection de réserve*[1] a été introduit entre le milieu des années 1970 et le début des années 1980 (cf. [8] pour une revue historique sur le sujet). C'est un problème combinatoire qui s'applique dans un espace géographique maillé $\mathcal{P}$, généralement selon une grille carrée. L'objectif est de mettre en évidence une zone qui répond à un ensemble de contraintes écologiques et socioéconomiques, avec ou sans optimisation. Cette zone est composée de réserves, chacune composée d'un ensemble de cellules connectées selon un voisinage défini dans la maille. Un exemple est fourni dans la Figure 1 et deux voisinages dans une grille carrée sont illustrés dans la Figure 2.

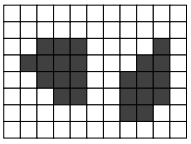


FIGURE 1 – Exemple de solution au problème de sélection de réserve : deux réserves.

On distingue deux catégories de contraintes : les contraintes de couverture et les contraintes spatiales.

**Contraintes de couverture** L'espace géographique $\mathcal{P}$ est caractérisé par un ensemble de caractéristiques spatiales $\mathcal{F}$. Pour chaque caractéristique $j \in \mathcal{F}$, une valeur numérique $v_{ji}$ (binaire, quantitative ou probabiliste) est définie pour chaque cellule $i \in \mathcal{P}$. Les caractéristiques spatiales peuvent représenter l'occupation du sol (e.g. écosystèmes), la biodiversité (e.g. espèces observées), ou des valeurs socioéconomiques (e.g. zones exploitables). Les contraintes de couverture s'appliquent sur ces caractéristiques et permettent de définir les seuils de couverture minimaux et/ou maximaux que la zone doit couvrir pour chacune d'elles.

**Contraintes spatiales** Les contraintes spatiales permettent de contrôler la configuration spatiale de la zone à délimiter, selon les attributs définis par

*Papier doctorant : Dimitri Justeau-Allaire[1,2,3] est auteur principal.

1. Aussi connu comme *conception de réserve*, ou *planification systématique de la conservation* [5].

Williams et al. [9] : *nombre de réserves* (i.e. de zones connectées), *surface des réserves*, *surface totale*, distance entre les réserves, connectivité dans les réserves, forme des réserves et zones tampon autour des réserves. Dans le cadre de ce travail nous nous sommes limités aux contraintes représentées en italique, les autres feront l'objet de travaux futurs.

(a) Voisinage 4-connecté. (b) Voisinage 8-connecté.

FIGURE 2 – Deux voisinages dans une grille carrée (en gris : voisinage de $x$).

## 2 Un modèle CP unifié

Nous avons proposé un modèle CP permettant d'intégrer les contraintes de couverture et les contraintes spatiales de manière transparente. Dans ce modèle, chaque cellule $i$ de la maille est représentée par une variable booléenne $b_i$. La cellule $i$ est dans la zone à délimiter si et seulement si $b_i = 1$. Ces variables sont les variables de décision du modèle et il est possible d'y exprimer directement les contraintes de couverture via des contraintes de flot ou des contraintes de couverture par ensembles. Afin d'exprimer les contraintes spatiales, nous avons introduit $G = (\mathcal{P}, E)$, un graphe spatial non dirigé qui représente la maille. Un sommet est associé à chaque cellule $i \in \mathcal{P}$ et $E$ est défini tel qu'il existe une arrête $(u, v)$ si et seulement si $u$ et $v$ sont adjacents selon le voisinage choisi. A partir de $G$ nous avons défini la variable graphe $g = (v, e) \in [\varnothing, G]$, chaînée avec les variables de décision de manière à ce que $b_i = 1 \Leftrightarrow i \in v \wedge (i, j) \in e \Leftrightarrow b_i = b_j = 1$. La variable $g$ permet alors d'exprimer les contraintes de connectivité et celles sur la taille des réserves via ses composantes connexes. Ce modèle a été implémenté avec le solveur `Choco` [7] et son extension `Choco-graph` [2] qui permet l'utilisation de variables graphes.

## 3 Cas d'utilisation : fragmentation forestière au sud de la Nouvelle-Calédonie

La Nouvelle-Calédonie est un archipel tropical situé dans la région du Pacifique Sud. C'est un point chaud de la biodiversité [6] dont la flore se caractérise par un taux d'endémisme exceptionnellement élevé. Malheureusement, les forêts néo-calédoniennes ne sont pas épargnées par la déforestation et elles sont menacées par la fragmentation, qui fragilise les communautés végétales et animales. Pour notre cas d'utilisation, nous nous sommes basés sur une étude d'Ibanez et al. conduite dans le sud de la Nouvelle-Calédonie [3]. Celle-ci visait à mieux comprendre l'impact de la fragmentation sur les communautés d'arbres dans une zone où la biodiversité est menacée par l'extraction du nickel, un des pilier de l'économie néo-calédonienne.

En concertation avec les écologues à l'origine de cette étude, nous avons considéré un scénario fictif basé sur des données réelles. Dans celui-ci, il est question de protéger une zone qui couvre 223 espèces d'arbres, au moins 340ha de forêt, qui est formée d'au plus deux réserves d'au moins 300ha et dont la surface totale n'excède pas 1000ha. Afin de minimiser les coûts d'établissement et de gestion, nous avons cherché à minimiser la surface totale de la zone à protéger. La meilleure solution obtenue (sans preuve d'optimalité) est illustrée dans la Figure 3.

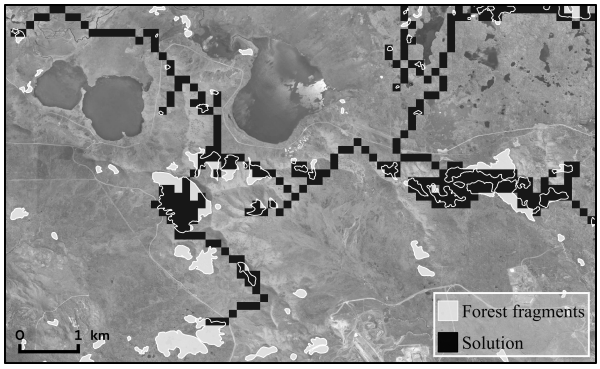



FIGURE 3 – Meilleure solution obtenue dans le cas d'utilisation : environ 496ha, 1 réserve.

## 4 Conclusions et perspectives

Dans cet article, nous avons introduit le premier modèle CP permettant d'aborder le problème de sélection de réserve en combinant des contraintes spatiales et de couverture. Nous avons montré que cette technique permet d'unifier les différents aspects du problème dans un modèle à partir duquel nous avons pu traiter un cas d'étude basé sur des données réelles. Ce travail a été présenté aux autorités néo-calédoniennes et ouvre différentes perspectives en matière d'aide à la décision pour la conservation de la nature. Parmi celles-ci, les priorités pour nos travaux futurs concernent la modélisation de contraintes spatiales complexes (e.g. zone tampon) et la généralisation du modèle en un problème de partitionnement. Cette approche permettrait ainsi d'exprimer des contraintes sur des zones correspondant à différents niveaux de protection, en accord avec les lignes conductrices de l'IUCN pour la gestion des aires protégées [1].

## Références

[1] Nigel Dudley : *Guidelines for Applying Protected Area Management Categories*. IUCN, 2008.

[2] Jean-Guillaume Fages, Charles Prud'homme et Xavier Lorca : Choco Graph Documentation, février 2018.

[3] Thomas Ibanez, Vanessa Hequet, Céline Chambrey, Tanguy Jaffré et Philippe Birnbaum : How does forest fragmentation affect tree communities ? A critical case study in the biodiversity hotspot of New Caledonia. *Landscape Ecology*, 32(8) :1671–1687, août 2017.

[4] Dimitri Justeau-Allaire, Philippe Birnbaum et Xavier Lorca : Unifying Reserve Design Strategies with Graph Theory and Constraint Programming. *In* John Hooker, éditeur : *Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, pages 507–523. Springer International Publishing, 2018.

[5] C. R. Margules et R. L. Pressey : Systematic conservation planning. *Nature*, 405(6783) :243–253, mai 2000.

[6] Norman Myers, Russell A. Mittermeier, Cristina G. Mittermeier, Gustavo A. B. da Fonseca et Jennifer Kent : Biodiversity hotspots for conservation priorities. *Nature*, 403(6772) :853–858, février 2000.

[7] Charles Prud'homme, Jean-Guillaume Fages et Xavier Lorca : *Choco Documentation*. 2017.

[8] Sahotra Sarkar : Complementarity and the selection of nature reserves : Algorithms and the origins of conservation planning, 1980–1995. *Archive for History of Exact Sciences*, 66(4) :397–426, juillet 2012.

[9] Justin C. Williams, Charles S. ReVelle et Simon A. Levin : Spatial attributes and reserve design models : A review. *Environmental Modeling & Assessment*, 10(3) :163–181, septembre 2005.

# Une approche de Programmation par Contraintes pour résoudre le Problème de Transport de Patients

**Charles Thomas**[*1] **Quentin Cappart**[2,3] **Pierre Schaus**[1] **Louis-Martin Rousseau**[2,3]

[1] UCLouvain, Institute of Information and Communication Technologies, Electronics and Applied Mathematics (ICTEAM), Belgique

[2] Ecole Polytechnique de Montréal, Canada

[3] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT), Canada

{charles.thomas,pierre.schaus}@uclouvain.be

{quentin.cappart,louis-martin.rousseau}@{polymtl.ca,cirrelt.ca}

**Résumé**

Le Problème de Transport de Patients (PTP) consiste à transporter des patients à des rendez-vous médicaux. Dans cet article, nous proposons une approche de Programmation par Contraintes capable de résoudre ce problème et suffisamment flexible pour admettre d'autres contraintes. De plus, nous introduisons une stratégie de recherche qui maximise de manière efficace le nombre de requêtes sélectionnées. Nos résultats montrent que l'approche proposée peut résoudre des instances réelles et est plus performante que les stratégies gloutonnes utilisées par des planificateurs humains.

Le problème considéré dans cet article (l'article original a été présenté à la conférence CP 2018 [1]) a été proposé par une association sans but lucratif opérant à Liège (Belgique) qui fournit un service de transport pour rendez-vous médicaux. Nous y référons sous le nom de Problème de Transport de Patient. Il s'agit d'une variante spécifique de problème de transport à la demande (DARP - Dial-A-Ride Problem) [2]. Le but de ce problème est de définir des itinéraires et horaires pour un ensemble d'utilisateurs qui spécifient des requêtes de transport entre deux endroits. Les travaux les plus récent sur cette variante du problème incluent l'article original ainsi qu'un article de Liu et al. sur le Problème de Transport de Seniors [3].

Le Problème de Transport de Patients ajoute plusieurs contraintes au problème de transport à la demande original : les temps auxquels les patients sont transportés sont limités étant donné qu'ils doivent arriver à temps pour leur rendez-vous et que leur temps d'attente doit être limité. De plus, les requêtes de patients sont hétérogènes. Certains patients ne nécessitent qu'un seul trajet aller tandis que d'autres ont aussi besoin d'un retour après le rendez-vous. Les requêtes peuvent impliquer plus d'un passager et peuvent nécessiter un véhicule spécifique pour embarquer des patients handicapés. La flotte de véhicules est également hétérogène. En effet, les véhicules diffèrent de par leur capacité, localisation de départ ou d'arrivée (dépôts) et horaires de disponibilité.

Le problème est considéré comme statique : l'ensemble des requêtes est connu à l'avance. La solution développée est utilisée par l'association dans le but de planifier son horaire opérationnel journalier étant donné les requêtes reçues la veille. L'objectif est de maximiser le nombre de patients pouvant être transportés sur la journée. Trois niveaux de décision sont considérés : sélectionner quelles requêtes seront satisfaites, assigner des véhicules à ces requêtes et planifier l'itinéraire des véhicules.

La contribution de cet article est une approche flexible et efficace utilisant la Programmation par Contraintes (CP) pour modéliser et résoudre le Problème de Transport de Patients statique. Le modèle proposé est basé sur une logique d'ordonnancement : transporter un patient d'un endroit à un autre est modélisé comme une activité. Un véhicule est vu comme une ressource cumulative qui peut effectuer plusieurs

[*] Papier doctorant : Charles Thomas est auteur principal.

activités simultanément.

Par exemple, un véhicule transportant deux patients se rendant au même hôpital est représenté dans le modèle de la manière suivante : le véhicule visit d'abord la localisation du premier patient afin de le ramasser ce qui correspond au début d'une première activité représentant le transport du premier patient. Ensuite, le véhicule se rend chez le second patient afin de le prendre ce qui correspond au début d'une seconde activité représentant le transport du second patient. Finalement, le véhicule va à l'hôpital et dépose les deux patients, terminant ainsi les deux activités.

Les contraintes de temps de transport entre les différents endroits sont implémentées comme des temps de transitions entre les débuts et fins des activités réalisées par un même véhicule. La contrainte de chargement de chaque véhicule est vérifiée par une contrainte cumulative globale [4]. Plusieurs extensions telles que un temps de service pour embarquer ou désembarquer d'un véhicule ou bien des requêtes obligatoires peuvent être ajoutées facilement au modèle.

Une stratégie de recherche générique pour maximiser le nombre de requêtes sélectionnés est également proposée. Celle-ci est similaire à la recherche proposée dans [5]. Elle consiste à brancher en priorité sur les variables de sélection de requête. Les variables correspondant aux activités ne sont explorées que si la requête correspondante a été sélectionnée. Cela permet d'éviter d'explorer une partie de l'arbre de recherche dans les branches où une requête n'est pas sélectionnée. Une recherche à voisinage large (LNS - Large Neighbourhood Search) [6] est utilisée afin d'améliorer les performances sur de grandes instances.

Les performances de l'approche proposée ont été comparées avec un algorithme glouton et un modèle successeur, une approche standard pour résoudre des problèmes de tournées de véhicules en programmation par contraintes. Les résultats expérimentaux montrent que l'approche par ordonnancement surpasse les performances de la stratégie gloutonne et du modèle successeur. La solution proposée est utilisable en pratique grâce à son efficacité sur des grandes instances et à sa flexibilité pour être adaptée à de nouvelles situations.

### Remerciements

Ce projet de recherche est financé par la Région Wallonne dans le cadre du projet PRESupply. Le problème a été proposé par la Centrale de Services à Domicile (CSD), une association sans but lucratif affiliée à la mutuelle Solidaris et opérant à Liège.

## Références

[1] Cappart, Q., Thomas, Ch., Schaus P. et Rousseau, L.-M. *A Constraint Programming Approach for Solving Patient Transportation Problems.* International Conference on Principles and Practice of Constraint Programming, 2018.

[2] Cordeau, J.-F. and Laporte, G. *The dial-a-ride problem : models and algorithms.* Annals of Operations Research, 2007.

[3] Liu, C., Aleman, D. M., et Beck, J. C. *Modelling and solving the senior transportation problem.* International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research, 2018.

[4] Gay, S., Hartert, R. et Schaus, P. *Simple and scalable time-table Filtering for the cumulative constraint.* International Conference on Principles and Practice of Constraint Programming, 2015.

[5] Barco, A. F., Fages, J. G., Vareilles, E., Aldanondo, M., et Gaborit, P. *Open packing for facade-layout synthesis under a general purpose solver.* International Conference on Principles and Practice of Constraint Programming, 2015.

[6] Shaw, P. *Using constraint programming and local search methods to solve vehicle routing problems.* International Conference on Principles and Practice of Constraint Programming, 1998.

# Octogones entiers pour le problème RCPSP

**Pierre Talbot**[1] **David Cachera**[2] **Éric Monfroy**[1] **Charlotte Truchet**[1]

[1] Université de Nantes, LS2N, 44000 Nantes, France
[2] Univ Rennes, Inria, CNRS, IRISA, 35000 Rennes, France
{prénom.nom}@univ-nantes.fr david.cachera@irisa.fr

**Résumé**

En programmation par contraintes, l'efficacité des solveurs est principalement due aux contraintes globales qui encapsulent des algorithmes de résolution spécifiques. Une conséquence est la prolifération de centaines de contraintes globales qui sont spécialisées pour des problèmes très particuliers. Dans cet article, nous étudions une approche basée sur les domaines abstraits de l'analyse de programmes par interprétation abstraite. Les domaines abstraits permettent de résoudre efficacement des conjonctions de contraintes primitives dans un langage fixé. Essentiellement, nous utilisons une décomposition de la contrainte globale `cumulative` vers des conjonctions de contraintes primitives pouvant être prises en charge par les domaines abstraits. Le premier avantage est que ces derniers sont moins nombreux car plus généralistes, et peuvent donc être réutilisés sur une gamme de problèmes plus large. Le deuxième est que ces domaines abstraits peuvent échanger des contraintes grâce à la technique du produit réduit issue de l'interprétation abstraite. La contribution principale est d'utiliser les contraintes réifiées afin de programmer cet échange d'information entre le domaine abstrait des intervalles et des octogones. Nous explicitons cette approche sur le problème de gestion de projet à contraintes de ressources (RCPSP). Les expérimentations préliminaires montrent que l'approche offre un bon compromis entre efficacité et expressivité.

## 1 Introduction

L'efficacité d'un solveur de contraintes tient principalement à la présence de contraintes globales permettant d'encapsuler des algorithmes spécifiques à une sous-structure d'un problème. Par exemple, la contrainte globale `alldifferent`$(x_1, \ldots, x_n)$ s'assure que les variables $x_1, \ldots, x_n$ sont toutes différentes. On recense presque 400 contraintes globales [2] possédant chacune plusieurs algorithmes de propagation plus ou moins efficaces. Ce qui fait à la fois la force et la faiblesse des contraintes globales est leur flexibilité : on peut toujours créer une nouvelle contrainte globale capturant un problème spécifique, au prix de passer plus de temps dans l'implémentation algorithmique que sur la partie modélisation. De plus, la plupart des solveurs de contraintes n'implémentent que quelques dizaines de contraintes globales—les plus courantes. Un dernier problème est l'absence de communication forte entre deux contraintes globales identiques. Par exemple, si un modèle contient deux contraintes `alldifferent` partageant des variables, cette information est difficilement prise en compte dans les solveurs classiques. Une solution est de passer par des algorithmes de consistance plus fortes que ceux implémentés dans les solveurs classiques [3]. Une autre solution est de créer une nouvelle contrainte globale agrégeant plusieurs `alldifferent`, qui par ailleurs existe et s'appelle `k_alldifferent`.

Dans cet article, nous étudions une approche basée sur les domaines abstraits venant du domaine de l'analyse de programme par interprétation abstraite [7]. Au lieu de capturer des sous-structures du problème, un domaine abstrait capture des sous-théories de la programmation par contraintes [20]. Une théorie représente une conjonction de contraintes dans un langage fixé. Par exemple, on peut imaginer le domaine abstrait `Diff` qui gère les contraintes de la forme $x \neq y$. Dans ce cas, toutes les contraintes $\neq$ du modèle seront propagées dans ce domaine abstrait par un algorithme efficace. Le domaine abstrait `Diff` subsume les deux contraintes `alldifferent` et `k_alldifferent`. Un premier avantage est que les domaines abstraits sont moins dépendants d'un problème en particulier, et peuvent donc être réutilisés sur des problèmes avec des sous-structures très différentes. Un deuxième avantage est que les domaines abstraits peuvent communiquer entre eux grâce à la technique du produit réduit de l'interprétation abstraite.

Il existe une multitude de produits réduits entre deux

domaines et il doivent généralement être programmés à la main. La contribution de cet article est de proposer un produit réduit générique entre deux domaines abstraits en utilisant des contraintes d'équivalence de la forme $c_1 \Leftrightarrow c_2$, également appelées *contraintes réifiées*. Soit deux domaines abstraits $A$ et $B$, ces contraintes réifiées permettent de faire le pont entre les contraintes $c_1$ appartenant à $A$ et $c_2$ appartenant à $B$. Afin d'obtenir ce produit réduit automatiquement, nous étendons la définition d'un domaine abstrait avec un opérateur de déduction (*entailment*). Cet opérateur est ensuite utilisé pour programmer le propagateur de la contrainte réifiée.

Nous illustrons cette méthode en proposant d'étudier le produit réduit du domaine des intervalles et des octogones afin de résoudre le problème de gestion de projet à contraintes de ressources (RCPSP) [9, 10], que nous introduisons en Section 2. L'idée principale est d'utiliser une décomposition de la contrainte globale `cumulative` pour qu'elle puisse être gérée par le domaine des octogones [22]. Les contraintes restantes, non octogonales, peuvent être prise en charge par le domaine des intervalles (également appelés « boîtes ») qui peut être vu comme le domaine de base en programmation par contraintes. Une boite discrète est un produit Cartésien d'intervalles de la forme $[l, u]_x \in \mathbb{N}^2$ tel que la variable $x$ peut être affectée à une valeur dans l'ensemble $\{v \mid l \leq v \leq u\}$.

Les définitions nécessaires d'interprétation abstraite sont introduites en Section 3 ainsi que les octogones entiers en Section 4. Le produit réduit par contraintes réifiées est donné en Section 5. Nous fournissons une implémentation [1] de ces domaines abstraits et testons son efficacité sur une série de *benchmarks* standard pour le RCPSP (Section 6).

Finalement, afin d'éviter toute ambiguïté, notons de suite que l'idée des domaines abstraits est très proche des solveurs *Satisfiability Modulo Theories* (SMT). La différence principale est que les domaines abstraits sont basés sur la théorie des treillis plutôt que sur un fondement logique comme en SMT. Des travaux récents connectant les deux domaines sont disponibles [8].

## 2 Le problème RCPSP

Le problème de gestion de projet à contraintes de ressources (RCPSP) est un problème classique d'ordonnancement de tâches sous ressources. Un problème RCPSP est un triplet $(T, P, R)$ où $T$ est un ensemble de tâches, $P$ est l'ensemble des priorités entre les tâches, que nous noterons $i \ll j$ pour indiquer que la tâche $i$ doit terminer avant que $j$ commence, et $R$ est l'ensemble des ressources. Chaque tâche $i \in T$ a une durée $d_i \in \mathbb{N}$ et une utilisation des ressources $r_{k,i} \in \mathbb{N}$ pour toutes ressources $k \in R$. Chaque ressource $k \in R$ a une capacité $c_k \in \mathbb{N}$ décrivant la quantité de cette ressource disponible à chaque instant. Le but est de trouver un planning des tâches $T$ respectant les priorités dans $P$ et n'excédant pas, à chaque instant, la capacité des ressources disponibles. Généralement, on cherche également à optimiser la durée totale d'exécution du planning.

Un modèle par contraintes de ce problème consiste à représenter une tâche $i$ par sa date de début $s_i$. Les constantes et variables du problème sont toutes discrètes. Les contraintes temporelles sont exprimées comme suit :

$$\forall (i \ll j) \in P,\ s_i + d_i \leq s_j \tag{1}$$

Les contraintes de ressources sont exprimées comme suit :

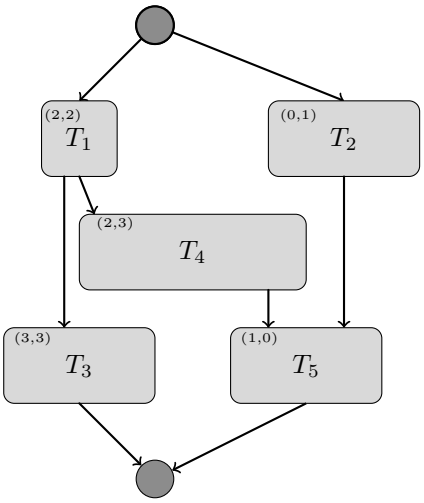
$$\forall t \in [0..h-1],\ \forall k \in R,\ (\sum_{i \in T, s_i \leq t < s_i + d_i} r_{k,i}) \leq c_k \tag{2}$$

où $h$ est l'horizon du problème représentant la date maximale de fin d'une tâche. Une façon simple d'obtenir l'horizon est de faire la somme de la durée des tâches. On vérifie que l'utilisation de chaque ressource $k$ n'excède pas sa capacité $c_k$ à chaque instant $t$. Une ressource est potentiellement utilisée dans un instant $t$ si une tâche utilisant cette ressource s'exécute à cet instant.

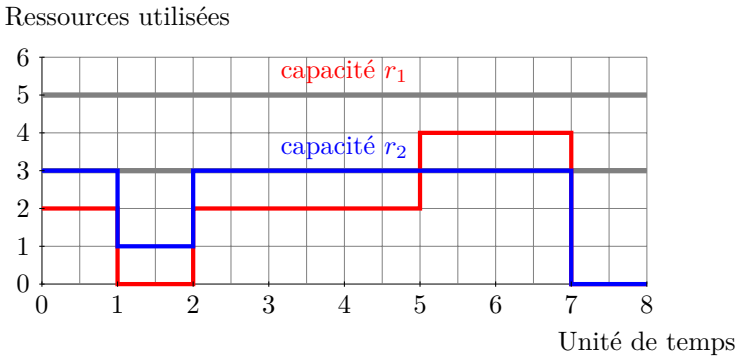
**Exemple 1.** Nous considérons un exemple de problème RCPSP sur la Figure 1. Ce problème comporte 5 tâches dont les priorités sont décrites sur la Figure 1a, et 2 ressources de capacité 5 et 3 dont l'usage est également donné sur la Figure 1a dans le coin supérieur gauche de chaque tâche. La durée totale minimum d'un planning pour cet exemple est de 7 unités de temps (Figure 1c). Le profil d'utilisation des ressources est représenté sur la Figure 1b : à chaque instant la somme des ressources utilisées n'excède pas leurs capacités. Par exemple, durant le deuxième instant seule la tâche $T_2$ est active et consomme 1 unité de la ressource $r_2$.

Dans la suite, nous considérons la version généralisée *RCPSP/max* où les priorités entre deux tâches sont généralisées. L'ensemble $P$ contient des contraintes temporelles de la forme $\pm s_i - \pm s_j \leq c$ où $i, j$ sont des tâches et $c \in \mathbb{Z}$ une constante entière. Ainsi, si nous avons $s_2 - s_1 \leq 3$, cela signifie que la tâche 2 doit commencer au plus tard 3 instants après la tâche 1. Les contraintes temporelles « au plus tard », « au plus tôt »,

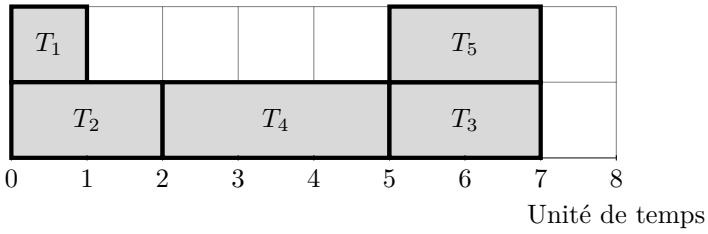
1. Le code et les *benchmarks* sont publiquement disponibles à l'adresse github.com/ptal/AbSolute.




(a) Contraintes temporelles et de ressources.




(b) Profil des ressources.




(c) Une solution minimisant la durée totale.

FIGURE 1 – Un exemple du problème RCPSP sur 5 tâches et 2 ressources de capacité 5 et 3.

« exactement », « après » et « avant » peuvent également être encodées. Cette forme de contraintes apparaît dans de nombreux travaux comme les réseaux temporels de contraintes [9], les octogones en interprétation abstraite [15], et la théorie de la logique différentielle (DL) dans les solveurs *Satisfiability Modulo Theories* (SMT) [21]. Dans cet article, nous nous intéressons aux octogones entiers.

## 3 Interprétation abstraite

L'interprétation abstraite est une technique d'analyse de programmes par sur-approximation de l'ensemble des valeurs concrètes que peuvent prendre les variables du programme [7]. Nous nous intéressons à un fragment particulier de cette théorie qui est les *domaines abstraits*. Ceux-ci encapsulent des conjonctions de contraintes de forme prédéfinie comme les contraintes de bornes $x \geq c$ ou $x \leq c$ (domaine des intervalles) ou les contraintes linéaires de la forme $a_1 * x_1 + \ldots + a_n * x_n \geq c$ où $a_i$ et $c$ sont des constantes (domaine des polyèdres).

### 3.1 Notations

On utilise l'ensemble $K = \{true, false, unknown\}$ pour représenter les éléments de la logique de Kleene (on notera par exemple que $false \wedge unknown = false$ et $true \wedge unknown = unknown$). Une contrainte est un prédicat logique défini sur un ensemble de variables, et nous notons $C$ l'ensemble de toutes les contraintes logiques. Nous appellerons *domaine concret* l'ensemble des solutions d'une conjonction de contraintes $\bigwedge c \in C$.

### 3.2 Domaine abstrait

En interprétation abstraite, un domaine abstrait est un ensemble partiellement ordonné muni de certaines opérations utiles à l'analyse de programme. Les domaines abstraits ont été adaptés au cadre de la programmation par contraintes, avec des opérateurs spécifiques, dans [19]. Dans cet article, nous nous plaçons dans ce cadre et introduisons un domaine entier intégrant des contraintes de différences, les octogones entiers. Nous ne donnons que les opérations nécessaires dans la suite de cet article, et notamment les correspondances de Galois ne sont pas explicitées.

**Définition 2** (Domaine abstrait)**.** Un domaine abs-

trait pour la programmation par contraintes est un treillis complet $\langle Abs, \leq \rangle$ dont les éléments sont représentables en machine, et muni des opérations suivantes :

- $\bot$ est le plus petit élément.
- $\sqcup$ est l'opérateur d'union (*join*) de deux éléments.
- $state : Abs \to K$ donne l'état d'un élément : *true* si l'élément satisfait toutes les contraintes du domaine abstrait, *false* si c'est un élément qui ne satisfait pas toutes les contraintes du domaine abstrait, et *unknown* si la satisfiabilité de cet élément ne peut pas encore être décidée.
- $[\![.]\!] : C \to Abs$ est une fonction partielle transformant une contrainte en un élément du domaine abstrait. Cette fonction n'est pas nécessairement définie pour toutes les contraintes puisqu'un domaine abstrait ne permet de gérer que certaines contraintes.
- $closure : Abs \to Abs$ est une fonction extensive ($x \leq closure(x)$) permettant d'éliminer des valeurs inconsistantes du domaine abstrait. Dans le jargon de la programmation par contraintes, il s'agit de la propagation.

Les *solutions* d'un élément abstrait $a \in Abs$ sont données par l'ensemble $sol(a) = \{s \in Abs \mid a \leq s \wedge state(s) = true\}$. Quand une confusion est possible, nous annotons les opérations du domaine abstrait avec le nom du domaine, par exemple $\bot_{Box}$ pour l'élément $\bot$ sur le domaine des intervalles $Box$. Finalement, notons que le domaine abstrait contient les contraintes que celui-ci peut propager, et l'opérateur *closure* peut être vu comme l'algorithme de propagation des contraintes à l'intérieur du domaine abstrait. Plus de détails concernant les autres opérateurs nécessaires sont disponibles dans [19]. Nous illustrons maintenant cette définition avec l'exemple du domaine abstrait des octogones entiers.

## 4 Octogones entiers

### 4.1 Définitions

Le domaine abstrait des octogones entiers est défini sur un ensemble de variables $(x_0, \ldots, x_{n-1})$ et une conjonction de contraintes octogonales de la forme suivante :

$$\pm x_i - \pm x_j \leq d$$

où $d \in \mathbb{Z}$ est une constante. En étendant le domaine à $2n$ variables $(x'_0, \ldots, x'_{2n-1})$, Miné [15] donne une transformation de ces contraintes octogonales en contraintes de potentiel de la forme $x'_i - x'_j \leq d$ (les variables ne sont que positives). Cette représentation est intéressante car un système de contraintes de potentiel peut être résolu en temps cubique par l'algorithme des plus

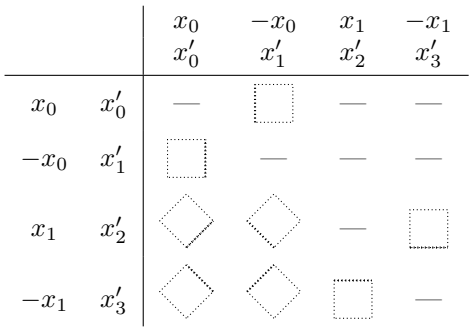



FIGURE 2 – Correspondance DBM/octogone en dimension 2.

courts chemins de Floyd-Warshall. On peut représenter ces contraintes dans une matrice à différences bornées (DBM) qui correspond à l'octogone initial [15]. Une DBM est une matrice $m$ où un élément $m_{i,j}$ représente la constante $d$ d'une contrainte de potentiel $x'_j - x'_i \leq d$. Nous noterons que lorsque $j/2 > i/2$ (où / est la division entière), l'élément représenté $m_{j,i}$ est redondant. Une matrice dont les éléments redondants sont égaux est appelée *cohérente* dans la littérature. Nous ne représenterons pas ces éléments dans les exemples [2] et gardons la matrice entière pour définir les opérations.

Afin de donner quelques intuitions, nous pouvons voir un octogone à $n$ dimensions comme une intersection de boîtes à $n$ dimensions également. Sur la Figure 2, on donne un octogone en 2 dimensions comme l'intersection de deux boîtes (dont une tournée à 45°) et les contraintes associées aux différents côtés de l'octogone. Les lignes et colonnes de la matrice sont annotées avec les variables correspondantes, et on schématise l'élément de la matrice avec la borne du côté qu'il représente. Notons qu'à partir d'une entrée $(i, j)$ de la DBM, nous pouvons récupérer les indices du côté opposé avec $(\bar{\imath}, \bar{\jmath})$ où :

$$\bar{\imath} = \begin{cases} i + 1 \text{ si } i \text{ est pair} \\ i - 1 \text{ si } i \text{ est impair} \end{cases}$$

et similairement pour $\bar{\jmath}$.

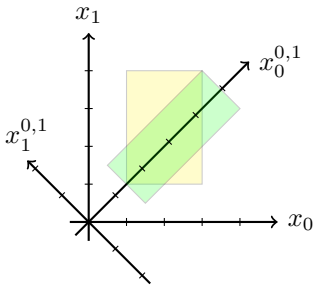
Nous pouvons passer d'un ensemble de contraintes octogonales à un ensemble de contraintes de potentiel par la réécriture suivante (avec $i \neq j$) :

$$\begin{array}{lcll}
x_i \geq d & \rightsquigarrow & x'_{2i+1} - x'_{2i} & \leq -2d \\
x_i \leq d & \rightsquigarrow & x'_{2i} - x'_{2i+1} & \leq 2d \\
x_i - x_j \leq d & \rightsquigarrow & x'_{2i} - x'_{2j} & \leq d \\
x_i + x_j \leq d & \rightsquigarrow & x'_{2i} - x'_{2j+1} & \leq d \\
-x_i - x_j \leq d & \rightsquigarrow & x'_{2i+1} - x'_{2j} & \leq d \\
-x_i + x_j \leq d & \rightsquigarrow & x'_{2i+1} - x'_{2j+1} & \leq d
\end{array}$$

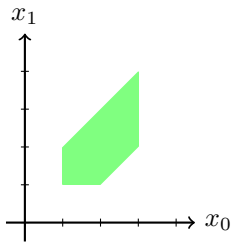
2. C'est la représentation utilisée dans les implémentations [15].

| | $x'_0$ | $x'_1$ | $x'_2$ | $x'_3$ |
|---|---|---|---|---|
| $x'_0$ | 0 | -2 | | |
| $x'_1$ | 6 | 0 | | |
| $x'_2$ | 1 | -2 | 0 | -2 |
| $x'_3$ | 7 | 1 | 8 | 0 |

(a) DBM




(b) Octogone par intersection de deux boîtes




(c) Octogone

FIGURE 3 – DBM et sa représentation par intersection de deux boîtes.

**Exemple 3.** Afin d'illustrer la correspondance entre les contraintes octogonales, la DBM et sa représentation géométrique, on considère les contraintes suivantes :

$$\begin{array}{ll} x_0 \geq 1 \wedge x_0 \leq 3 & x_1 \geq 1 \wedge x_1 \leq 4 \\ x_0 - x_1 \leq 1 & -x_0 + x_1 \leq 1 \end{array}$$

Les contraintes de borne sur $x_0$ et $x_1$ sont représentées par la boîte en jaune sur la Figure 3b, et les contraintes octogonales par la boîte en vert. L'intersection des deux boîtes est donnée sur la Figure 3c.

La contrainte de potentiel associée à $-x_0+x_1 \leq 1$ est $x'_1 - x'_3 \leq 1$, représentée dans la DBM (Figure 3a) par l'entrée $dbm_{3,1} = 1$. Nous pouvons facilement retrouver le côté de l'octogone représenté à la position $(3, 1)$ grâce à la Figure 2.

## 4.2 Opérations

Nous prenons la représentation matricielle d'un octogone pour définir les opérations du domaine abstrait. Pour définir ces opérations, nous considérons deux DBMs $m$ et $m'$ de dimension $n$, et $N = [0..2n-1]$. On voit une matrice $m$ comme un ensemble d'éléments indexés $\{d^{i,j} \mid i, j \in N\}$.

Premièrement, la plus petite matrice $\perp$ est définie par $\{\infty^{i,j} \mid i, j \in N\}$.

En second, nous avons l'opérateur $\sqcup$ défini comme suit :

$$m \sqcup m' = \{min(m_{i,j}, m'_{i,j})^{i,j} \mid i, j \in N\} \qquad (3)$$

L'intersection géométrique de deux octogones se fait en prenant le minimum des coefficients des deux DBMs. En effet, si un coefficient est plus petit, cela signifie que l'intervalle entre le côté qu'il représente et le côté opposé est plus étroit.

Ensuite, nous avons l'opération *state* :

$$state(m) = \begin{cases} \textit{true} & \text{si } \forall i, j \in N,\ m_{i,j} = m_{\bar{\imath},\bar{\jmath}} \\ \textit{false} & \text{si } \exists i, j \in N,\ m_{i,j} + m_{\bar{\imath},\bar{\jmath}} < 0 \\ \textit{unknown} & \text{sinon} \end{cases} \qquad (4)$$

Un octogone est dans l'état *true* si toutes ses variables sont des singletons, ce qui signifie que la DBM représente un point dans l'espace, et chaque côté est identique à son côté opposé. Un octogone est inconsistant (*false*) si un de ses côtés inférieurs a dépassé le côté supérieur correspondant, ce qui se traduit par la somme des coefficients inférieure à zéro [3]. Dans tous les autres cas, l'octogone est dans un état indéterminé.

L'opération de transfert dans les octogones n'est définie que pour les contraintes octogonales. Considérons $a, b$ les indices des variables octogonales et $i, j$ des variables de potentiel, on a la réécriture suivante :

$$\pm x_a - \pm x_b \leq d \rightsquigarrow x'_i - x'_j \leq d'$$

et donc le transfert suivant :

$$[\![\pm x_a - \pm x_b \leq d]\!] = \perp \sqcup \{d'_{i,j}\}$$

On utilise la transformation d'une contrainte octogonale en contrainte de potentiel et affectons le coefficient obtenu $d'$ dans la case correspondante de la matrice. Finalement l'opérateur de clôture est obtenu grâce à l'algorithme de Floyd-Warshall. Nous ne redonnons pas l'algorithme de clôture ici qui est étudié extensivement dans [15, 1, 4]. Il est particulièrement intéressant de considérer la clôture incrémentale qui permet de mettre à jour la DBM avec une contrainte octogonale en $O(n^2)$ [15, 4] au lieu de la clôture générale qui est en $O(n^3)$.

3. Notons que le signe du coefficient d'un côté inférieur doit être inversé pour obtenir sa coordonnée dans le plan Euclidien.

# 5 Produit réduit via contraintes réifiées

## 5.1 Décomposition du problème RCPSP/max

Les contraintes temporelles du problème RCPSP/max sont des contraintes octogonales qui peuvent être gérées efficacement par les octogones entiers. L'étape non triviale est de gérer les contraintes de ressources qui ne sont pas purement octogonales (sinon la résolution du problème serait polynomiale).

En programmation par contraintes, les contraintes de ressources sont généralement représentées par la contrainte globale `cumulative` :

$$\texttt{cumulative}_k([s_1, \ldots, s_n], [d_1, \ldots, d_n], [r_{k,1}, \ldots, r_{k,n}], c_k)$$

qui assure que les tâches n'excèdent pas la capacité $c_k$ de la ressource $k$. Notons que cette contrainte ne gère qu'une seule ressource $k \in R$ à la fois. Afin d'extraire des contraintes octogonales de cumulative, nous considérons sa *décomposition par tâches* telle que donnée dans [22] :

$$\begin{aligned} &\forall j \in [1..n],\ \forall i \in [1..n] \setminus \{j\}, \\ &\quad b_{i,j} \Leftrightarrow (s_i \leq s_j \wedge s_j < s_i + d_i) \end{aligned} \tag{5}$$

$$\forall j \in [1..n],\ r_{k,j} + (\sum_{i \in [1..n] \setminus \{j\}} r_{k,i} * b_{i,j}) \leq c_k \tag{6}$$

La décomposition (5) introduit $n^2 - n$ variables booléennes, où une variable $b_{i,j}$ est vraie ssi les tâches $i$ et $j$ se chevauchent. La contrainte de chevauchement $s_i \leq s_j \wedge s_j < s_i + d_i$ est réifiée dans la variable booléenne $b_{i,j}$. Pour ce qui est de la décomposition (6), l'intuition principale est que l'utilisation maximale d'une ressource $k$ ne peut changer que lorsque une tâche commence, et pas pendant son exécution. Ainsi, pour toutes les dates de début $s_j$, la somme de $r_{k,j}$ et des ressources utilisées par les tâches $i$ s'exécutant à l'instant $s_j$ ne doit pas excéder la capacité $c_k$.

## 5.2 Opération de déduction

Les contraintes temporelles du RCPSP (en (1)) peuvent être propagées dans le domaine abstrait des octogones, tandis que les contraintes de ressources (en (6)) peuvent être propagées dans le domaine abstrait des intervalles. Il reste à faire communiquer les deux domaines et les contraintes réifiées remplissent ce rôle. Une contrainte réifiée est une contrainte de la forme $c_1 \Leftrightarrow c_2$ permettant de poser des équivalences entre contraintes. La force de la méthode proposée ici est que les contraintes $c_1$ et $c_2$ ne sont pas forcément représentées dans le même domaine abstrait. Ce mécanisme est donc particulièrement adapté pour échanger de l'information entre deux domaines abstraits.

Nous avons des contraintes réifiées en (5) où les variables booléennes $b_{i,j}$ sont représentées dans le domaine abstrait des intervalles et les contraintes $s_i \leq s_j \wedge s_j < s_i + d_i$ dans le domaine des octogones. Afin de gérer cet échange d'information, nous devons munir nos domaines abstraits de l'opération de déduction *entailment* permettant de détecter si une contrainte est déjà induite par le problème.

**Définition 4** (Déduction)**.** Soit un domaine abstrait *Abs* et deux éléments $a, b \in Abs$, l'opération $entailment : Abs \times Abs \to K$ est un opérateur de déduction si il satisfait les propriétés suivantes :

1. Si $entailment(a, b) = true$ (noté $a \vDash b$) alors $sol(a) = sol(a \sqcup b)$ ; on n'obtient pas moins de solutions avec l'élément $b$, il est redondant dans $a$.
2. Si $entailment(a, b) = false$ (noté $a \nvDash b$) alors $sol(a \sqcup b) = \emptyset$ ; on ne pourra jamais déduire $b$ de $a$ dans une solution.
3. Si $entailment(a, b) = unknown$ alors $state(a) = unknown \vee state(b) = unknown$ ; on répond *unknown* si un des éléments $a$ ou $b$ est lui-même *unknown*.

L'opérateur de déduction peut être beaucoup plus facile à calculer qu'établir la satisfiabilité du problème puisqu'on peut répondre *unknown*. Une version générique de cette opération utilisant les opérations $\sqcup$ et *state* d'un domaine abstrait peut-être définie comme suit :

$$entailment(a, b) = \begin{cases} true & \text{si } (a \sqcup b) = a \\ false & \text{si } state(a \sqcup b) = false \\ unknown & \text{sinon} \end{cases}$$

**Proposition 5.** Soit les DBMs $m$ et $m'$, $entailment(m, m')$ est un opérateur de déduction (Définition 4) pour le domaine abstrait des octogones.

*Démonstration.* On procède par cas :

— *true* : Si on a $m \sqcup m' = m$ alors par la Définition 3, toutes les entrées de $m$ sont plus petites que celles de $m'$. La DBM $m'$ contient donc déjà toutes les solutions présentes dans $m$, et ne permet pas de réduire cet ensemble.

— *false* : Si on a $state(m \sqcup m') = false$ alors par la Définition 4, il existe une dimension nulle dans $m \sqcup m'$, et donc l'octogone ne contient pas de solution.

— *unknown* : Si $m$ ou $m'$ est inconsistant ($state(m) = false \vee state(m') = false$), alors leur union donne également une DBM inconsistante. Si $m$ et $m'$ sont consistants ($state(m) =$

$true \wedge state(m') = true$) alors leur union ne peut donner qu'une DBM consistante ou inconsistante, vu que toutes les bornes inférieures sont égales aux bornes supérieures, soit $m_{i,j} = m_{\bar{i},\bar{j}}$. Par conséquent, *unknown* n'est possible que lorsqu'une des deux DBMs est *unknown*.

□

Dans le cas des octogones, nous pourrions également fournir une version plus précise de l'opérateur de déduction en utilisant $closure(a \sqcup b)$, mais cette version est plus coûteuse vu qu'elle est de complexité $O(n^3)$—la première version étant en $O(n^2)$. Finalement, notons que dans le cas du RCPSP, nous appliquons l'opérateur de déduction entre une DBM et une seule contrainte octogonale. La complexité de la première version est ainsi en temps constant, et celle de la deuxième en $O(n^2)$ avec la clôture incrémentale.

### 5.3 Produit réduit des intervalles et octogones

Un nouveau domaine abstrait $Box \times Oct$ issu de la combinaison du domaine des intervalles $Box$ et des octogones $Oct$ peut être obtenu par produit direct. Le produit direct est un produit cartésien de deux domaines abstraits avec les opérateurs redéfinis sur ce produit [16]. Nous pouvons le définir sur $Box \times Oct$ de la manière suivante :

$$\bot = (\bot_{Box}, \bot_{Oct})$$

$$(b, o) \sqcup (b', o') = (box \sqcup_{Box} b', o \sqcup_{Oct} o')$$

$$state((b, o)) = state_{Box}(b) \wedge state_{Oct}(o)$$

$$[\![c]\!] = \begin{cases} ([\![c]\!]_{Box}, [\![c]\!]_{Oct}) & \\ ([\![c]\!]_{Box}, \bot_{Oct}) & \text{si } [\![c]\!]_{Oct} \text{ n'est pas défini} \\ (\bot_{Box}, [\![c]\!]_{Oct}) & \text{si } [\![c]\!]_{Box} \text{ n'est pas défini} \end{cases}$$

$$closure((b, o)) = (closure(b), closure(o))$$

Le problème du produit direct est que les domaines abstraits n'échangent pas d'information. Le *produit réduit* augmente le produit direct avec un échange d'information [16]. Bien sûr, il existe une multitude de produits réduits entre deux domaines abstraits. Nous proposons un produit réduit entre les intervalles et octogones où les deux communiquent exclusivement via des contraintes réifiées. Nous ajoutons au produit direct un ensemble $R$ de contraintes réifiées, et redéfinissons l'opérateur de clôture. Pour ce faire, nous introduisons d'abord un propagateur pour les contraintes réifiées.

À l'aide de l'opérateur de déduction nous pouvons donner un propagateur générique pour les contraintes d'équivalence entre deux domaines abstraits. Bien que ce propagateur soit générique, nous utilisons le domaine des intervalles $Box$ et octogones $Oct$ afin d'illustrer son fonctionnement. Soit $c_1 \Leftrightarrow c_2$ une contrainte d'équivalence où $[\![c_1]\!]_{Box}$ et $[\![c_2]\!]_{Oct}$ sont définis, nous avons :

$$prop_{\Leftrightarrow}(b, o, c_1 \Leftrightarrow c_2) := \begin{cases} b \vDash [\![c_1]\!]_{Box} \implies (b, o \sqcup [\![c_2]\!]_{Oct}) \\ b \nvDash [\![c_1]\!]_{Box} \implies (b, o \sqcup [\![\neg c_2]\!]_{Oct}) \\ o \vDash [\![c_2]\!]_{Oct} \implies (b \sqcup [\![c_1]\!]_{Box}, o) \\ o \nvDash [\![c_2]\!]_{Oct} \implies (b \sqcup [\![\neg c_1]\!]_{Box}, o) \end{cases}$$

Cette fonction peut être généralisée à des contraintes d'équivalence de la forme $c_1 \wedge \ldots \wedge c_n \Leftrightarrow c'_1 \wedge \ldots \wedge c'_m$ en étendant l'opération de déduction sur des conjonctions.

Nous pouvons maintenant définir l'opérateur de clôture du produit réduit $Box \times Oct$ :

$$closure_R(box, oct, R) = (\textstyle\bigsqcup_{r \in R} prop_{\Leftrightarrow}(box, oct, r), R)$$

$$closure((box, oct, R)) = (closure_R(closure(box'), closure(oct'), R))$$

Soit $e$ un élément du produit réduit, l'opérateur de clôture peut être appliqué jusqu'à réaliser le point fixe de $closure(e) = e$.

## 6 Implémentation et expérimentations

Les fonctions introduites dans les sections précédentes nous permettent d'intégrer le domaine abstrait $Box \times Oct$ dans le solveur abstrait `AbSolute` [19] que nous améliorons pour gérer le cas discret. Nous fournissons trois domaines abstraits : les intervalles, les octogones et le produit réduit de ces deux domaines. L'implémentation et les *benchmarks* présentés dans cette section sont publiquement disponibles à l'adresse github.com/ptal/AbSolute. Les mesures sont réalisées sur un Dell XPS 13 9370, avec un processeur Intel(R) Core(TM) i7-8550U cadencé à 1.8GHz sous GNU Linux. Le solveur `AbSolute` est compilé avec la version `4.07.1+flambda` du compilateur OCaml.

### 6.1 Implémentation fonctionnelle

Une partie de la complexité d'implémentation d'un solveur de contraintes vient des structures de données qui doivent être *backtrackable*. Par conséquent, à chaque fois qu'une variable doit être restaurée lors d'un *backtrack*, celle-ci doit être manipulée au travers d'une mémoire dédiée qui est gérée par le solveur. Cette mémoire apparait de part et autre dans le code du solveur, complexifiant les interfaces et la programmation, alors qu'il devrait s'agir d'un aspect transversal du solveur.

Le paradigme fonctionnel fournit par défaut une mémoire non-mutable qui peut être utilisée dans des algorithmes de *backtrack* sans effort particulier. De plus, le langage `OCaml`, dans lequel est programmé

| | faisable (%) | optimal (%) | insatisfiable (%) |
|---|---|---|---|
| **J30** | 100 | 100 | 0 |
| `AbSolute` | 100 | 47.71 | 0 |
| `GeCode` | 100 | 72.29 | 0 |
| `Chuffed` | 100 | **99.38** | 0 |
| **UBO100** | 86.87 | 86.87 | 13.13 |
| `AbSolute` | 50 | 12.22 | 11.11 |
| `GeCode` | 64.44 | 32.22 | **12.22** |
| `Chuffed` | **85.55** | **74.44** | 8.89 |

FIGURE 4 – Résultats sur les instances J30 et UBO100.

notre solveur `AbSolute`, propose des traits impératifs classiques. Cela permet à tout moment de choisir si une variable est globale à l'arbre d'exploration (via une référence mutable) ou locale à une branche de l'arbre (via une structure fonctionnelle). L'avantage est que la mémoire *backtrackable* est gérée par le langage lui-même au lieu du solveur `AbSolute`.

On peut comparer la stratégie de restauration offerte par OCaml à une stratégie de *trailing* où seules les modifications sur une structure sont prises en compte, au lieu de réaliser une copie [5]. D'autre part, nous utilisons les tableaux persistants de Conchon et Filliatre [6] afin d'implémenter efficacement la DBM. La technique décrite dans leur article est très proche d'un *trailing* à gros grain, donc sans compression des modifications successives sur une même case mémoire lors de l'étape de propagation (technique appelée *time-stamping*). Nos mesures indiquent néanmoins que la compression ne réduit que de 10% la taille de la structure dans le cas général, et au maximum de 50% sur certaines instances. Dans l'implémentation, nous avons fait le choix de ne pas compresser les modifications.

### 6.2 Expérimentations préliminaires

On considère deux ensembles d'instances générés par l'outil ProGen/max [13] :

— `J30` pour le problème RCPSP contenant 480 instances de 30 activités [12].
— `UBO100` pour le problème RCPSP/max contenant 90 instances de 100 activités et 5 ressources [11].

Le temps maximal pour résoudre une instance est fixé à 60 secondes.

On teste et compare trois solveurs :

— `AbSolute` avec le produit réduit décrit en Section 5 et un algorithme de *branch and bound* classique.
— `GeCode 6.0.1` avec un modèle du RCPSP utilisant la contrainte globale `cumulative` et un algorithme de *timetabling* [24].
— `chuffed 0.10.0` avec un modèle du RCPSP sans contraintes cumulatives, avec la décomposition par tâches (Section 5.1). Ce solveur utilise la technique de génération de clauses paresseuses (*lazy clause generation*) [18] qui est la méthode la plus efficace pour résoudre le RCPSP et RCPSP/max à ce jour.

La stratégie d'exploration est basée sur les temps de début des tâches : on sélectionne la variable qui a la plus petite borne inférieure, et on affecte cette variable à cette borne.

Des résultats préliminaires sont disponibles sur la Figure 4 où on donne en pourcentage la proportion de problèmes faisables (au moins une solution est trouvée), optimal (preuve de la meilleure solution) et insatisfiables (preuve d'insatisfiabilité). La première ligne de chaque ensemble d'instances (commençant par `J30` et `UBO100`) contient la répartition des instances faisables et insatisfiables. On constate que le solveur `Chuffed` est largement le meilleur, sauf pour les preuves d'insatisfiabilité de `UBO100` où `AbSolute` parvient à en faire 2 de plus dans le temps imparti. On remarque que la différence entre `AbSolute` et `GeCode` est moindre mais reste quand même conséquente.

Il faudra conduire d'autres expérimentations pour tirer des conclusions plus précises sur les faiblesses et forces de ces différentes approches.

## 7 Conclusion

Les contraintes globales sont nécessaires à l'efficacité de la programmation par contraintes mais sont trop nombreuses. Une solution intermédiaire entre efficacité et expressivité est d'utiliser des domaines abstraits capturant des sous-théories réutilisables pour une large gamme de problèmes. Dans ce but, nous avons présenté un produit réduit de deux domaines abstraits où l'échange d'information se fait via des contraintes réifiées. En particulier, nous avons formalisé et expérimenté cette idée sur le produit réduit des intervalles et octogones afin de résoudre le problème RCPSP. Les expérimentations montrent que l'approche est suffisam-

ment efficace pour de nombreuses instances de tailles variées, mais reste en deçà des algorithmes de l'état de l'art comme la génération de clauses paresseuses.

Les perspectives futures sont variées, tant au niveau de la performance brute du domaine abstrait que des problèmes pouvant être résolus avec cette méthode. Premièrement, il est possible d'optimiser les octogones afin de cibler des instances de grandes tailles (au delà de 200 tâches), notamment par décomposition en sous-octogones [23]. Deuxièmement, de nombreuses extensions du RCPSP existent [17] et cachent certainement des contraintes propices à l'utilisation de domaines abstraits. Finalement, d'autres contraintes globales peuvent être décomposées vers des contraintes octogonales comme la contrainte `sequence` [14]. L'avantage étant qu'on pourrait supporter ces contraintes efficacement sans effort puisque le domaine des octogones existe déjà.

## Références

[1] Roberto Bagnara, Patricia M. Hill et Enea Zaffanella : Weakly-relational shapes for numeric abstractions : Improved algorithms and proofs of correctness. *Formal Methods in System Design*, 35(3):279–323, 2009.

[2] Nicolas Beldiceanu, Mats Carlsson et Jean-Xavier Rampon : Global Constraint Catalog, 2nd Edition (revision a), février 2011. SICS research report T2012-03, http ://soda.swedish-ict.se/5195/.

[3] Christian Bessiere, Stéphane Cardon, Romuald Debruyne et Christophe Lecoutre : Efficient algorithms for singleton arc consistency. *Constraints*, 16(1):25–53, janvier 2011.

[4] Aziem Chawdhary, Ed Robbins et Andy King : Incrementally closing octagons. *Formal Methods in System Design*, janvier 2018.

[5] Chiu Wo Choi, Martin Henz et Ka Boon Ng : Components for state restoration in tree search. *In Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming*, CP '01, pages 240–255, London, UK, UK, 2001. Springer-Verlag.

[6] Sylvain Conchon et Jean-Christophe Filliâtre : Semi-persistent data structures. *In European Symposium on Programming*, pages 322–336. Springer, 2008.

[7] Patrick Cousot et Radhia Cousot : Abstract interpretation : a unified lattice model for static analysis of programs by construction or approximation of fixpoints. *In Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 238–252. ACM, 1977.

[8] Patrick Cousot, Radhia Cousot et Laurent Mauborgne : Theories, solvers and static analysis by abstract interpretation. *Journal of the ACM (JACM)*, 59(6):31, 2012.

[9] Rina Dechter, Itay Meiri et Judea Pearl : Temporal constraint networks. *Artificial intelligence*, 49(1-3):61–95, 1991.

[10] Thibaut Feydy, Andreas Schutt et Peter J. Stuckey : Global difference constraint propagation for finite domain solvers. *In Proceedings of the 10th international ACM SIGPLAN conference on Principles and practice of declarative programming*, pages 226–235. ACM, 2008.

[11] Birger Franck, Klaus Neumann et Christoph Schwindt : Truncated branch-and-bound, schedule-construction, and schedule-improvement procedures for resource-constrained project scheduling. *OR-Spektrum*, 23(3):297–324, 2001.

[12] Rainer Kolisch et Arno Sprecher : PSPLIB-a project scheduling problem library. *European journal of operational research*, 96(1):205–216, 1997.

[13] Rainer Kolisch, Arno Sprecher et Andreas Drexl : Characterization and generation of a general class of resource-constrained project scheduling problems. *Management science*, 41(10):1693–1703, 1995.

[14] Michael Maher, Nina Narodytska, Claude-Guy Quimper et Toby Walsh : Flow-based propagators for the SEQUENCE and related global constraints. *In International Conference on Principles and Practice of Constraint Programming*, pages 159–174. Springer, 2008.

[15] A. Miné : The octagon abstract domain. *Higher-Order and Symbolic Computation (HOSC)*, 19(1): 31–100, 2006.

[16] A. Miné : Tutorial on static inference of numeric invariants by abstract interpretation. *Foundations and Trends in Programming Languages (FnTPL)*, 4(3–4):120–372, 2017.

[17] Ernesto Nunes, Marie Manner, Hakim Mitiche et Maria Gini : A taxonomy for task allocation problems with temporal and ordering constraints. *Robotics and Autonomous Systems*, 90:55–70, avril 2017.

[18] Olga Ohrimenko, Peter J. Stuckey et Michael Codish : Propagation via lazy clause generation. *Constraints*, 14(3):357–391, septembre 2009.

[19] Marie Pelleau, Antoine Miné, Charlotte Truchet et Frédéric Benhamou : A constraint solver

based on abstract domains. *In Verification, Model Checking, and Abstract Interpretation*, pages 434–454. Springer, 2013.

[20] Marie Pelleau, Charlotte Truchet et Frédéric Benhamou : The octagon abstract domain for continuous constraints. *Constraints*, 19(3):309–337, 2014.

[21] Vaughan Pratt : Two easy theories whose combination is hard. Rapport technique, Technical report, Massachusetts Institute of Technology, 1977.

[22] Andreas Schutt, Thibaut Feydy, Peter J. Stuckey et Mark G. Wallace : Why cumulative decomposition is not as bad as it sounds. *In International Conference on Principles and Practice of Constraint Programming*, pages 746–761. Springer, 2009.

[23] Gagandeep Singh, Markus Püschel et Martin Vechev : Making numerical program analysis fast. pages 303–313. ACM Press, 2015.

[24] Petr Vilím : *Global constraints in scheduling*. Thèse de doctorat, Charles University in Prague, 2007.

# De la pertinence des littéraux dans les contraintes pseudo-booléennes apprises

**Daniel Le Berre**[1,2] **Pierre Marquis**[1,2,3] **Stefan Mengel**[1] **Romain Wallon**[1,2]
[1] CRIL-CNRS UMR 8188, Lens, France
[2] Université d'Artois
[3] Institut Universitaire de France
{leberre,marquis,mengel,wallon}@cril.fr

**Résumé**

L'apprentissage de contraintes pseudo-booléennes (PB) dans les solveurs PB via le système de preuve des plans-coupes n'est pas aussi bien compris que celui des clauses dans les solveurs SAT utilisant l'analyse de conflit. En particulier, les approches basées sur la résolution généralisée peuvent déduire des contraintes PB comportant des littéraux non pertinents, i.e. des littéraux dont la valeur de vérité (quelle qu'elle soit) ne change jamais la valeur de vérité de la contrainte. Cela peut se produire même quand la formule PB considérée en entrée ne comporte pas de tels littéraux. Nous observons expérimentalement de telles situations sur des instances de l'évaluation pseudo-booléenne 2016. Nous montrons que la détection des littéraux non pertinents dans les contraintes PB apprises, bien que calculatoirement difficile, peut avoir un impact positif sur le temps d'exécution des solveurs (dans notre cas, Sat4j), en permettant le retrait de ces littéraux.

**Abstract**

Learning pseudo-Boolean (PB) constraints in PB solvers using the cutting planes proof system is a problem that is not as well understood as clause learning in conflict driven clause learning solvers. In particular, it turns out that approaches based on generalized resolution may derive PB constraints containing irrelevant literals, i.e. literals whose assigned truth values (whatever they are) never change the truth value of the constraint. This may happen even when starting from inputs without irrelevant literals. We observe such situations experimentally on benchmarks from the 2016 PB evaluation. We show that the detection and removal of irrelevant literals in learned PB constraints, though computationally expensive, can have a positive impact on the runtime of solvers (in our case, Sat4j).

## 1 Introduction

En dépit de l'efficacité relative des solveurs SAT modernes dans la résolution de nombreux problèmes industriels, il subsiste des instances qui restent difficiles à résoudre, même pour les solveurs de l'état de l'art. C'est en particulier le cas des formules incohérentes pour lesquelles la preuve d'incohérence nécessite un nombre exponentiel d'étapes de résolution, comme par exemple celles représentant le *principe du pigeonnier* [13]. Parmi ces formules difficiles, nombreuses sont celles requérant la capacité de détecter des symétries et de « compter » pour pouvoir générer des preuves courtes. De ce constat est né l'engouement pour le raisonnement pseudo-booléen [20], qui bénéficie à la fois de l'expressivité des *contraintes pseudo-booléennes* (des équations et inéquations linéaires en variables booléennes) et du pouvoir d'inférence du système de preuve des *plans-coupes* [3, 12, 14, 19]. Ce système de preuve est en théorie strictement plus puissant que le système de preuve à base de résolution utilisé dans les solveurs SAT. Cependant, en pratique, aucun des solveurs PB existants n'utilise pleinement le pouvoir du système de preuve des plans-coupes [22]. En effet, la plupart des solveurs actuels sont fondés sur une forme spécifique de ce système, pouvant être vue comme une généralisation de la résolution [14]. Celle-ci permet d'étendre l'inférence clausale à l'inférence pseudo-booléenne, et ainsi d'hériter de nombreuses techniques utilisées dans la résolution du problème SAT [2, 6, 8]. En particulier, l'apprentissage de clauses guidé par les conflits (*conflict-driven clause learning*) à la base de l'architecture des solveurs SAT modernes [9, 17, 18] peut être généralisé aux solveurs PB, permettant ainsi d'*apprendre* des contraintes pour effectuer des retours-arrière non chronologiques.

Lorsqu'un conflit survient durant la recherche (i.e. lorsqu'une contrainte pseudo-booléenne est falsifiée), la règle de résolution généralisée [14] est appliquée entre la contrainte conflictuelle et la raison de la propagation de l'un de ses littéraux pour produire une nouvelle contrainte. Il est important de noter que, contrairement aux solveurs utilisant la résolution, la raison doit parfois être *affaiblie* au préalable pour que la contrainte inférée soit toujours conflictuelle. Cette opération est répétée jusqu'à ce que la contrainte déduite permette de propager certains de ses littéraux. La contrainte est ensuite apprise, avant d'effectuer un retour-arrière. Au fil des ans, de nombreux solveurs pseudo-booléens implantant des variantes du système des plans-coupes ont été développés [2, 7, 15, 21]. Cependant, depuis la première évaluation pseudo-booléenne [16], le constat est décevant : ces solveurs ne parviennent pas à résoudre une gamme de problèmes aussi large que les solveurs utilisant la résolution, qui encodent les contraintes sous forme de clauses. En particulier, les bonnes performances des solveurs PB sur certaines catégories spécifiques d'instances ne se généralisent pas à toutes les catégories [10]. Ceci est en partie dû au fait que les règles du système des plans-coupes sont difficiles à implanter efficacement, et que choisir lesquelles desdites règles il faut appliquer est loin d'être trivial. En premier lieu, l'approche considérée a été de remplacer l'application des règles de la résolution par celles de la résolution généralisée pendant l'analyse de conflit. Cependant, une telle approche n'est pas satisfaisante, car elle est équivalente à la résolution lorsqu'elle est appliquée à des clauses et nécessite un traitement spécifique pour déduire des contraintes de cardinalité [1]. Plus récemment, une amélioration a été proposée et implantée dans le solveur *RoundingSat* [11]. Elle consiste en une utilisation agressive de la règle de division fournie par le système des plans-coupes (mais pas par la résolution généralisée).

Dans cet article, nous considérons un nouveau point de vue sur les propriétés des systèmes de preuve introduits plus haut, et, en particulier, sur l'une de leurs faiblesses. En effet, il semble naturel de penser que les contraintes apprises au cours de l'analyse de conflit contiennent uniquement des *littéraux pertinents*, i.e. des littéraux dont l'affectation a une influence sur la valeur de vérité de la contrainte. Tant que les contraintes inférées sont des clauses ou des contraintes de cardinalité, la question ne se pose pas : les littéraux jouant un rôle symétrique dans ces contraintes, chacun d'eux est pertinent sauf si la contrainte est valide. Cela peut cependant ne pas être le cas pour des contraintes pseudo-booléennes. Alors que, le plus souvent, les instances pseudo-booléennes ne contiennent que des littéraux pertinents, la résolution généralisée peut produire des contraintes comportant des littéraux qui ne le sont pas. Nous observons que leur présence peut nuire à l'efficacité des solveurs PB. D'un côté, de tels littéraux rendent la contrainte plus complexe que nécessaire, à la fois en termes de nombre de littéraux et de taille de coefficients. De l'autre, ils rendent caduque le critère syntaxique permettant la détection de clauses et de contraintes de cardinalité, pour lesquelles les solveurs disposent de structures de données plus efficaces. Pire, la présence de littéraux non-pertinents peut conduire à l'inférence de contraintes plus faibles que celles qui auraient pu être déduites. Ce constat s'applique tout particulièrement lorsque des arrondis sont appliqués aux contraintes. Nous montrons que ces opérations peuvent non seulement donner plus d'importance aux littéraux non pertinents qu'ils n'en ont initialement, mais peuvent même les rendre pertinents, réduisant ainsi le pouvoir de propagation de la contrainte apprise. Ces observations conduisent naturellement à l'idée de détecter les littéraux non pertinents, afin de pouvoir les supprimer. Malheureusement, vérifier si un littéral est pertinent dans une contrainte pseudo-booléenne est NP-complet [5, Section 9.6]. D'un point de vue pratique, il semble donc irréaliste d'effectuer cette opération sur toutes les contraintes produites. Cependant, comme nous le montrons plus loin, les littéraux non pertinents présentent des propriétés intéressantes qui peuvent être exploitées pour réduire le nombre de tests à réaliser sur une contrainte pour les identifier. Grâce à cela, nous avons pu implanter diverses approches permettant de détecter une large majorité des littéraux non pertinents dans les contraintes apprises. Nos expérimentations montrent que ces approches, combinées à des bornes adéquates sur les contraintes à traiter, permettent de détecter et de supprimer ces littéraux des contraintes apprises, et peuvent même améliorer significativement le temps d'exécution des solveurs.

La suite de cet article est organisée comme suit. Dans un premier temps, nous introduisons quelques préliminaires relatifs au raisonnement pseudo-booléen. Puis, nous discutons de l'existence de littéraux non pertinents dans les contraintes pseudo-booléennes, et expliquons dans quelle mesure ils peuvent affecter les performances des solveurs. Nous présentons ensuite notre algorithme pour identifier et supprimer ces littéraux, avant de l'évaluer expérimentalement. Enfin, nous concluons en présentant quelques perspectives de futurs travaux.

## 2 Préliminaires

Nous considérons un cadre propositionnel défini sur un ensemble fini de variables propositionnelles $V$ interprété classiquement. Un *littéral* $l$ est une variable

propositionnelle $x \in V$ ou sa négation $\bar{x}$. Notons que, dans ce contexte, les valeurs booléennes sont représentées par les entiers 1 (vrai) et 0 (faux) et que $\bar{x} = 1 - x$. L'implication logique est symbolisée par $\models$ et l'équivalence logique par $\equiv$.

### 2.1 Définitions

**Contraintes pseudo-booléennes.** Une *contrainte pseudo-booléenne (PB)* est une contrainte de la forme $\sum_{i=1}^{n} a_i l_i \triangle d$, où les $a_i$ et $d$ sont des entiers, les $l_i$ des littéraux et $\triangle \in \{\leq, <, =, >, \geq\}$. Chaque $a_i$ est appelé *poids* ou *coefficient* et $d$ est appelé *degré* de la contrainte. Toute contrainte PB peut être *normalisée* en une (conjonction de) contraintes de la forme $\sum_{i=1}^{n} a_i l_i \geq d$ dans laquelle les coefficients et le degré sont positifs. C'est pourquoi, dans la suite, nous supposons que toutes les contraintes PB sont normalisées. Une *contrainte de cardinalité* est une contrainte PB dont tous les coefficients valent 1, et une *clause* est une contrainte de cardinalité de degré 1. Cette définition coïncide avec la définition usuelle d'une clause comme une disjonction de littéraux, et illustre le fait que les contraintes de cardinalité et pseudo-booléennes généralisent les clauses.

**Conditionnement.** Étant données une contrainte PB $\chi$ et une conjonction de littéraux $\tau$, $\chi|\tau$ représente le *conditionnement* de $\chi$ par $\tau$, qui est équivalent à la contrainte $\chi$ où chaque littéral est remplacé par 1 s'il apparaît dans $\tau$, ou par 0 si son opposé apparaît dans $\tau$. Les constantes apparaissant dans le membre gauche sont ensuite déplacées dans le membre droit, pour obtenir une contrainte PB correspondant à la définition précédente. Par ailleurs, notons que, pour deux termes $\tau_1$ et $\tau_2$ conjointement cohérents, $(\chi|\tau_1)|\tau_2$ est équivalent à $\chi|(\tau_1 \wedge \tau_2)$.

### 2.2 Règles d'inférence et systèmes de preuve

Le système de preuve des *plans-coupes* [12] constitue la contre-partie du système de preuve de la *résolution* dans le cadre pseudo-booléen. Nous présentons ici quelques-unes de ses règles.

**Saturation.** La contrainte $al + \sum_{i=1}^{n} a_i l_i \geq d$, où $a > d$, est équivalente à $dl + \sum_{i=1}^{n} a_i l_i \geq d$. Autrement dit, toute contrainte PB peut être réécrite de manière à ce que tous ses coefficients soient au plus égaux à son degré.

***Clashing addition.*** La conjonction des deux contraintes PB $\chi_1 : al + \sum_{i=1}^{n} a_i l_i \geq d_1$ et $\chi_2 : b\bar{l} + \sum_{i=1}^{n} b_i l_i \geq d_2$, permet de déduire la contrainte $\sum_{i=1}^{n} (ba_i + ab_i) l_i \geq (bd_1 + ad_2 - ab)$. Notons que, lorsque $\chi_1$ et $\chi_2$ sont des clauses, cette opération est équivalente à la résolution classique [14].

**Division.** Étant donné un entier positif $\alpha$, la contrainte $\sum_{i=1}^{n} a_i l_i \geq d$ implique $\sum_{i=1}^{n} \lceil \frac{a_i}{\alpha} \rceil l_i \geq \lceil \frac{d}{\alpha} \rceil$. De plus, si tous les $a_i$ sont divisibles par $\alpha$, alors la première contrainte est équivalente à la seconde.

Dans la suite, nous appelons *résolution généralisée* le système de preuve utilisant les règles de saturation et de *clashing addition* et par *plans-coupes* le système de preuve autorisant toute combinaison linéaire de contraintes et la règle de division.

## 3 Littéraux non pertinents

Un problème spécifique aux contraintes PB, que l'on ne retrouve pas dans les clauses ou contraintes de cardinalité non valides, est l'existence de littéraux *non pertinents*.

**Definition 1** (Littéral non pertinent)**.** *Un littéral $l$ est dit* non pertinent *dans une contrainte $\chi$ lorsque $\chi|l \equiv \chi|\bar{l}$. Sinon, $l$ est dit* pertinent *dans $\chi$. Dans ce dernier cas, $\chi$ est aussi dite* dépendante *de $l$. De manière équivalente, $l$ est* non pertinent *lorsque, pour tout modèle $M$ de $\chi$, inverser la valeur de $l$ dans $M$ ne peut en faire un contre-modèle de $\chi$.*

Dans la suite, lorsqu'il n'y a pas d'ambiguïté sur la contrainte $\chi$, nous omettons cette contrainte et disons simplement que $l$ est pertinent ou non.

### 3.1 Impact des littéraux non pertinents

Considérons la contrainte PB $\chi$ définie comme suit :

$$\chi : 17a + 10b + 10c + \mathbf{d} + \mathbf{e} \geq 17$$

Nous observons que $d$ et $e$ ne sont ici pas pertinents : pour satisfaire $\chi$, il suffit de satisfaire $a$ ou de satisfaire $b$ et $c$. Donc, $d$ et $e$ peuvent être retirés de la contrainte. Pour ce faire, plusieurs solutions existent, la plus simple étant de se contenter de les ôter du membre gauche. Une approche plus intéressante consiste à satisfaire ces littéraux de manière à réduire le degré de la contrainte. En appliquant cela à notre exemple, nous obtenons la contrainte suivante :

$$\chi \equiv 17a + 10b + 10c \geq \mathbf{15}$$

Remarquons que la règle de saturation peut être appliquée sur cette contrainte, nous donnant ainsi :

$$\chi \equiv \mathbf{15a} + 10b + 10c \geq 15$$

Notons que tous les coefficients de cette contrainte sont divisibles par 5. En divisant par ce nombre, nous obtenons la contrainte équivalente suivante :

$$\chi \equiv 3a + 2b + 2c \geq 3$$

Cet exemple montre qu'en identifiant et en retirant les littéraux non pertinents d'une contrainte, il est possible de diminuer le nombre de ses littéraux, mais aussi la valeur de ses coefficients et de son degré. Ces simplifications permettent d'améliorer les performances du solveur, en détectant parfois des clauses ou des contraintes de cardinalité *cachées*, pour lesquelles les solveurs disposent de structures de données optimisées. En effet, il est possible qu'après avoir retiré tous les littéraux non pertinents, les coefficients de la contrainte obtenue soient tous égaux. Par exemple, considérons la contrainte $3a+3b+3c+3d+\mathbf{e}+\mathbf{f} \geq 6$. En retirant les littéraux non pertinents $e$ et $f$, la contrainte $3a + 3b + 3c + 3d \geq 4$ est produite. Il s'agit d'une contrainte de cardinalité, puisqu'en la divisant par 3, nous obtenons $a + b + c + d \geq 2$.

### 3.2 Littéraux inutiles dans les contraintes apprises

Comme nous venons de le montrer, les solveurs PB – contrairement aux solveurs SAT – peuvent être conduits à gérer des contraintes comportant des littéraux non pertinents. Ces contraintes peuvent soit provenir de la formule originale (même si cela ne se produit pas en pratique), soit être apprises lors de l'analyse de conflit. Rappelons que, lorsqu'un solveur rencontre un conflit, il applique successivement des règles de son système d'inférence pour déduire de nouvelles contraintes. Il se trouve qu'à partir de contraintes ne comportant que des littéraux pertinents, la résolution généralisée peut produire des contraintes dont certains littéraux ne le sont pas.

Par exemple, considérons les contraintes $4a + 2b + 2c+2y+x \geq 5$ et $3\bar{x}+d+e \geq 4$. En appliquant la règle de *clashing addition* sur ces deux contraintes, avec $x$ comme pivot, la contrainte suivante est produite :

$$12a + 6b + 6c + 6y + \mathbf{d} + \mathbf{e} \geq 16$$

Dans cette contrainte, $d$ et $e$ ne sont pas pertinents. Si nous résolvons maintenant cette nouvelle contrainte avec $6\bar{y} + 5a + 4b + 4c \geq 7$ en prenant $y$ comme pivot, nous obtenons la contrainte de notre exemple précédent :

$$17a + 10b + 10c + \mathbf{d} + \mathbf{e} \geq 17$$

Nos expérimentations (cf. section 5) montrent que ce scénario se produit en pratique assez fréquemment, et en particulier que de nombreux littéraux non pertinents peuvent apparaître dans les contraintes apprises par le solveur *Sat4j*, qui utilise la résolution généralisée.

### 3.3 Littéraux non pertinents et division

Récemment, *RoundingSat* [11] a proposé une amélioration dans l'implantation de solveurs PB visant à étendre la gamme des instances résolues efficacement par ces solveurs. Empiriquement, nous avons constaté que les contraintes produites par ce solveur peuvent également contenir des littéraux non pertinents, mais dans une mesure bien moindre que *Sat4j*. C'est un pas en avant dans la conception d'un système de preuve garantissant que les littéraux des contraintes inférées sont tous pertinents si c'est le cas des contraintes originales. Cependant, éviter la présence de ces littéraux n'est pas suffisant, puisqu'il est possible d'affaiblir une contrainte au point de rendre pertinents des littéraux qui ne le sont pas initialement. Malheureusement, cela peut se produire dans *RoundingSat*, comme nous le montrons ci-après. Retirer les littéraux non pertinents des contraintes produites par *RoundingSat* présente donc également un intérêt.

**Impact de la division.** Avant d'appliquer la règle de *clashing addition* au cours de l'analyse de conflit, *RoundingSat* applique sur les deux contraintes la division par le coefficient $c$ du littéral servant de pivot, afin de s'assurer qu'il devienne égal à 1. Les littéraux non falsifiés dont le coefficient n'est pas divisible par $c$ sont affaiblis, mais les coefficients des littéraux falsifiés sont divisés et arrondis par excès. Cette opération peut alors rendre *artificiellement* pertinents des littéraux qui ne le sont pas.

Considérons par exemple $6x + 3y + 3z + \mathbf{t} \geq 9$ où $t$ n'est pas pertinent. Supposons que $t$ soit falsifié, et que *RoundingSat* utilise $y$ comme pivot pour la *clashing addition*. En divisant la contrainte par 3, nous obtenons $2x+y+z+t \geq 3$, où $t$ est pertinent : il joue maintenant un rôle symétrique à celui de $y$ et $z$.

Notons que, si $t$ avait été retiré avant d'appliquer la division, nous aurions obtenu la contrainte plus forte $2x + y + z \geq 3$.

**Impact de la réduction en contrainte de cardinalité.** Lors de l'analyse de conflit, si *RoundingSat* détecte que les coefficients deviennent « trop grands », la contrainte apprise est réduite en une contrainte de cardinalité. Par exemple, $6x+3y+3z+\mathbf{t} \geq 9$ devient $x+y+z+t \geq 2$. Cette fois encore, une contrainte plus forte aurait pu être apprise en retirant le littéral non pertinent $t$, en l'occurrence $x + y + z \geq 2$.

## 4 Identification et suppression

Comme nous l'avons vu plus haut, supprimer les littéraux non pertinents des contraintes PB peut permettre de simplifier ces dernières. Malheureusement, il est connu que vérifier si un littéral est pertinent dans une contrainte PB donnée est NP-complet [5, Section 9.6]. En conséquence, vérifier la pertinence de chaque littéral de chacune des contraintes produites semble hors de portée d'un point de vue pratique. Cependant, comme nous le montrons ici, certaines propriétés des littéraux non pertinents peuvent être exploitées pour définir un algorithme détectant « efficacement » les littéraux non pertinents afin de les supprimer. Afin d'illustrer ces propriétés, nous considérons la pertinence du littéral $l$ dans la contrainte suivante, utilisée comme exemple récurrent :

$$\chi : al + \sum_{i=1}^{n} a_i l_i \geq d$$

### 4.1 Identifier un littéral non pertinent

Rappelons que $l$ n'est pas pertinent si, et seulement si, $\chi|l \equiv \chi|\bar{l}$, ou, autrement dit :

$$\sum_{i=1}^{n} a_i l_i \geq d - a \equiv \sum_{i=1}^{n} a_i l_i \geq d$$

Notons que, comme $d > d - a$, la seconde contrainte implique trivialement la première, donc la seule chose à vérifier est l'implication suivante :

$$\sum_{i=1}^{n} a_i l_i \geq d - a \models \sum_{i=1}^{n} a_i l_i \geq d$$

En conséquence, une première approche pour déterminer si $l$ est pertinent ou pas consiste à vérifier la cohérence de la conjonction des contraintes :

$$\sum_{i=1}^{n} a_i l_i \geq d - a \wedge \sum_{i=1}^{n} a_i l_i < d$$

Ceci revient, après normalisation, à vérifier la cohérence des deux contraintes PB :

$$\sum_{i=1}^{n} a_i l_i \geq d - a \wedge \sum_{i=1}^{n} a_i \bar{l_i} \geq \sum_{i=1}^{n} a_i - d$$

Cette formule est cohérente si, et seulement si, le littéral $l$ est pertinent. Ainsi, il suffit d'utiliser son solveur PB préféré pour tester la pertinence d'un littéral.

Notons que le fait qu'il n'y ait que deux contraintes ne signifie pas que ce problème est facile à résoudre (le problème NP-complet *subset-sum* se réduit à celui de la cohérence de deux contraintes PB). Ceci est d'autant plus vrai que le nombre de littéraux dans les contraintes est grand.

Une autre approche pour tester la pertinence de $l$ consiste à utiliser la *programmation dynamique*. En effet, rappelons l'implication que nous souhaitons vérifier :

$$\sum_{i=1}^{n} a_i l_i \geq d - a \models \sum_{i=1}^{n} a_i l_i \geq d$$

Observons que cette implication est vraie si, et seulement si, il n'existe aucune interprétation de $\sum_{i=1}^{n} a_i l_i$ égale à un nombre compris entre $d - a$ et $d - 1$. Pour vérifier la non-pertinence de $l$, il suffit de vérifier qu'il n'existe aucun sous-ensemble de $a_1, ..., a_n$ dont la somme vaut l'un de ces nombres, et donc, de résoudre *subset-sum* pour chacune de ces entrées.

Il est bien connu que cela peut se faire en temps $O(nd)$ avec un algorithme de programmation dynamique [4, Chapter 34.5], et donc en temps pseudo-polynomial dans la taille de la contrainte et de son degré. Dans cette approche, un tableau $S$ de valeurs booléennes est construit, et $S_j$ est vrai si, et seulement si, il existe un sous-ensemble des éléments dont la somme vaut $j$. Dans notre cas, pour savoir si $l$ n'est pas pertinent, il suffit de vérifier qu'une fois le tableau rempli, $S_j$ est faux pour tout $j$ entre $d - a$ et $d - 1$.

### 4.2 Réduire le nombre de tests

Vérifier la pertinence d'un littéral étant calculatoirement difficile, il n'est pas possible en pratique de faire le test pour *tous* les littéraux d'une contrainte. Heureusement, cela n'est pas nécessaire. Observons dans un premier temps que, si $l$ n'est pas pertinent, alors tous les littéraux ayant le même coefficient ne sont pas pertinents non plus, puisqu'ils sont tous symétriques dans la contrainte. Par ailleurs, la propriété suivante permet d'optimiser le nombre de tests à effectuer :

**Proposition 1.** *S'il existe $i_0$ tel que le littéral $l_{i_0}$, de coefficient $a_{i_0} < a$, est pertinent, alors $l$ est aussi pertinent.*

*Démonstration.* Par l'absurde, supposons que $l$ ne soit pas pertinent dans ce cas. $l_{i_0}$ étant supposé pertinent, il existe un modèle $M$ de $\chi$ tel que $M$ satisfait $l_{i_0}$, et falsifier ce littéral fait de $M$ un contre-modèle de $\chi$. Notons $M'$ le contre-modèle ainsi obtenu. Quelle que soit la valeur donnée au littéral non pertinent $l$, $M$ est un modèle de $\chi$ et $M'$ en est un contre-modèle, donc supposons que $l$ soit falsifié.

L'interprétation $M$ satisfait la contrainte suivante :

$$\chi|(\bar{l} \wedge l_{i_0}) \equiv \sum_{i=1, i \neq i_0}^{n} a_i l_i \geq d - a_{i_0} \qquad (1)$$

Observons que c'est aussi le cas de $M'$ puisque cette contrainte contient seulement des littéraux sur lesquels les deux interprétations coïncident.

Rappelons maintenant que $l$ est supposé non pertinent, donc changer sa valeur ne permet pas de faire de $M'$ un modèle de $\chi$. En particulier, la contrainte suivante ne peut donc pas être satisfaite par $M'$.

$$\chi|(l \wedge \bar{l_{i_0}}) \equiv \sum_{i=1, i \neq i_0}^{n} a_i l_i \geq d - a \qquad (2)$$

Cependant, comme $a_{i_0} < a$, alors $d - a_{i_0} > d - a$, donc (1) $\models$ (2), et $M'$ est un modèle de (2). Comme cela n'est pas possible, $l$ est nécessairement pertinent. □

Pour résumer nos observations, nous avons que :
- si plusieurs littéraux ont le même coefficient et que l'un d'entre eux n'est pas pertinent, alors aucun de ces littéraux ne l'est ;
- si une contrainte dépend d'un littéral, alors elle dépend de tous les littéraux ayant un coefficient plus grand que celui de ce littéral.

Grâce à ces résultats, nous pouvons réduire le nombre de tests de pertinence en ordonnant les littéraux par coefficients croissants. Un seul test par coefficient est nécessaire, et dès qu'un littéral pertinent est identifié, tous les autres littéraux sont pertinents. Cette approche est résumée par l'algorithme 1, dans lequel la fonction *dépendDe* peut être implantée soit avec un solveur PB, soit en utilisant la programmation dynamique

**Algorithme 1 :** ôter-littéraux-non-pertinents

**Entrée :** Une contrainte PB $\chi$ non valide
**Sortie :** $\chi$ sans ses littéraux non pertinents

Trier les coefficients de $\chi$ par ordre croissant
**pour chaque** *$c$ coefficient dans $\chi$* **faire**
  Choisir un littéral $l$ ayant $c$ pour coefficient
  **si** *dépendDe*$(l, \chi)$ **alors**
    **retourner** $\chi$
  **fin**
  Retirer tous les littéraux de $\chi$ de coefficient $c$
  Adapter le degré de $\chi$
  Saturer $\chi$
**fin**

**Proposition 2.** *L'algorithme 1 est correct et se termine.*

*Démonstration.* D'une part, l'algorithme est trivialement correct de par nos résultats précédents. D'autre part, la contrainte $\chi$ contient un nombre fini de coefficients, et puisqu'elle n'est pas valide, elle dépend nécessairement d'au moins l'un de ses littéraux. □

Considérons par exemple la contrainte suivante (l'un de nos précédents exemples) comme entrée de notre algorithme :

$$17a + 10b + 10c + d + e \geq 17$$

Le premier coefficient est 1, et soit $d$ soit $e$ est choisi. Le test de pertinence révèle que le littéral choisi ne l'est pas, et donc $d$ et $e$ sont retirés tous les deux. Le degré est mis à jour, et la contrainte saturée :

$$15a + 10b + 10c \geq 15$$

Le coefficient considéré ensuite est 10, et soit $b$ soit $c$ est choisi. Le littéral est pertinent, donc la contrainte ne peut plus être modifiée. Elle est donc apprise sous cette forme (la détection du diviseur commun est laissée au solveur).

## 5 Résultats expérimentaux

Cette section présente des résultats expérimentaux illustrant la présence de littéraux non pertinents dans les contraintes apprises, ainsi que leur impact sur les performances des solveurs PB. Toutes les expérimentations présentées dans cette section ont été réalisées sur un cluster de calcul équipé de bi-processeurs quadri-cœur Intel XEON E5-5637 v4 (3.5 GHz) et de 128 Go de mémoire. Les instances considérées sont les 777 instances de décision soumises à la dernière évaluation pseudo-booléenne (`http://www.cril.univ-artois.fr/PB16/`). Nous avons implanté trois approches pour détecter les littéraux non pertinents, utilisant soit la résolution généralisée avec le solveur *Sat4j-CP*, soit la résolution avec le solveur *Sat4j-Res*, soit un algorithme de programmation dynamique résolvant *subset-sum*. Les diverses implantations sont disponibles à l'adresse `https://gitlab.ow2.org/sat4j/sat4j/tree/sat19`. Pour rester efficace, seules les contraintes comportant moins de 1000 littéraux et ayant un degré inférieur à 20000 sont traitées. Les autres contraintes ne sont pas modifiées. Par ailleurs, un délai de garde de 5 secondes est fixé aux deux approches utilisant un solveur : lorsqu'il est atteint, le littéral est supposé pertinent, rendant ces approches incomplètes mais correctes. L'approche par programmation dynamique ne nécessite pas de tel délai, les bornes choisies fournissant suffisamment de garanties quant à sa complexité temporelle.

### 5.1 Expérimentations préliminaires

Une étude préalable des instances considérées a révélé que les littéraux de leurs contraintes sont tous pertinents. Cela semble naturel, puisqu'elles modélisent des

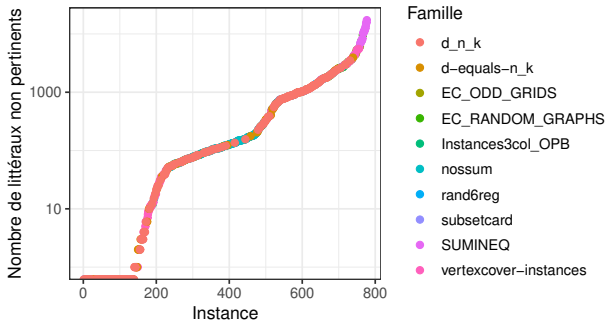



FIGURE 1 – Nombre de littéraux non pertinents par instance. Chaque point est une instance, et son ordonnée est le nombre de littéraux non pertinents détectés dans les contraintes apprises par *Sat4j-CP* sur cette instance (en échelle logarithmique).

problèmes réels et que la phase de modélisation n'introduit normalement pas de littéraux non pertinents.

Afin d'évaluer dans quelle mesure la résolution généralisée produit des littéraux non pertinents sur ces instances, nous avons exécuté *Sat4j-CP* [15] sur ces instances et récupéré pour chacune d'elles les 5000 premières contraintes apprises. La figure 1 montre le nombre de littéraux non pertinents qui y sont détectés.

Pour ces expérimentations, nous avons utilisé les trois approches proposées. Même si celles-ci détectent un nombre équivalent de littéraux non pertinents, la figure 2 montre que la plus efficace est celle utilisant la programmation dynamique.

Nous avons réalisé les mêmes expérimentations avec *RoundingSat* [11]. Celles-ci ont révélé que ses contraintes apprises ne comportent que très peu de littéraux non pertinents : aucun pour une grande majorité d'instances, et jusqu'à 12 pour 15 instances parmi les 24 de la famille `SUMINEQ`. Comme nous l'avons vu plus haut, cela est dû au fait que la division appliquée par *RoundingSat* rend pertinents des littéraux qui ne le sont pas dans ses contraintes apprises.

## 5.2 Impact du retrait

Comme nous l'avons montré dans une précédente section, le retrait des littéraux non pertinents peut, d'un point de vue théorique, simplifier les contraintes et ainsi améliorer les performances du solveur. Pour évaluer son impact pratique, nous l'avons implanté dans *Sat4j-CP*. Plus précisément, à l'issue de chaque analyse de conflit, la contrainte apprise est traitée de manière à en retirer les littéraux non pertinents. Pour des raisons d'efficacité, seule cette contrainte est considérée ; les contraintes intermédiaires ne sont pas

FIGURE 2 – Comparaison des trois approches en termes de temps d'exécution (en échelle logarithmique). L'abscisse mesure le temps de l'approche par programmation dynamique, et l'ordonnée celui des approches utilisant *Sat4j* : *Sat4j-CP* (en haut) et *Sat4j-Res* (en bas). Les couleurs utilisées sont les mêmes qu'à la figure 1.

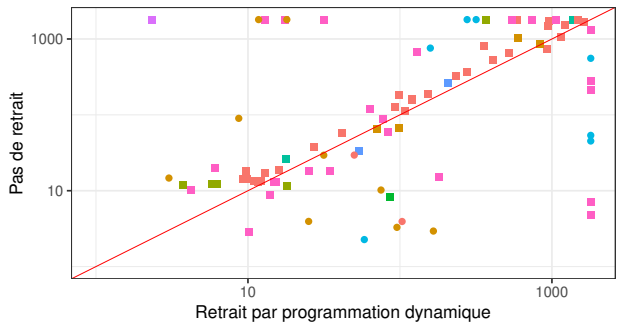



FIGURE 3 – Comparaison des temps d'exécution de *Sat4j-CP* en utilisant le retrait des littéraux non pertinents par programmation dynamique et sans ce retrait (en échelle logarithmique). Les cercles et carrés représentent les instances cohérentes et contradictoires, respectivement. Les couleurs représentent les mêmes familles qu'à la figure 1.

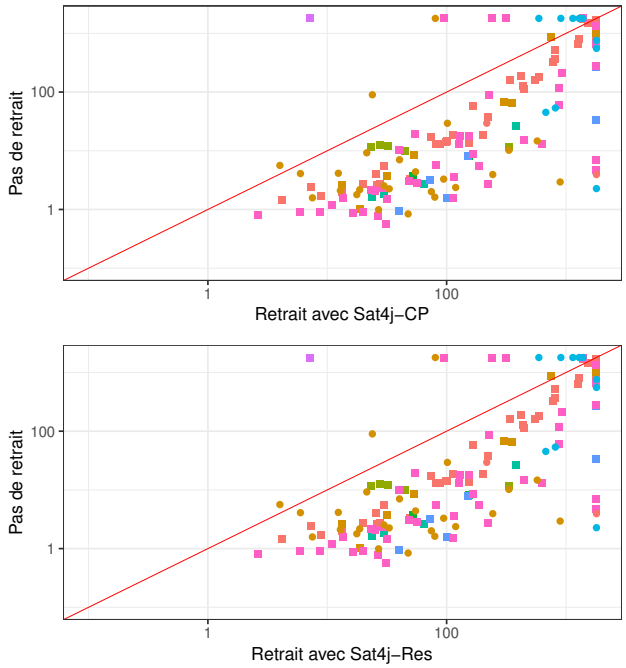



FIGURE 4 – Comparaison des temps d'exécution de *Sat4j-CP* en utilisant le retrait des littéraux non pertinents avec *Sat4j-CP* (en haut) et *Sat4j-Res* (en bas) et sans ce retrait (en échelle logarithmique). Les cercles et carrés représentent les instances cohérentes et contradictoires, respectivement. Les couleurs représentent les mêmes familles qu'à la figure 1.

modifiées. La figure 3 compare les temps d'exécution du solveur lorsque le retrait des littéraux non pertinents est réalisé par programmation dynamique avec ceux obtenus par le solveur sans ce retrait. Pour plus de clarté, les instances résolues en moins de 10 secondes par chacune des deux approches sont omises. Il en va de même pour celles n'étant résolues dans aucun cas.

Le diagramme de dispersion obtenu ne permet pas de tirer des conclusions aussi claires que celles résultant du diagramme en figure 2 : *Sat4j-CP* est assez sensible aux changements dans ses contraintes, donc modifier ses contraintes apprises peut modifier son espace de recherche, ce qui peut expliquer les différences de temps observées. Cependant, il montre que, en dépit de la complexité de la détection des littéraux non pertinents, activer leur retrait permet une amélioration des performances du solveur. En effet, son temps d'exécution est par exemple diminué pour chacune des instances de la famille `d_n_k`.

En revanche, comme le montre la figure 4, les approches utilisant un solveur pour détecter les littéraux non pertinents ne sont pas assez efficaces pour contrebalancer le coût de la détection des littéraux non pertinents.

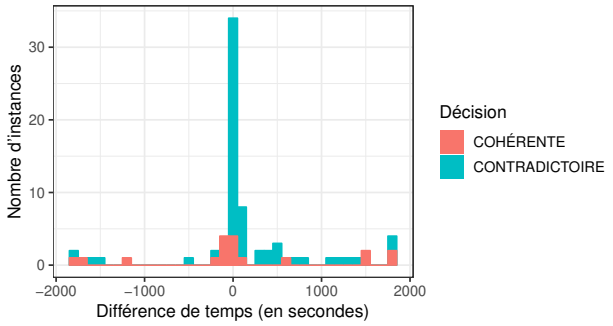



FIGURE 5 – Histogramme des différences de temps d'exécution entre *Sat4j-CP* où le retrait des littéraux non pertinents est activé ou pas. Les barres à gauche de 0 sont mieux résolues sans le retrait, et celles à droite de 0 sont mieux résolues avec.

### 5.3 Effets concrets sur le solveur

Puisque l'approche par programmation dynamique produit les meilleurs résultats, nous étudions ici ses effets pratiques sur le solveur. D'une part, comme le suggère la figure 3, les instances *faciles*, résolues en moins de 100 secondes, ne sont pas résolues aussi efficacement lorsque la détection est activée. C'est le prix à payer pour résoudre un problème calculatoirement difficile pendant l'analyse de conflit : comme les instances sont résolues rapidement, il n'y a pas assez de temps pour amortir son coût. D'autre part, les instances difficiles, et plus particulièrement celles étant incohérentes (il est bien connu que les instances cohérentes sont résolues par « chance »), sont résolues plus rapidement lorsque la détection est appliquée. Ce gain est illustré à la figure 5.

Notons que, bien que 7 instances n'ont pas pu être résolues avec le retrait activé, 15 de plus sont résolues avec cette fonctionnalité. Des statistiques sur ces instances sont données dans la table 1. Notons que l'instance `vertexcover-grid10x25` est omise car le solveur n'a pas fait d'affichage dans le temps imparti.

La table 1 révèle que la vitesse (i.e. le nombre d'affectations réalisées par le solveur chaque seconde) augmente pour les instances gagnées, et diminue pour les instances perdues. Dans les instances perdues, peu de littéraux sont détectés comme non pertinents, surtout lorsque l'on compare leur nombre à celui des instances gagnées, où l'on en détecte un grand nombre. Nous pouvons également remarquer que, pour ces instances, de nombreuses clauses sont identifiées, permettant ainsi au solveur d'optimiser ses structures de données internes. Les nombreuses saturations appliquées permettent également de réduire les coefficients, contribuant à améliorer l'efficacité de la résolution généralisée.

| Instance | Contraintes inchangées | Contraintes modifiées | Littéraux retirés | Saturations déclenchées | Clauses détectées | Vitesse avec retrait (affect./s) | Vitesse sans retrait (affect./s) |
|---|---|---|---|---|---|---|---|
| **3col-left3reg-l050-r049-n1** | 16286 | 10342 | 18030 | 10325 | 5544 | **10561** | 6595 |
| **5_6_19** | 31210 | 3959 | 10337 | 3957 | 3201 | **2953** | 2311 |
| **8_6_18** | 23065 | 3589 | 11622 | 3586 | 2904 | **5023** | 2249 |
| **compression16_11** | 2592 | 64 | 472 | 60 | 42 | **22572** | 1161 |
| **compression16_12** | 2596 | 54 | 336 | 53 | 36 | **15992** | 1199 |
| **ECgrid5x20split** | 35514 | 27841 | 72576 | 26734 | 24811 | **1760** | 597 |
| **sha1-size80-round21-1** | 6151 | 59 | 448 | 53 | 30 | **6085** | 1995 |
| **sha1-size80-round21-5** | 5817 | 41 | 193 | 35 | 20 | **5712** | 1886 |
| **sumineqArity3pyramidP0016** | 391 | 126 | 1270 | 122 | 120 | **7334** | 305 |
| **vertexcover-grid10x19-hard** | 7693 | 652 | 14025 | 629 | 619 | **834** | 236 |
| **vertexcover-grid10x19** | 1237 | 50 | 1255 | 47 | 45 | **3056** | 37 |
| **vertexcover-grid10x21** | 1615 | 50 | 1111 | 49 | 47 | **4513** | 50 |
| **vertexcover-grid10x29** | 4514 | 204 | 9309 | 191 | 181 | **592** | 291 |
| **vertexcover-grid8x17-hard** | 1641 | 85 | 1539 | 79 | 76 | **5320** | 105 |
| *sha1-size80-round21-4* | 8269 | 44 | 188 | 33 | 13 | 1973 | **19504** |
| *sha1-size96-round21-5* | 8066 | 59 | 262 | 42 | 22 | 2102 | **4302** |
| *sha1-size96-round21-6* | 8496 | 54 | 199 | 33 | 59 | 67 | **10067** |
| *vertexcover-grid10x13-hard* | 1075 | 40 | 1077 | 40 | 40 | 36 | **234** |
| *vertexcover-grid10x17* | 1598 | 46 | 977 | 4 | 33 | 89 | **7261** |
| *vertexcover-grid10x23* | 1231 | 24 | 547 | 19 | 15 | 72 | **493** |

TABLE 1 – Détails sur les instances **gagnées** et *perdues*

## 6 Conclusion et perspectives

Dans cet article, nous avons montré que les contraintes pseudo-booléennes, contrairement aux clauses et aux contraintes de cardinalité, peuvent contenir des littéraux non pertinents. En particulier, de tels littéraux peuvent être introduits dans les contraintes apprises par l'application de la résolution généralisée, même si la formule considérée en entrée n'en contient pas. Nous avons proposé plusieurs approches permettant de détecter ces littéraux, fondées soit sur un solveur PB, soit sur un algorithme de programmation dynamique résolvant *subset-sum*. Nos résultats expérimentaux montrent que l'approche par programmation dynamique est la technique la plus efficace parmi celles considérées. En dépit de sa complexité élevée, cette détection réduit le temps d'exécution de *Sat4j* sur de nombreuses instances de l'évaluation pseudo-booléenne 2016. Un tel gain peut s'expliquer par le fait que le retrait de ces littéraux conduit à une réduction de la taille des contraintes, à la fois en termes de nombre de littéraux et de taille de coefficients. Ce retrait peut également révéler, dans certains cas, des clauses ou des contraintes de cardinalité cachées. Combinées, ces opérations permettent d'obtenir des contraintes pour lesquelles la détection des propagations et l'application de la résolution généralisée sont plus efficaces. La présence de littéraux non pertinents indique également que les contraintes produites par *Sat4j* ne sont pas aussi fortes que ce qu'elles pourraient être. *RoundingSat*, quant à lui, produit moins de littéraux non pertinents. Cependant, des littéraux initialement non pertinents peuvent devenir *artificiellement* pertinents en appliquant un arrondi lors de la division de la contrainte. De ce fait, le nombre de littéraux non pertinents ne peut pas être considéré comme une mesure de la qualité des contraintes apprises. Des travaux supplémentaires sont nécessaires pour étudier la présence de littéraux non pertinents dans les contraintes intervenant dans l'analyse de conflit.

Notre but ultime serait de définir un système de preuve assurant de ne produire que des littéraux pertinents à partir de contraintes qui ne comportent que de tels littéraux. La principale difficulté est de trouver des règles qui n'affectent pas la pertinence des littéraux, contrairement à la règle de division. Notons que la restriction de cette règle à la division de contraintes dans lesquelles tous les coefficients sont divisibles par

le diviseur (et donc pour lesquelles seul le degré est arrondi) n'affecte pas la pertinence des littéraux.

D'un point de vue plus pragmatique, notre algorithme de détection pourrait être amélioré, par exemple en approximant *subset-sum* plutôt qu'en le résolvant de manière exacte. Cela pourrait permettre de supprimer systématiquement tous les littéraux non pertinents des contraintes produites par *Sat4j*. Rappelons qu'actuellement, notre algorithme de détection s'applique uniquement à la contrainte apprise à l'issue de l'analyse de conflit. Il serait alors possible de s'assurer qu'aucune des contraintes produites ne contienne de littéraux non pertinents.

## Références

[1] Armin Biere, Daniel Le Berre, Emmanuel Lonca et Norbert Manthey : Detecting cardinality constraints in CNF. *In Theory and Applications of Satisfiability Testing*, pages 285–301, 2014.

[2] Donald Chai et Andreas Kuehlmann : A fast pseudo-Boolean constraint solver. *IEEE Trans. on CAD of Integrated Circuits and Systems*, pages 305–317, 2005.

[3] William Cook, Collette R. Coullard et György Turán : On the Complexity of Cutting-plane Proofs. *Discrete Appl. Math.*, pages 25–38, 1987.

[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest et Clifford Stein : *Introduction to Algorithms, Third Edition.* 2009.

[5] Yves Crama et Peter L. Hammer : *Boolean Functions : Theory, Algorithms, and Applications.* 2011.

[6] Heidi Dixon : *Automating Pseudo-Boolean Inference Within a DPLL Framework.* Thèse de doctorat, University of Oregon, 2004.

[7] Heidi E. Dixon et Matthew L. Ginsberg : Inference methods for a pseudo-boolean satisfiability solver. *In AAAI'02*, pages 635–640, 2002.

[8] Heidi E. Dixon, Matthew L. Ginsberg et Andrew J. Parkes : Generalizing boolean satisfiability I : background and survey of existing work. *Journal of Artificial Intelligence Research*, pages 193–243, 2004.

[9] Niklas Eén et Niklas Sörensson : An extensible sat-solver. *In Theory and Applications of Satisfiability Testing*, pages 502–518, 2004.

[10] Jan Elffers, Jesús Giráldez-Crú, Jakob Nordström et Marc Vinyals : Using combinatorial benchmarks to probe the reasoning power of pseudo-boolean solvers. *In Theory and Applications of Satisfiability Testing*, pages 75–93, 2018.

[11] Jan Elffers et Jakob Nordström : Divide and conquer : Towards faster pseudo-boolean solving. *In Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 1291–1299, 2018.

[12] Ralph E. Gomory : Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, pages 275–278, 1958.

[13] Armin Haken : The intractability of resolution. *Theoretical Computer Science*, pages 297–308, 1985.

[14] John N. Hooker : Generalized resolution and cutting planes. *Annals of Operations Research*, pages 217–239, 1988.

[15] Daniel Le Berre et Anne Parrain : The SAT4J library, Release 2.2, System Description. *Journal on Satisfiability, Boolean Modeling and Computation*, pages 59–64, 2010.

[16] Vasco Manquinho et Olivier Roussel : The first evaluation of pseudo-boolean solvers (pb'05). *JSAT*, pages 103–143, 2006.

[17] Joao Marques-Silva et Karem A. Sakallah : Grasp : A search algorithm for propositional satisfiability. *IEEE Trans. Computers*, pages 220–227, 1999.

[18] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang et Sharad Malik : Chaff : Engineering an efficient sat solver. *In Proceedings of the 38th Annual Design Automation Conference*, pages 530–535, 2001.

[19] Jakob Nordström : On the Interplay Between Proof Complexity and SAT Solving. *ACM SIGLOG News*, pages 19–44, 2015.

[20] Olivier Roussel et Vasco M. Manquinho : Pseudo-Boolean and Cardinality Constraints. *In Handbook of Satisfiability*, chapitre 22, pages 695––733. 2009.

[21] Hossein M. Sheini et Karem A. Sakallah : Pueblo : A Hybrid Pseudo-Boolean SAT Solver. *JSAT*, pages 165–189, 2006.

[22] Marc Vinyals, Jan Elffers, Jesús Giráldez-Crú, Stephan Gocht et Jakob Nordström : In between resolution and cutting planes : A study of proof systems for pseudo-boolean SAT solving. *In Theory and Applications of Satisfiability Testing*, pages 292–310, 2018.

# Contraintes de cardinalité cachées dans les preuves d'insatisfaisabilité

**Valentin Montmirail**[1] **Marie Pelleau**[1] **Jean-Charles Régin**[1] **Laurent Simon**[2]
[1] Université Côte d'Azur, CNRS, I3S, Nice, France
[2] Université de Bordeaux, CNRS, LaBRI, Talence, France
prénom.nom@univ-cotedazur.fr lsimon@labri.fr

**Résumé**

Les solveurs SAT sont utilisés avec succès dans de nombreuses applications combinatoires et du monde réel. Il n'est maintenant pas rare de lire qu'une preuve mathématique implique des centaines de gigaoctets de traces de solveur SAT. L'ampleur de la preuve commence à constituer une véritable limite à l'approche globale. Dans ce travail, nous proposons de rechercher des contraintes de haut niveau dans les preuves UNSAT. Un travail similaire a été proposé il y a quelques annéespour les contraintes de cardinalité les plus simples. Nous étendons cette idée à un cas plus général (sans limitation sur les bornes des contraintes de cardinalité) en généralisant l'algorithme de Bron & Kerbosh (pour l'énumération de cliques dans les graphes) aux hypergraphes. Nous démontrons expérimentalement la capacité de notre approche à trouver des contraintes de cardinalité dans les grandes preuves, ouvrant ainsi un nouveau moyen de générer des preuves plus courtes et compréhensibles pour les problèmes difficiles.

**Abstract**

SAT solvers are successfully used in many real-world and combinatoric applications. It is now not uncommon to read that a mathematical proof implied hundreds of gigabytes of SAT solver traces. The size of the proof by itself begins to be a real limit to the whole approach. In this work, we propose to search for higher-level constraints in UNSAT proofs. A similar work was proposed a few years ago, but only on the original formulafor the most simple cardinality constraints. We extend this idea to a more general case (with no limitations on the bounds of the cardinality constraints) by generalizing the Bron & Kerbosh algorithm (for clique enumeration in graphs) to hypergraphs. We experimentaly demonstrate the ability of our approach to find for cardinality constraints in large proofs, opening a new way of generating more shorter and human-understandable UNSAT proofs of hard problems.

## 1 Introduction

Le succès des solveurs SAT dans la résolution de problèmes réels et combinatoires repose sur leur capacité à construire efficacement de preuves UNSAT. La plupart des solveurs actuels, basés sur le framework *Conflict-Driven Clause Learning* (CDCL) [22], peuvent produire des dizaines de milliers de lemmes par seconde, grâce à une intégration de nombreux ingrédients (2-Watched Literals, VSIDS, prétraitement, redémarrages, *etc.*) Cependant, même si les ingrédients sont bien connus, la stratégie globale d'un solveur est encore inconnue et est difficilement expliquée. En conséquence, quand une formule s'avère être UNSAT, le seul moyen de prouver le résultat est de "rejouer" les étapes élémentaires du solveur, en veillant à ce que chaque clause de la preuve puisse être déduite par résolution.

Beaucoup de progrès ont été réalisés ces dernières années dans la génération, la manipulation et la vérification des preuves UNSAT. Cependant, la plupart des efforts sont consacrés à la conception d'outils [13, 26] afin de garantir de la correction des solveurs SAT. La limite de ces approches est néanmoins la taille et l'expressivité de la preuve. Jusqu'à présent, il n'est pas possible de l'exploiter pour comprendre la preuve ou même l'expliquer. Il n'est maintenant plus rare de voir qu'une preuve mathématique implique des centaines de tera-octets de traces d'un solveur [16, 14]. Cependant, vérifier que la preuve est correcte ne donne aucun indice sur le fonctionnement du solveur. Plus important encore, la taille de la preuve, elle-même commence à être une véritable limite à l'ensemble de l'approche.

Dans ce travail, nous proposons de rechercher des contraintes de haut niveau dans les preuves. Nous montrons que, même si ces contraintes ne sont pas présentes dans la formule originale, les solveurs SAT ajoutent

généralement ces contraintes *in intenso* au cours de leur recherche. Il existe de nombreux travaux sur la détection d'informations dans des CNF. Nous pouvons citer en exemple l'idée de détecter les contraintes de cardinalité en utilisant la propagation unitaire [4], des équations booléennes [23], des variables dépendantes [12], *etc.* Cependant, il y a moins de travaux concernant l'analyse de la preuve générée. L'un d'eux était centré sur le pourcentage de temps consacré par le solveur à des calculs inutiles pour la preuve finale [24], un autre sur le lien entre les mesures de complexité de la preuve et la difficulté des instances [18]. Un travail récent [10] analyse le graphe de dépendance généré par la résolution pour détecter quel ensemble de clauses apprises est nécessaire pour dériver la contradiction finale.

Afin de trouver de la structure dans les preuves UNSAT, nous proposons de rechercher des contraintes de cardinalité dans la preuve DRAT [26] produite par un solveur CDCL, en étendant les approches précédentes à un cas plus général (sans limite sur les bornes des contraintes de cardinalité). Notre algorithme de détection est une première contribution : la taille de la formule dans laquelle la recherche est lancée est nettement plus grande que dans les travaux précédents. Ceci est rendu possible grâce à une généralisation de l'algorithme de Bron & Kerbosh (pour l'énumération de cliques dans les graphes) [8] aux hypergraphes que nous proposons dans cet article. Nous montrons expérimentalement que notre approche peut trouver des contraintes de cardinalité dans de grandes preuves. De plus, nous obtenons des preuves étonnamment plus courtes et compréhensibles sur certains cas typiques. Il est étonnant de constater que, même lors de la résolution d'instances aléatoires, les solveurs ont tendance à produire de nombreuses contraintes de cardinalité dans les preuves. Nous nous attendons à ce que cette découverte ouvre de nouvelles améliorations afin de comprendre comment les solveurss se comportent.

## 2 Préliminaires

Définissons quelques notions sur la logique propositionnelle et la théorie des graphes.

### 2.1 Logique propositionnelle et SAT

Nous considérons que le lecteur est familiarisé avec la logique propositionnelle et les bases des solveurs SAT de type *conflict-driven clause learning* (CDCL). Nous orientons le lecteur intéressé vers [5] pour une introduction approfondie du sujet. Nous présentons quelques notions essentielles pour plus de clarté.

*Remarque 2.1* – Nous appelons clause *positive* (resp. *négative*), une clause contenant uniquement des littéraux positifs (resp. négatifs). Nous appelons *uniforme* une clause négative ou positive.

Les solveurs CDCL utilisent le principe de résolution à chaque analyse de conflit. Si le certificat pour les instances satisfiables est simplement l'affectation de variables évaluant la formule à vrai, le certificat pour les formules non satisfaisables s'appuie indirectement sur la règle de résolution. Chaque fois que le solveur CDCL apprend une nouvelle clause, le lemme obtenu par résolution est enregistré et fera partie de la preuve.

DRAT (deletion resolution asymmetric tautology) [26] est un exemple populaire de système de preuve clausal, le standard des preuves non satisfaisantes dans la résolution pratique de SAT, qui est une généralisation du format DRUP (deletion reverse unit propagation) [13]. DRAT autorise l'ajout d'une clause s'il s'agit d'une clause dite *Resolution Asymmetric Tautology* (RAT) [17]. Comme il est possible de vérifier efficacement si une clause est une RAT et que les RAT couvrent une grande partie des clauses redondantes, le système de preuve DRAT est très puissant. L'importance de la redondance est de certifier que si nous obtenons $\perp$ en ajoutant uniquement des lemmes qui ne changent pas la satisfiabilité de la formule, alors par transitivité, la formule est insatisfiable. RAT est défini comme suit :

**Définition 2.1** (*Resolution Asymmetric Tautology* [17])**.** Soit $\Sigma$ une formule et $C$ une clause. Alors, $C$ est une *resolution asymmetric tautology* en fonction de $\Sigma$ s'il contient un littéral $l$ tel que, pour chaque clause $d \in \Sigma|_{\bar{l}}$, la propagation unitaire sur la négation des littéraux de $(d \setminus \{l\})$ sur $\Sigma$ dérive un conflit.

Il est relativement facile d'émettre une preuve DRAT à partir d'un solveur CDCL. Les solveurs CDCL gèrent une base de données contenant les clauses d'origine, et la création d'une preuve DRAT consiste généralement à imprimer chaque modification sur la base de données (*c.-à-d.* la suppression d'une clause ou l'ajout de nouvelles). Maintenant que nous avons introduit comment les solveurs SAT peuvent produire une preuve, passons en revue certaines notions de la théorie des graphes.

### 2.2 Théorie des graphes

La première notion dont nous avons besoin est la notion d'hyperarête, définie comme suit :

**Définition 2.2** (Hypergraphe [3])**.** Soit $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ un ensemble fini. Un *hypergraphe* sur $\mathcal{X}$ est une famille $H = (E_1, E_2, \ldots, E_m)$ de sous-ensembles de $\mathcal{X}$ tel que :

$$E_i \neq \emptyset \qquad (i = 1, 2, \ldots, m) \tag{1}$$

$$\bigcup_{i=1}^{m} E_i = \mathcal{X} \tag{2}$$

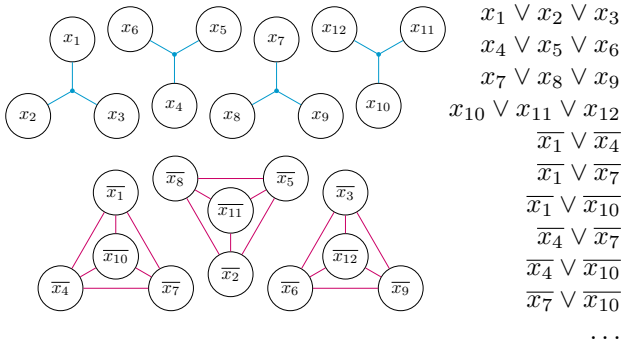



**Figure 1** – Le problème des pigeonniers en encodage naïf et sa représentation en hypergraphe.

Les éléments de $\mathcal{X}$ sont appelés *sommets*, et les ensembles $E_1, E_2, \ldots, E_m$ sont les *arêtes* de l'hypergraphe (ou *hyperarêtes*). L'*ordre* de $H$ est le nombre de sommets. Un hypergraphe est dit *simple* si aucune arête n'en contient une autre ($E_i \subseteq E_j \Longrightarrow i = j$).

**Définition 2.3** (Hypergraphe uniforme)**.** Le *rang* est $r(H) = \max_j |E_j|$, l'*anti-rang* est $s(H) = \min_j |E_j|$. $H$ est un hypergraphe *uniforme* si $r(H) = s(H)$. Un hypergraphe simple de rang $r$ est également appelé $r$-uniforme ou $r$-hypergraphe.

**Définition 2.4** (Hypergraphe $r$-uniforme complet)**.** L'hypergraphe *$r$-uniforme complet* ou *$r$-complet*, noté $K_n^r$, est l'hypergraphe contenant $n$ sommets dans lequel chaque sous-ensemble de taille $r$ des sommets représentent une arête. On observe facilement que le nombre d'arêtes dans $K_n^r$ est $\binom{n}{r} = \frac{n!}{r!(n-r)!}$

*Remarque 2.2* – Dans la suite, nous appelons *clique* $\mathcal{C}$ d'ordre $n$ un hypergraphe $r$-complet $K_n^r$. Par abus de notation, $\mathcal{C}$ correspond à un sous-ensemble de sommets.

Maintenant que nous avons rappelé suffisamment de définitions de la théorie des graphes, passons au codage en graphe d'une preuve SAT et à la détection des contraintes qui s'y trouvent. L'idée pour représenter la formule est : tout ensemble de clauses (*c.-à-d.* une formule d'entrée ou une preuve DRAT), peut être représenté à l'aide d'un hypergraphe où les sommets correspondent aux littéraux et les arêtes aux clauses.

*Exemple 2.1* – Le fameux exemple des pigeonniers, avec l'encodage naïf, correspond à l'hypergraphe de la Fig. 1. Ici, il y a 4 pigeons et 3 pigeonniers.

## 3 Détection des contraintes de cardinalité

La détection des contraintes de cardinalité dans les formules exprimées en CNF avait déjà été proposée dans [4]. Le but de ce travail était d'accélérer le temps de résolution des instances difficiles à résoudre en proposant une approche à la volée. Les contraintes sont détectées lors de la recherche du solveur SAT et n'ont pas lors d'une approche post-mortem sur la preuve.

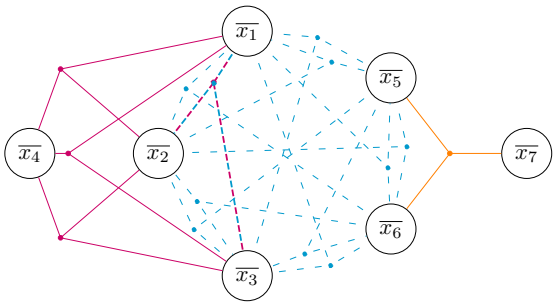


**Figure 2** – Hypergraphe de la CNF de l'exemple 3.1.

### 3.1 Détection pour les clauses uniformes

Si on représente les clauses négatives binaires de l'encodage naïf de la contrainte AtMost-1 par un graphe, il correspond à une clique. Nous étendons cette affirmation aux clauses d'arité $k$. Une clause d'arité $k$ ($\overline{x_1} \vee \overline{x_2} \vee \cdots \vee \overline{x_k}$) est équivalente à :

$$\begin{aligned} (1 - x_1) + \cdots + (1 - x_k) &\geq 1 \\ k - x_1 - \cdots - x_k &\geq 1 \\ k - 1 &\geq x_1 + \cdots + x_k \end{aligned}$$

En d'autres termes, Une clause d'arité $k$ ($\overline{x_1} \vee \overline{x_2} \vee \cdots \vee \overline{x_k}$) est équivalente à une contrainte AtMost-$(k-1)$. Toute CNF peut être représentée sous la forme d'un hypergraphe dont les sommets correspondent aux littéraux et les arêtes aux clauses.

*Exemple 3.1* – Considérons la CNF suivante :

$$\begin{array}{lll} \overline{x_1} \vee \overline{x_2} \vee \overline{x_3} & \overline{x_1} \vee \overline{x_2} \vee \overline{x_4} & \overline{x_1} \vee \overline{x_2} \vee \overline{x_5} \\ \overline{x_1} \vee \overline{x_2} \vee \overline{x_6} & \overline{x_1} \vee \overline{x_3} \vee \overline{x_4} & \overline{x_1} \vee \overline{x_3} \vee \overline{x_5} \\ \overline{x_1} \vee \overline{x_3} \vee \overline{x_6} & \overline{x_1} \vee \overline{x_5} \vee \overline{x_6} & \overline{x_2} \vee \overline{x_3} \vee \overline{x_4} \\ \overline{x_2} \vee \overline{x_3} \vee \overline{x_5} & \overline{x_2} \vee \overline{x_3} \vee \overline{x_6} & \overline{x_2} \vee \overline{x_5} \vee \overline{x_6} \\ \overline{x_3} \vee \overline{x_5} \vee \overline{x_6} & \overline{x_5} \vee \overline{x_6} \vee \overline{x_7} & \end{array}$$

Cette instance peut être représentée par l'hypergraphe figure 2.

**Proposition 3.1.** *Dans un hypergraphe $r$-uniforme représentant les clauses négatives d'une CNF, une clique $\mathcal{C}$ d'ordre $n$ avec $n \geq r$ correspond à une contrainte AtMost-$(r-1)$, autrement dit $\sum_{x \in \mathcal{C}} x \leq r - 1$.*

*Démonstration.*

Par récurrence. Dans un $r$-hypergraphe représentant les clauses négatives d'une CNF, une hyperarête est une clique particulière d'ordre $r$ et correspond à une contrainte AtMost-$(r-1)$. Supposons que la propriété soit valable pour une clique d'ordre $n$. Soit $\mathcal{C} = \{x_1, x_2, \ldots, x_{n+1}\}$ une clique d'ordre $n+1$. On note $\mathcal{C}_{\setminus x}$ la clique $\mathcal{C}$ dans laquelle le sommet $x$ et toutes les arêtes auxquelles il appartient ont été supprimés. Pour tout $x \in \mathcal{C}$, $\mathcal{C}_{\setminus x}$ est une clique d'ordre $n$. Ainsi, la clique $\mathcal{C}$ correspond à $n+1$ contraintes AtMost-$(r-1)$, et chaque sommet $x \in \mathcal{C}$ apparaît dans exactement $n$ contraintes (toutes les cliques sauf $\mathcal{C}_{\setminus x}$). Si nous additionnons toutes les contraintes AtMost, nous obtenons :

$$\begin{aligned} nx_1 + nx_2 + \cdots + nx_{n+1} &\leq (n+1)(r-1) \\ x_1 + x_2 + \cdots + x_{n+1} &\leq \lfloor \tfrac{(n+1)}{n}(r-1) \rfloor \\ x_1 + x_2 + \cdots + x_{n+1} &\leq (r-1) \end{aligned}$$

Par conséquent, dans un $r$-hypergraphe représentant les clauses négatives de la CNF, une clique correspond à une contrainte AtMost-$(r-1)$. □

*Exemple 3.2* – L'hypergraphe de la figure 2 a une clique d'ordre 5 ($\{\overline{x_1}, \overline{x_2}, \overline{x_3}, \overline{x_5}, \overline{x_6}\}$, en bleu), 6 cliques d'ordre 4 ($\{\overline{x_1}, \overline{x_2}, \overline{x_3}, \overline{x_4}\}$, en rose, et les 5 incluses dans la clique d'ordre 5) et 25 cliques d'ordre 3 ($\{\overline{x_5}, \overline{x_6}, \overline{x_7}\}$, en orange, et les 24 incluses dans les cliques d'ordre 4). En ne gardant que les cliques maximales, nous obtenons 3 cliques correspondant aux contraintes suivantes :

$$\begin{aligned} x_1 + x_2 + x_3 + x_4 &\leq 2 \\ x_1 + x_2 + x_3 + x_5 + x_6 &\leq 2 \\ x_5 + x_6 + x_7 &\leq 2 \end{aligned}$$

Il est évident que ce principe fonctionne de manière similaire dans le cas d'une contrainte AtLeast-$k$. Une clause d'arité $k$ $(x_1 \vee x_2 \vee \cdots \vee x_k)$ équivaut à $(x_1 + x_2 + \cdots + x_k \geq 1)$. En d'autres termes, une clause d'arité $k$ $(x_1 \vee x_2 \vee \cdots \vee x_k)$ est équivalente à une contrainte AtLeast-1.

**Proposition 3.2.** *Dans un hypergraphe $r$-uniforme représentant les clauses positives d'une CNF, une clique $\mathcal{C}$ d'ordre $n$ avec $n \geq r$ correspond à une contrainte AtLeast-$(n-r+1)$, autrement dit $\sum_{x \in \mathcal{C}} x \geq n-r+1$.*

### 3.2 Rechercher toutes les cliques maximales

La recherche de cliques maximales est un problème difficile. Le problème CLIQUE est l'un des 21 problèmes complets de Karp et, un graphe à $n$ sommets peut contenir jusqu'à $3^{\frac{n}{3}}$ cliques maximales [21].

Pour traiter la difficulté intrinsèque de notre problème, nous proposons ici de généraliser le fameux algorithme de Bron & Kerbosch (BK) [8] dans le cas des hypergraphes. À notre connaissance, cela n'a pas été fait auparavant. Comme indiqué dans [20], il est généralement supposé que cela ne peut pas être fait efficacement. En effet, la notion de voisin est plus complexe dans les hypergraphes que dans les graphes classiques, et sa généralisation n'est pas simple. Cependant, cela peut être décidé efficacement dans la pratique.

**Algorithme 1 :** Hyper-Bron-Kerbosch

```
1  Hyper-BK (Courant, Candidats, Exclus)
2    si Candidats = ∅ et Exclus = ∅ alors
3      signale Courant est une clique maximale
4    pour chaque sommet v dans Candidats
       faire
5      Courant ← Courant ∪ {v}
6      VoisinC ← γ(Courant)
7      Hyper-BK(Courant, Candidats ∩
        VoisinC, Exclus ∩ VoisinC)
8      Courant ← Courant \ v
9      Candidats ← Candidats \ {v}
10     Exclus ← Exclus ∪ {v}
```

Nous proposons la définition suivante du voisinage de clique afin de généraliser l'algorithme BK [8].

**Définition 3.1** (Voisinage de clique)**.** Soit $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ un ensemble fini de sommets et soit $H = (E_1, E_2, \ldots, E_m)$ un hypergraphe. Pour une clique $\mathcal{C}$ et $\mathcal{Y} \subseteq \mathcal{X} \setminus \mathcal{C}$, nous notons par $\gamma(\mathcal{C})$, le *voisinage de clique* de $\mathcal{C}$, un ensemble de sommets tel que si un sommet est ajouté à $\mathcal{C}$, elle forme une nouvelle clique :

$$\begin{aligned} \gamma(\mathcal{C}) = \{y \in \mathcal{Y} \,|\, \forall N \subseteq \mathcal{C} \text{ avec } |N| \leq r-1, \\ \exists E_i \in H \text{ t.q. } N \cup \{y\} \in E_i\} \end{aligned}$$

Dans cette définition, $\mathcal{Y}$ est introduit pour l'efficacité du calcul. Il peut toujours être défini par $\mathcal{X} \setminus \mathcal{C}$, nous proposons une amélioration à la fin de cette section. À partir de cette définition, nous proposons l'algorithme 1, basé sur l'algorithme BK [8], pour trouver toutes les cliques maximales dans un hypergraphe.

Lors du premier appel, *Courant* et *Exclus* ont pour valeur $\emptyset$, et *Candidats* contient tous les sommets de l'hypergraphe. *Courant* est le résultat temporaire qui est une clique, *Candidats*, l'ensemble des candidats possibles pouvant étendre la clique actuelle pour en former une nouvelle, et *Exclus* contient les sommets à partir desquels des cliques ont déjà été calculées (pour éviter de générer des cliques non maximales).

Il est clair que cet algorithme peut trouver toutes les cliques maximales dans un graphe simple, il n'est pas évident qu'il peut être étendu aussi facilement pour un

hypergraphe. Pour cela, prouvons qu'à chaque étape, *Courant* est une clique et que si elle est signalée, elle est maximale (correction). Prouvons aussi que toutes les cliques maximales sont trouvées (complétude).

**Proposition 3.3.** *À chaque étape de l'algorithme 1, Courant est une clique.*

*Démonstration.*

Un ensemble vide ou contenant un seul sommet est une clique. Supposons que *Courant* est une clique d'ordre $n \geq 2$. Soit *Candidats* est vide et nous revenons à un niveau supérieur où le dernier sommet ajouté est supprimé, et *Courant* reste une clique. Soit *Candidats* contient au moins un sommet $v$. *Candidats* est un sous-ensemble de $VoisinC$, alors par Déf. 3.1, $v$ peut être ajouté à *Courant* de manière à former une nouvelle clique. □

**Proposition 3.4.** *Une clique est signalée par l'algorithme 1, si et seulement si elle est maximale*

*Démonstration.*

Après l'ajout d'un sommet à *Courant*, qui est une clique (Prop. 3.3), si *Candidats* devient vide, alors, soit *Exclus* est vide et *Courant* est maximale car aucun sommet ne peut être ajouté (ligne 3). Ou *Exclus* contient au moins un sommet $v$. *Exclus* est un sous-ensemble de $VoisinC$, alors par Def. 3.1, $v$ peut être ajouté à *Courant* de manière à former une nouvelle clique. Donc la clique n'est pas maximale et Alg. 1 ne la signale pas. Il en va de même si le *Candidats* ne devient pas vide. □

**Proposition 3.5.** *Lorsqu'un candidat $c$ est éliminé dans l'algorithme 1, il ne peut pas étendre Courant.*

*Démonstration.*

Un candidat $c$ est supprimé grâce à l'intersection entre les ensembles *Candidats* et $VoisinC$. Si $c$ n'est pas dans le voisinage de *Courant*, alors par Déf. 3.1, $c$ ne peut pas augmenter *Courant* et est donc éliminé. □

À partir de ces propositions, nous pouvons maintenant proposer le théorème suivant :

**Théorème 3.6.** *L'algorithme 1 termine et trouve toutes les cliques maximales dans un hypergraphe.*

*Démonstration.*

La correction vient directement de Prop. 3.4, l'algorithme 1 ne renvoie que des cliques maximales. La complétude vient de Prop. 3.5, qui garantit que l'ensemble de tous les candidats possibles est testé étant donné qu'un candidat n'est éliminé que s'il ne peut pas faire partie de la clique actuelle. Ainsi, toutes les cliques sont essayées et signalées si elles sont maximales. Enfin, l'algorithme 1 se termine. Sinon, selon le lemme de Kőnig, *Candidats* est infini ou la pile d'appel est infinie. *Candidats* est un sous-ensemble de $VoisinC$ qui est un sous-ensemble de $\mathcal{X}$ qui est fini par Déf. 2.2. Ainsi, *Candidats* ne peut être infini. Comme la pile d'appels dépend de la taille de *Candidats*, elle ne peut pas être infinie. □

Comme indiqué lors de l'introduction de Def. 3.1, l'efficacité du calcul du voisinage de la clique dépend de l'ensemble des sommets considérés $\mathcal{Y}$. Nous proposons un filtrage effectué chaque fois que nous ajoutons un nouveau sommet $v$ dans la clique. Pour tous les candidats $c$, nous savons qu'il nous faut au moins une hyperarête contenant $v$, $c$ et les sous-ensembles de tous les sommets déjà présents dans la clique actuelle. En effet, considérons une clique actuelle d'ordre 4 dans un 3-hypergraphe : $\{x_1, x_2, x_3, x_4\}$, avec $x_4$ étant le dernier élément ajouté. Un sommet $c$ reste candidat si les hyperarêtes suivantes existent $\{x_4, x_1, c\}, \{x_4, x_2, c\}, \{x_4, x_3, c\}$. Bien entendu, beaucoup plus d'hyperarêtes sont nécessaires pour étendre la clique à un ordre $n = 5$. Cependant, si une de ces hyperarêtes n'existent pas, il est certain que $c$ ne peut pas étendre la clique et peut être ignoré.

**Proposition 3.7.** *Soit $x_n$ le dernier sommet ajouté à Courant et prenons $c$ un sommet avec $c \in \mathcal{X} \setminus Courant$. S'il existe de sous-ensemble $\texttt{comb} \subseteq Courant \setminus \{x_n\}$ avec $|\texttt{comb}| \leq r - 2$, de sorte qu'aucune hyperarête ne contienne $\{x_n, \texttt{comb}, c\}$, alors $c \notin Candidats$.*

*Démonstration.*

Directement de la définition 3.1. □

Une façon d'implémenter ce filtrage consiste à utiliser un filtre de Bloom [6]. C'est une structure de données probabiliste qui permet de vérifier si un élément est membre d'un ensemble. Des faux positives sont possibles, mais les faux négatifs ne le sont pas, une requête renvoie "éventuellement dans l'ensemble" ou "définitivement pas dans l'ensemble". Nous utilisons cette structure de données pour filtrer les candidats pour lesquels au moins un hyperarête est manquante, afin qu'ils ne soient pas considérés comme des candidats pour l'extension de la clique. La combinatoire peut être extrêmement volumineuse, il est plus efficace d'avoir des faux positifs à confirmer que de calculer le voisinage de la clique pour tous les sommets.

### 3.3 Cas des clauses mixtes

Dans la section précédente, nous avons émis l'hypothèse que nous avions affaire à des encodages de

$$
\begin{array}{ll}
 & x_1 + x_2 + x_4 + x_5 \leq 1 \\
\equiv & (x_1 + x_2 + x_3 \leq 1) \\
 & \wedge(\overline{x_3} + x_4 + x_5 \leq 1) \\
\equiv & (\overline{x_1} \vee \overline{x_2}) \\
 & (\overline{x_1} \vee \overline{x_3}) \qquad \text{Missing :} \\
 & (\overline{x_2} \vee \overline{x_3}) \qquad (\overline{x_1} \vee \overline{x_4}) \\
 & (x_3 \vee \overline{x_4}) \qquad (\overline{x_1} \vee \overline{x_5}) \\
 & (x_3 \vee \overline{x_5}) \qquad (\overline{x_2} \vee \overline{x_4}) \\
 & (\overline{x_4} \vee \overline{x_5}) \qquad (\overline{x_2} \vee \overline{x_5})
\end{array}
$$

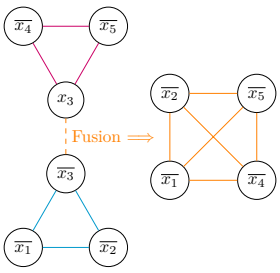



**Figure 3** – Détection de clique avec clauses mixtes

cardinalité naïf en CNF : nous n'avions envisagé que des encodages avec des clauses uniformes. Cependant, cette hypothèse est clairement trop forte pour la recherche de contraintes de cardinalité dans une preuve ou simplement dans des cas plus complexes. Nous appelons *mixte* une clause faisant partie d'un encodage de contraintes de cardinalité contenant des littéraux positifs et négatifs. Pour expliquer comment nous pouvons contourner ce nouveau problème de détection, illustrons-le avec un exemple.

Dans la configuration de la figure 3, nous voyons qu'une recherche syntaxique simple échouera probablement à détecter une clique. Cependant, comme le montre la représentation graphique de ces clauses, nous pouvons *fusionner* les deux petites cliques pour en créer une plus grande. Ce principe s'appelle Merge AtMost-1 dans [4]. Nous proposons maintenant de généraliser ce résultat précédent aux contraintes AtMost-$k$ et AtLeast-$k$. Tout d'abord, nous devons déterminer si notre approche peut récupérer des encodages *imbriqués*, qui sont probablement les encodages les plus courants pour les contraintes de cardinalité.

*Remarque 3.1* – Toute contrainte de cardinalité $(x_1 + x_2 + \cdots + x_n \diamond k)$ avec $\diamond \in \{\leq, \geq, =\}$ en encodage imbriqué peut être retrouvée en fusionnant les littéraux utilisés pour scinder la contrainte d'origine, *c.-à-d.* en fusionnant des littéraux qui apparaissent positivement et négativement dans chaque paire de contraintes.

$$
\begin{array}{ll}
 & x_1 + x_2 + \cdots + x_n \diamond k \\
\equiv & (x_1 + \cdots + x_n + y_1 + \cdots + y_m \diamond c_1) \\
 & \wedge(\overline{y_1} + \cdots + \overline{y_m} \diamond c_2) \\
\equiv & \ldots \\
\equiv & (x_1 + y_1 + \cdots + y_m \diamond c_1) \wedge (x_2 + \overline{y_1} \diamond c_2) \wedge \ldots \\
 & \wedge(x_n + \overline{y_m} \diamond c_n)
\end{array}
$$

avec $\sum_i c_i = k + m$

Tout cela fonctionne bien lorsque la CNF (et la preuve) est bien formée, *c.-à-d.* qu'elle ne contient que des clauses de même taille, bien adaptées pour retrouver une clique. Cependant, cela ne suffit pas pour couvrir tous les cas, en particulier lors de la recherche dans les preuves, où les clauses unitaires apparaissent souvent. L'idée est de les utiliser comme des hyperarêtes spéciales afin de détecter de plus grandes cliques, comme ce qui a été fait dans la section 4.1 dans [4].

*Remarque 3.2* – Toute clause de taille $k$ est une clause de taille $k'$ avec $1 \leq k \leq k'$. Cette clause peut être obtenue en introduisant autant de $\perp$ littéraux que nécessaire, *c.-à-d.* en considérant $C' = (C \vee \perp \vee \cdots \vee \perp)$, avec $C'$ de taille $k'$.

En effet, comme expliqué dans la remarque 3.2, une clause plus petite peut être considérée comme une clause plus grande, un nœud spécial étant le nœud $\perp$. Lorsqu'un littéral l apparaît dans une clause unitaire dans la preuve, il peut être ajouté à toute arête, car $F \wedge l$ est égal à $F \wedge l \wedge (l \vee x)$ pour tout $x$.

### 3.4 à la recherche de meilleures explications

Dans notre quête d'explication la plus simple possible des preuves d'insatisfiabilité, il n'est pas suffisant de trouver autant de contraintes de cardinalité que possible. Nous devons trouver les contraintes les plus fortes. Illustrons ce problème avec, encore une fois, le problème bien connu des pigeonniers.

*Exemple 3.3* – Voici les contraintes que nous pouvons extraire en recherchant des cliques dans l'exemple 2.1.

$$
\begin{array}{rr}
x_1 + x_4 + x_7 + x_{10} \leq 1 & x_1 + x_2 + x_3 \geq 1 \\
x_2 + x_5 + x_8 + x_{11} \leq 1 & x_4 + x_5 + x_6 \geq 1 \\
x_3 + x_6 + x_9 + x_{12} \leq 1 & x_7 + x_8 + x_9 \geq 1 \\
 & x_{10} + x_{11} + x_{12} \geq 1
\end{array}
$$

Le problème est insatisfiable, mais l'explication reste floue. Ce que nous aimerions idéalement avoir, c'est : $(\sum_{i=1}^{12} x_i \leq 3)$ et $(\sum_{i=1}^{12} x_i \geq 4)$.

Obtenir de telles contraintes structurées comme expliqué dans l'exemple 3.3 est assez simple, il suffit de faire la somme des AtLeast-$k$ d'un côté et la somme des AtMost-$k$ de l'autre.

Nous obtiendrons des contraintes de la forme $(\sum_{i=1}^{n} x_i \leq k)$ and $(\sum_{i=1}^{n} x_i \geq k')$. Si nous obtenons $k = k'$, nous pouvons remplacer ces deux contraintes par $(\sum_{i=1}^{n} x_i = k)$ qui est plus fort. Sinon, si nous obtenons $k' > k$, nous pouvons alors supprimer toutes les autres contraintes et ne garder que ces deux contraintes, car elles suffisent à expliquer l'insatisfaisabilité. Sinon, si nous obtenons $k < k'$, alors tout est en ordre et nous

ne pouvons rien faire d'autre que de stocker les résultats finaux. De toutes ces extractions et simplifications, nous pouvons obtenir la définition suivante qui sera utilisée dans la partie expérimentale.

**Définition 3.2** (Explication-Cardinalité)**.** Une preuve DRAT est dite *cardinalité-expliquée*, si on obtient l'ensemble de contraintes suivant : $(\sum_{x \in C} x \leq Y)$ et $(\sum_{x \in C} x \geq Y')$ avec $Y > Y'$.

Cependant, dans le cas général, il ne suffit de faire la somme de toutes les contraintes jusqu'à ce que l'on obtienne une cardinalité-explication. Pour contourner ce problème, nous avons proposé une recherche gloutonne à travers les différentes contraintes afin d'en obtenir de plus fortes. Malheureusement, étant une heuristique, certaines instances pourraient être cardinalité-expliquées, mais ne le seront pas en raison de l'ordre choisi pour additionner les contraintes.

Nous avons deux types de critères pour choisir les contraintes à additionner. Premièrement, la diversification, si deux contraintes n'ont pas de variables en commun, elles peuvent être ajoutées en toute sécurité. Le deuxième critère est l'intensification. Deux contraintes qui ont exactement les mêmes variables peuvent également être additionnées ainsi que leurs bornes. Puis la borne est divisée par deux et arrondie, afin d'éviter un coefficient devant les variables de la contrainte. Nous effectuons cette recherche à la fois dans AtMost-$k$ et AtLeast-$k$ afin d'obtenir une cardinalité-explication ou une contrainte Equals-$k$, afin de supprimer en toute sécurité deux contraintes. En effet, si on obtient $(\sum_i x_i \leq k)$ et $(\sum_i x_i \geq k)$, on peut supprimer les deux contraintes et ajouter la contrainte suivante : $(\sum_i x_i = k)$.

## 4 Étude expérimentale

Nous avons implémenté l'approche proposée dans un solveur open source (écrit en C ++) `EXPResSO`, signifiant*EXplainable PRoof SOlver*. La structure de données pour manipuler l'hypergraphe est basée sur la bibliothèque HTD [1].

Le but de cette section d'évaluation est double. (1) nous devons évaluer les performances de notre méthode et (2) vérifier combien de contraintes de cardinalité nous pouvons trouver et combien nous pouvons compresser les preuves d'insatisfaisabilité en utilisant des contraintes de cardinalité. Toutes les preuves étudiées sont des preuves DRAT données par `Glucose` [2] et certifiées par DRAT-trim [26]. Les contraintes de cardinalité trouvées dans la formule sont exprimées sous le format XCSP3 [7], qui est le format standard pour les compétitions de programmation par contraintes (CP).

`Nacre` [11] 1.0.4 garantie l'insatisfiabilité de l'instance XCSP3. Lorsque la preuve est réécrite en tant qu'instance XCSP3, `Nacre` trouve toujours la contradiction par un simple contrôle d'arc cohérence (pas de recherche). `Glucose` et `EXPResSO` fonctionnent sur un cluster d'ordinateurs identiques dotés de deux processeurs Intel (R) E5-2670, 2,60 GHz avec 64 Go de mémoire. Pour chaque problème, nous fixons un timeout de 900 secondes et une limite mémoire de 32 Go.

### 4.1 Contraintes de cardinalité dans les preuves unsatisfiable des formules aléatoires

Les premiers résultats que nous présentons sont probablement les plus surprenants. Nous considérons ici des formules aléatoires uniformes (au seuil) tirées de SATLIB. Les résultats montrent clairement un nombre impressionnant de contraintes de cardinalité trouvées dans les preuves UNSAT d'instances aléatoires. La figure 5 illustre le (surprenant) gain. On peut s'attendre à un gain exponentiel sur ces preuves (la diagonale signifie aucun gain).

La figure 4 montre les fréquences de chaque $\leq k$ (resp. $\geq k$). Comme on le voit, on est loin de trouver que des contraintes de cardinalité triviales. Les cliques deviennent même de plus en plus grandes avec la taille des instances. C'est un résultat surprenant qui pourrait probablement s'expliquer par le fait que les solveurs SAT ont tendance à générer des preuves de plus en plus grandes avec le même ensemble de variables, permettant ainsi de trouver plus de cliques. Cependant, nous pouvons aussi raisonnablement supposer que le nombre important de contraintes de cardinalité témoigne de la stratégie de recherche particulière qui émerge de la recherche même du solveur CDCL, même si cette hypothèse mérite de nouvelles investigations.

Concentrons-nous maintenant sur la figure 6. Rechercher toutes les cliques maximales et ensuite fusionner toutes les contraintes obtenues a malheureusement un coût élevé. `Glucose` est capable de produire une preuve pour toutes les instances en moins de 10 secondes, alors que `EXPResSO` peut prendre des centaines de secondes pour rechercher toutes les cliques maximales. La qualité des explications trouvées dans la preuve DRAT vaut toutefois la peine. Par exemple, une autre expérimentation a indiqué que plus de 27% de toutes les preuves UNSAT contiennent deux contraintes contradictoires de cardinalité. Bien sûr, encore une fois, ces contraintes peuvent être d'un grand intérêt pour comprendre la structure de l'instance originale ou même la forme de la recherche elle-même.

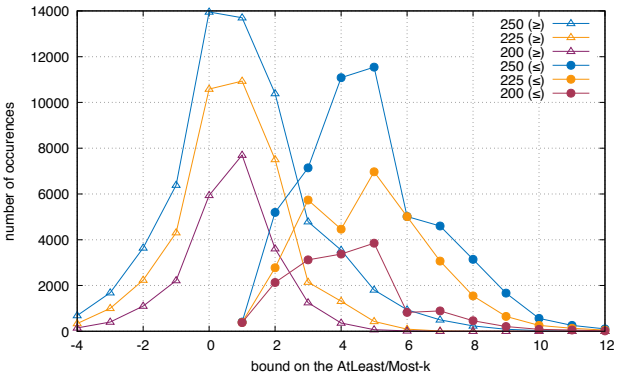



**Figure 4** – Instances aléatoires : fréquences des contraintes de cardinalité pour $\leq k$ et $\geq k$

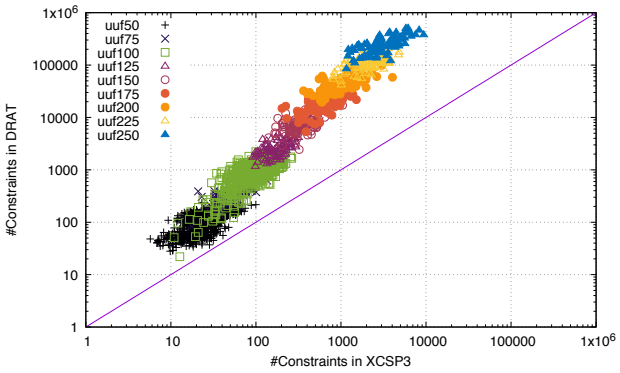



**Figure 5** – Instances aléatoires : taille des preuves en CNF vs nb. des contraintes de cardinalité

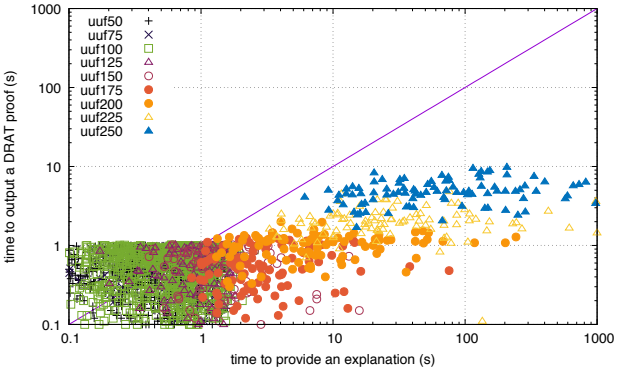



**Figure 6** – Instances aléatoires : temps (s) pour `Glucose` et `EXPResSO`

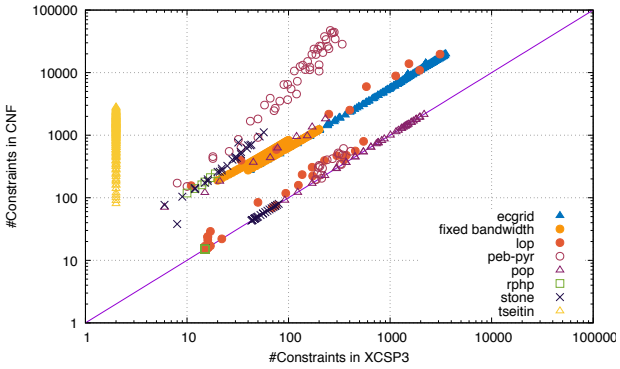



**Figure 7** – Instances crafted : taille des problèmes d'entrée vs nb. de contraintes de cardinalité

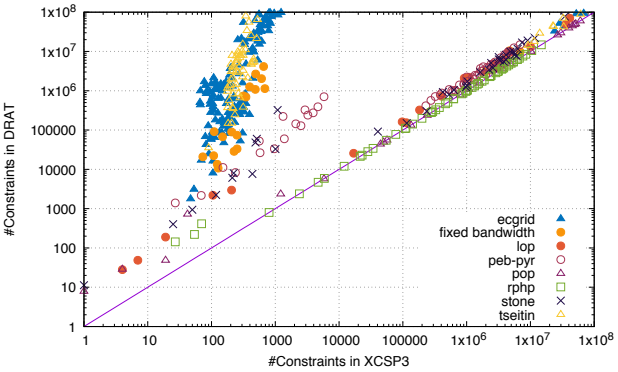



**Figure 8** – Instances crafted : taille de la preuve (DRAT) vs nb. des contraintes de cardinalité

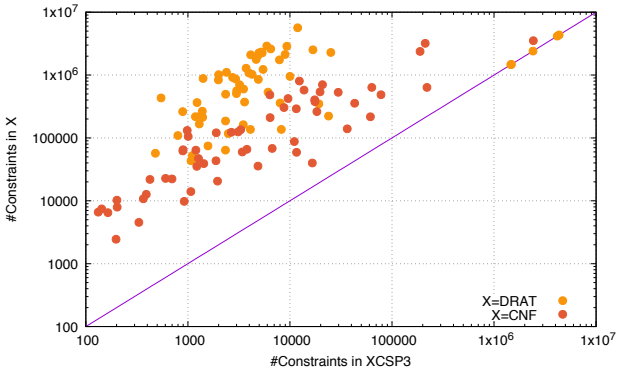



**Figure 9** – Instances Industrielles : taille de la CNF (resp. DRAT) vs nb. des contraintes de cardinalité

| Formula | Input | | DRAT | XCSP3 |
|---|---|---|---|---|
| | #vars | #clauses | #size | #const |
| `hole20` | 420 | 4 221 | 50 749 | 2 |
| `hole30` | 930 | 13 981 | 171 274 | 2 |
| `hole40` | 1640 | 32 841 | 405 899 | – |
| `hole50` | 2550 | 63 801 | 792 624 | – |
| `tph8` | 136 | 5 457 | 86 164 | 2 |
| `tph12` | 300 | 27 625 | 439 462 | 2 |
| `tph16` | 528 | 87 329 | 1 392 616 | – |
| `tph20` | 820 | 213 241 | 3 404 250 | – |
| `Urquhart-b1` | 106 | 714 | 179 330 | 1 |
| `Urquhart-b2` | 107 | 742 | 165 638 | 1 |
| `Urquhart-b3` | 121 | 1 116 | 299 910 | 1 |
| `Urquhart-b4` | 114 | 888 | 227 774 | 1 |

**Table 1** – Comparaison des tailles entre la formule d'ori-

| | `Glucose` | | | `EXPResSO` | | |
|---|---|---|---|---|---|---|
| | min | med | max | min | med | max |
| Original | 2448 | 66896 | 3520181 | 103 | 9598 | 3520181 |
| Proof | 33902 | 859211 | 4578043 | 511 | 10231 | 4578043 |

**Table 2** – Instances Industrielles : analyse des tailles entre l'instance, la preuve et l'explication fournie

### 4.2 Contraintes de cardinalité dans les preuves de problèmes crafted

Les problèmes que nous considérons maintenant sont généralement générés pour donner du fil à retordre aux solveurs SAT basés sur la résolution. Nous avons utilisé dans cette section une sélection de problèmes décrits plus en détail dans [9, 19]. Commentons d'abord la Table 1, qui résume les résultats obtenus sur les preuves des problèmes de [19]. Les résultats, reportés dans la Table 1, sont vraiment frappants : pour les familles `hole*` et `tph*`, quand le délai imparti le permet, nous avons pu extraire une preuve de 2 contraintes. Notez que le post-traitement s'est fait instantanément, en résumant simplement toutes les contraintes afin d'obtenir une *explication-cardinalité* pour `hole*`, (nous obtenons $\sum_{i=1}^{n} x_i \leq 19$ et $\sum_{i=1}^{n} x_i \geq 20$. pour hole20) et `tph*` (contraintes similaires), une fois que nous avons éliminé par post-traitement les autres contraintes inutiles pour expliquer l'insatisfiabilité. Notez que nous ne nous sommes pas comparés avec la résolution étendue (ER) [25] ni DPR (une généralisation de DRAT) [15] car notre objectif ici n'est pas d'obtenir un système de preuve complet, mais simplement de traiter la preuve DRAT afin d'obtenir une explication plus courte de la réfutation. Maintenant, sur les instances proposées dans [9]. Les résultats sont représentés sur les figures 7 et 8. Ici encore, nous observons une grande compression sur les problèmes initiaux et aussi sur les preuves lorsque `EXPResSO` est capable de les gérer (les points sur la diagonale signifient que notre outil n'a pas été en mesure de réécrire les contraintes). Ce que nous pouvons voir, c'est que ces instances, qui sont difficiles à résoudre, génèrent des preuves qui peuvent en fait être réduites à très peu de contraintes de cardinalité.

### 4.3 Contraintes de cardinalité dans les preuves des problèmes industriels

Dans cette dernière série d'instances [10], nous nous concentrons sur les instances "industrielles" typiques. Ces instances ont été sélectionnées de telle sorte que leur preuve soient de taille raisonnable. Une fois de plus, nous étudions comment notre outil peut obtenir des contraintes de cardinalité dans les formules originales et dans les preuves. Les résultats sont résumés dans la Table 2 ainsi que sur la figure 9. Trois instances ont produit une preuve trop grande pour notre outil : leur nombre de contraintes de cardinalité correspond donc exactement au nombre de clauses (voir le trois points sur la diagonale de la figure). Cette dernière expérimentation conclut cette section en montrant à quel point notre méthode est générique : les contraintes de cardinalité sont légion dans toutes les preuves expérimentées, quelles que soient les origines des problèmes.

## 5 Conclusion

Dans cet article, nous avons montré que les preuves DRAT peuvent être restructurées et exprimées sous la forme de contraintes de cardinalité, même dans le cas d'instances générées aléatoirement. Pour ce faire, nous encodons la preuve sous la forme d'un hypergraphe où chaque lemme est une hyperarête et chaque littéral est un sommet. Ensuite nous cherchons toutes les cliques maximales, codant les contraintes de cardinalité. Nous avons généralisé l'algorithme de Bron & Kerbosch, pour énumérer les cliques maximales d'un hypergraphe et démontré son efficacité et sa *scalabilité* avec des centaines de sommets et des millions d'hyperarêtes.

Dans la recherche de preuves simples et efficaces, la communauté tend à proposer des systèmes de preuves strictement plus forts avec de nouvelles stratégies de recherche. Cependant, cela a un coût important : les procédures de recherche sont soit conçues pour des problèmes particuliers, soit moins génériques que les solveurs CDCL actuels. Ici, nous adoptons le point de vue opposé : nous gardons les performances des solveurs SAT et proposons de rechercher dans la trace des régularités exploitables. Ce travail est le premier à présenter une structure particulière et compréhensible des preuves de solveur SAT.

Cependant, notre approche a aussi des inconvénients : pour que la preuve soit expliquée, il faut d'abord la trouver, ce qui peut avoir un coût exponentiel (nous ne pouvons pas aller au-delà des limites de la résolution). La prochaine étape consisterait à piloter la recherche du solveur SAT grâce à un solveur de théorie cherchant des cliques maximales dans la preuve courante. Si une partie d'une explication de cardinalité est trouvée, il pourrait être intéressant de forcer la recherche vers le cas "opposé", ce qui permettrait de produire une preuve beaucoup plus courte ou de diviser la recherche en deux. Nous conjecturons également la présence de contraintes de cardinalité importantes dans la preuve d'insatisfaisabilité comme l'un des principaux témoins de la recherche du solveur SAT. Une grande hyperclique centrale pourrait expliquer la grande non-dégénérescence des graphes induits par la preuve rapportée dans [10]. Une autre étape consisterait à fournir un format certifié pouvant être indépendant vérifié avec un `checker`.

## Remerciements

Une partie de ces travaux ont étés soutenues par le Ministère de l'Enseignement Supérieur et de la Recherche et partiellement financés par l'IDEX UCA[JEDI].

## Références

[1] Michael Abseher, Nysret Musliu et Stefan Woltran : htd - A free, open-source framework for (customized) tree decompositions and beyond. *In Proc. of CPAIOR*, pages 376–386, 2017.

[2] Gilles Audemard et Laurent Simon : Predicting Learnt Clauses Quality in Modern SAT Solvers. *In Proc. of IJCAI*, pages 399–404, 2009.

[3] Claude Berge : *Hypergraphs : Combinatorics of Finite Sets*, volume 45 de *Mathematical Library*. Elsevier, 05 1984.

[4] Armin Biere, Daniel Le Berre, Emmanuel Lonca et Norbert Manthey : Detecting Cardinality Constraints in CNF. *In Proc. of SAT*, pages 285–301, 2014.

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

[6] Burton H. Bloom : Space/Time Trade-offs in Hash Coding with Allowable Errors. *Commun. ACM*, 13(7):422–426, 1970.

[7] Frédéric Boussemart, Christophe Lecoutre et Cédric Piette : XCSP3 : An Integrated Format for Benchmarking Combinatorial Constrained Problems. *CoRR*, abs/1611.03398, 2016.

[8] Coen Bron et Joep Kerbosch : Algorithm 457 : Finding All Cliques of an Undirected Graph. *Commun. ACM*, 16(9):575–577, 1973.

[9] Jan Elffers, Jesús Giráldez-Cru, Stephan Gocht, Jakob Nordström et Laurent Simon : Seeking practical CDCL insights from theoretical SAT benchmarks. *In Proc. of IJCAI*, pages 1300–1308, 2018.

[10] Rohan Fossé et Laurent Simon : On the Non-degeneracy of Unsatisfiability Proof Graphs Produced by SAT Solvers. *In Proc. of CP*, pages 128–143, 2018.

[11] Gael Glorian : Nacre. *In* Christophe Lecoutre et Olivier Roussel, éditeurs : *Proc. of 2018 XCSP3 Competition*, pages 85–85, 2019.

[12] Éric Grégoire, Richard Ostrowski, Bertrand Mazure et Lakhdar Sais : Automatic Extraction of Functional Dependencies. *In Proc. of SAT*, 2004.

[13] Marijn Heule, Warren A. Hunt Jr. et Nathan Wetzler : Trimming while checking clausal proofs. *In Proc. of FMCAD*, pages 181–188, 2013.

[14] Marijn J. H. Heule : Schur Number Five. *In Proc. of AAAI*, pages 6598–6606, 2018.

[15] Marijn J. H. Heule, Benjamin Kiesl et Armin Biere : Short Proofs Without New Variables. *In Proc. of CADE*, pages 130–147, 2017.

[16] Marijn J. H. Heule, Oliver Kullmann et Victor W. Marek : Solving and Verifying the Boolean Pythagorean Triples Problem via Cube-and-Conquer. *In Proc. of SAT*, pages 228–245, 2016.

[17] Matti Järvisalo, Marijn Heule et Armin Biere : Inprocessing Rules. *In Proc. of IJCAR*, pages 355–370, 2012.

[18] Matti Järvisalo, Arie Matsliah, Jakob Nordström et Stanislav Zivny : Relating Proof Complexity Measures and Practical Hardness of SAT. *In Proc. of CP*, pages 316–331, 2012.

[19] Benjamin Kiesl, Adrián Rebola-Pardo et Marijn J. H. Heule : Extended Resolution Simulates DRAT. *In Proc. of IJCAR*, pages 516–531, 2018.

[20] Andreas Koeller : *Integration of Heterogeneous Databases : Discovery of Meta-Information and Maintenance of Schema-Restructuring Views*. Thèse de doctorat, Worcester Polytechnic Institute, dec 2001.

[21] J. W. Moon et L. Moser : On cliques in graphs. *Israel Journal of Mathematics*, 3(1):23–28, 1965.

[22] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang et Sharad Malik : Chaff : Engineering an Efficient SAT Solver. *In Proc. of DAC*, pages 530–535, 2001.

[23] Richard Ostrowski, Éric Grégoire, Bertrand Mazure et Lakhdar Sais : Recovering and exploiting structural knowledge from CNF formulas. *In Proc. of CP*, pages 185–199, 2002.

[24] Laurent Simon : Post Mortem Analysis of SAT Solver Proofs. *In Proc. of POS*, pages 26–40, 2014.

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

[26] Nathan Wetzler, Marijn Heule et Warren A. Hunt Jr. : DRAT-trim : Efficient Checking and Trimming Using Expressive Clausal Proofs. *In Proc. of SAT*, pages 422–429, 2014.

# Utilisation d'algorithmes d'approximation en Programmation Par Contraintes

**Arthur Godet**[1*] **Xavier Lorca**[2] **Gilles Simonin**[1]

[1] Equipe TASC, IMT Atlantique, LS2N, CNRS, 44307, Nantes, France
[2] Centre Génie Industriel, IMT Mines Albi, Campus Jarlard, 81013 Albi cedex 09, France
{arthur.godet,gilles.simonin}@imt-atlantique.fr xavier.lorca@mines-albi.fr

**Résumé**

Dans cet article, nous présenterons les travaux préliminaires menés sur l'utilisation d'algorithmes d'approximation en Programmation Par Contraintes afin d'améliorer le calcul de bornes lors de la résolution de problèmes d'optimisation sous contraintes. L'objectif de nos travaux est d'étudier plus particulièrement quels algorithmes d'approximation présentent suffisamment de flexibilité pour être utilisés en Programmation Par Contraintes, et comment les utiliser au sein d'un propagateur qui mettra à jour les bornes de la variable-objectif à chaque noeud de l'espace de recherche. Enfin l'idée sera d'appliquer cette approche à plusieurs familles de problèmes d'optimisation afin d'en extraire une généralisation.

**Abstract**

In this paper we present our current work on using approximation algorithms in constraint programming to improve bounds computing when solving Constraint Optimization Problems. The objective of our work is to study more particularly which approximation algorithms have enough flexibility to be used in constraint programming, and how to use them inside a propagator which will update the bounds of the objective-variable at each node of the search space. At last, we want to apply this methodology to several families of optimisation problems in order to obtain a generalisation.

## 1 Introduction

La **Programmation Par Contraintes (PPC)** tire sa force de son expressivité et de sa flexibilité. L'une par sa capacité à exprimer des sous-problèmes pertinents d'un problème donné au travers de contraintes indépendantes les unes des autres. L'autre par sa capacité à décomposer un problème combinatoire en sous-problèmes pour lesquels une résolution efficace est possible. Elle se distingue ainsi des autres technologies de résolution de problèmes combinatoires, telles que la Programmation Linéaire ou la Programmation Mixte en nombre entiers, par la richesse et la variété des contraintes exprimables, qui dépasse par exemple le cadre de simples équations linéaires. Cette richesse et cette diversité s'expriment typiquement par l'utilisation de **contraintes globales**, c'est-à-dire de contraintes encapsulant des sous-problèmes combinatoires pour lesquels des traitements algorithmiques efficaces sont connus. Ceci permet généralement un meilleur *filtrage*. Toutefois, et malgré de nombreuses applications industrielles, la PPC souffre d'un certain manque d'efficacité en ce qui concerne la résolution de **Problèmes d'Optimisation sous Contraintes (COP)**.

D'un autre côté, les **Algorithmes d'Approximation** sont des algorithmes polynomiaux en temps et en espace résolvant de manière approchée des problèmes $\mathcal{NP}$-difficile avec une certaine garantie de résultat par rapport à l'optimal.

**Definition 1.** *Un **algorithme d'approximation** est un algorithme polynomial, en temps et en espace, résolvant un problème de minimisation (resp. maximisation) sous contraintes avec une garantie de résultat. Ainsi, si l'on note $\rho$ le rapport d'approximation d'un algorithme d'approximation, ce dernier renverra, pour toute instance du problème, une solution dont l'évaluation ne sera pas plus grande (resp. petite) que $\rho \times opt$, où $opt$ représente l'évaluation d'une solution optimale de l'instance du problème.*

Pour chaque problème bien connu, les recherches menées tournent autour de l'élaboration d'algorithmes présentant de meilleurs ratios d'approximation, ou de prouver qu'il n'est pas possible de faire mieux qu'un cer-

*Papier doctorant : Arthur Godet[1] est auteur principal.

tain ratio. Ainsi, pour un même problème $\mathcal{NP}$-difficile, il existe souvent différents algorithmes d'approximation le résolvant de manière non optimale. Par exemple, pour le problème $\mathcal{NP}$-difficile du *Bin Packing* [4], plusieurs algorithmes d'approximation sont connus depuis longtemps, tels que le *First-Fit* et le *Best-Fit* [5]. Toutefois, les recherches menées ont permis de mettre au point des algorithmes avec de meilleures garanties [3]. L'intérêt de s'orienter vers la théorie de l'approximation pour améliorer la PPC réside en sa forte communauté, active depuis plusieurs décennies, ayant ainsi produit de nombreux résultats sur lesquels s'appuyer.

Dans ce papier, nous présentons les travaux que nous menons pour utiliser les résultats issus de l'étude d'algorithmes d'approximation en Programmation Par Contraintes afin d'améliorer l'efficacité de résolution de COP. Dans une première section, nous discuterons des difficultés d'application des algorithmes d'approximation dans le cadre de la PPC et présenterons un algorithme générique - illustrant notre méthodologie - qui peut être embarqué au sein d'un *propagateur*. Dans une seconde section, nous présenterons les premiers résultats expérimentaux que nous avons obtenus.

## 2 Méthodologie

La Programmation Par Contraintes présente une forte flexibilité quant aux contraintes que l'on peut poser, ainsi qu'aux techniques de résolution déployées au sein des solveurs de contraintes. Cette flexibilité de résolution est exactement là où réside la difficulté d'utilisation d'algorithmes d'approximation en PPC. En effet, les algorithmes d'approximation ont souvent un comportement très déterministe, ne laissant pas de place à des choix arbitraires. Bien que ce cadre déterministe soit souvent à la base des démonstrations des ratios d'approximation, il s'avère très limitant en Programmation Par Contraintes. En effet, les heuristiques de sélection de la variable sur laquelle brancher et sur quelle valeur brancher peuvent amener à une situation ne correspondant pas au cadre d'application de l'algorithme d'approximation. C'est-à-dire que les instantations déjà effectuées ne correspondent pas, telles quelles, à une situation intermédiaire dans laquelle l'algorithme d'approximation aurait pu se retrouver à partir de l'instance initiale et sous certaines règles de choix arbitraires.

### 2.1 Exemple montrant cette limite

Pour expliquer cette limite, nous nous baserons sur le problème de minimisation du *Bin Packing*. Considérons une liste de $n$ objets, chaque objet $i$ possédant un poids $w_i$. Nous disposons également de $n$ conteneurs homogènes de capacité $B$. L'objectif est de placer l'ensemble des objets dans le plus petit nombre possible de conteneurs.

Pour résoudre ce problème, comme nous l'avons dit, il existe de nombreux algorithmes d'approximation, le plus connu d'entre eux étant l'algorithme dit du *First-Fit Decreasing (FFD)*. Cet algorithme consiste à trier les objets par ordre décroissant de poids, puis de les placer chacun leur tour dans le premier conteneur pouvant le contenir.

Une modélisation possible de ce problème en Programmation Par Contraintes consiste en $n$ variables entières $x_1$, ..., $x_n$, chacune de domaine $[1, n]$ indiquant le conteneur auquel est associé l'objet. On pose finalement la contrainte globale *bin_packing* [7, 2]. Dans le cas du problème de minimisation, on utilise également une variable $z$ de domaine $[1, n]$ indiquant le nombre de conteneurs utilisés à l'aide de la contrainte $nvalue(z, \{x_1, ..., x_n\})$ [6] (une contrainte $max(z, \{x_1, ..., x_n\})$ peut être utilisée pour casser les symétries en l'absence de contraintes supplémentaires). On cherche donc une solution minimisant la valeur de $z$.

Supposons, sans perdre en généralité, que les variables $x_1$, ..., $x_n$ sont déjà triées par ordre décroissant de poids. N'importe quelle situation telle qu'une variable $x_j$ est instanciée sans qu'une variable $x_i$ ne le soit (avec $i < j$) ne correspond à aucune étape intermédiaire dans laquelle l'algorithme du *First-Fit Decreasing* aurait pu se retrouver, cet algorithme plaçant les objets par ordre décroissant de poids dans le premier conteneur pouvant le contenir. Cette situation se produirait systématiquement si l'heuristique de sélection de variable consiste en un ordre lexicographique inversé.

Notons qu'il peut être possible dans certains cas de se ramener à un cas d'application de l'algorithme, moyennant quelques "transformations" de l'instance sur laquelle on applique l'algorithme d'approximation. Ce point sera discuté dans la prochaine sous-partie, et illustré en conservant notre exemple.

### 2.2 Algorithme générique

En considérant un problème de minimisation sous contraintes (un problème de maximisation sous contraintes se traiterait de manière analogue) ainsi qu'un algorithme d'approximation **Heur** de ratio d'approximation $\rho > 1$, le propagateur que l'on ajoute est décrit par l'algorithme 1 :

Tout d'abord, le propagateur teste si l'algorithme d'approximation est applicable à partir de l'état actuel des variables. Bien entendu, la fonction **detectIfAlgoApplicable()** dépend de l'algorithme d'approximation sur lequel est basé le propagateur. Cette fonction

**Entrées** : $z$ the objective-variable, $x_i$ variables
**début**
  **si** *detectIfAlgoApplicable()* **alors**
    $inst \leftarrow$ buildCorrespondingInstance();
    $sol \leftarrow$ Heur($inst$);
    $z$.updateUpperBound($sol$);
    $z$.updateLowerBound($sol/\rho$);
  **fin**
**fin**

**Algorithme 1 :** APX-Propagator

renvoie donc s'il est possible, à partir de l'état actuel des variables, de créer une instance du problème sur laquelle appliquer l'algorithme d'approximation. Si les règles d'application de l'algorithme d'approximation ne permettent pas d'interdire des valeurs et que cela a un impact important, cette fonction devra en tenir compte.

La fonction **buildCorrespondingInstance()** construit l'instance correspondant à l'état actuel des variables et sur laquelle sera appliquée l'algorithme d'approximation. Cette fonction dépend également de l'algorithme d'approximation sur lequel est basé le propagateur.

Finalement, le propagateur applique l'algorithme d'approximation sur l'instance construite et met à jour les bornes de la variable-objectif $z$ à partir de la valeur renvoyée par l'algorithme d'approximation. Dans le cas d'un problème de minimisation, l'algorithme d'approximation renvoie une valeur comprise entre $opt$ et $\rho \times opt$. Ainsi on met à jour la borne inférieure de la variable-objectif par le maximum entre sa valeur actuelle et le résultat de l'algorithme d'approximation divisé par son rapport d'approximation. La borne supérieure est mise à jour par le minimum entre sa valeur actuelle et le résultat de l'algorithme d'approximation (plus grand ou égal à la valeur optimale dans le cas d'un problème de minimisation).

Il est intéressant de noter que les bornes calculées par ce propagateur sont valables dans tout l'espace de recherche.

À ce stade de notre étude, nous supposons qu'il existe un lien entre le nombre de situations applicables de l'algorithme d'approximation et l'impact, en terme de filtrage sur la variable-objectif, du propagateur dérivé de cet algorithme.

Ainsi, il pourrait être plus intéressant d'utiliser un algorithme d'approximation avec un moins bon ratio d'approximation s'il est davantage applicable qu'un algorithme d'approximation avec un meilleur ratio d'approximation mais des règles d'application plus restrictives. Dans le cas du problème de *Bin Packing*, il paraît évident que l'algorithme du *First-Fit*, ayant un rapport d'approximation de $\frac{17}{10} \times S^{OPT} + 2$ [5], est applicable dans davantage de situations que l'algorithme du *First-Fit Decreasing*, de rapport d'approximation $\frac{11}{9} \times S^{OPT} + 4$ [5]. L'algorithme du *First-Fit* a donc un ratio d'approximation de moins bonne qualité que celui du *First-Fit Decreasing*, mais le *First-Fit* étant applicable dans davantage de cas, on peut suspecter qu'il aura plus d'impact sur l'évaluation de la borne inférieure de la variable-objectif (ici le nombre de conteneurs utilisés). Le fait qu'il soit applicable dans plus de situations laisse également espérer qu'il pourra aussi mieux évaluer une borne supérieure.

## 3 Méthodologie expérimentale

Au moment de la rédaction de cet article, nous travaillons encore à l'élaboration et l'exécution de protocoles expérimentaux permettant d'observer les effets apportés par un tel propagateur. L'objectif est de comparer les statistiques de résolution (temps de résolution, nombre de noeuds parcourus, nombre de backtracks, nombre de fails) obtenues par un modèle et celles obtenues par ce même modèle auquel on aura ajouté notre propagateur. On observera en particulier le nombre d'appels à notre propagateur au cours de la recherche d'une solution optimale, ainsi que les effets qu'il aura eus.

L'idée est également de comparer les apports de différents algorithmes d'approximation, cherchant notamment à établir une idée du compromis qu'il peut y avoir entre rapport d'approximation et cas d'application. Pour cela, on construira deux modèles, l'un utilisant notre propagateur avec un algorithme d'approximation $A$, l'autre utilisant notre propagateur avec un algorithme d'approximation $B$. On comparera alors les statistiques de résolution des deux modèles, et l'on comparera plus particulièrement le nombre d'appels effectifs à notre propagateur dans chacun des deux modèles et les effets que chacun aura eu.

Finalement, l'objectif sera d'essayer d'appliquer cette méthodologie à différents problèmes, et différentes familles de problèmes, et d'essayer de faire émerger une généricité de l'approche.

| | Temps (s) | Noeuds | Erreurs |
|---|---|---|---|
| BFD-Prop | 17.36 | 177417 | 177340 |
| Shaw | 503.6 | 12147504 | 12147425 |
| Binary model | 379.5 | 14047374 | 14047295 |

FIGURE 1 – Résultats expérimentaux préliminaires pour le problème du Bin-Packing

Bien que la phase expérimentale soit en cours, certains résultats sont déjà très prometteurs. Dans le cadre

du problème de *Bin Packing*, nous avons généré des instances du problème en suivant le papier de Castiñeiras et al. [1]. Appliquer l'algorithme du *Best-Fit Decreasing* au noeud racine de l'arbre de recherche a permis de diviser par 20 le temps total de résolution à l'optimal de l'ensemble des 100 instances de taille 10, comme le rapporte la figure 1 (BFD-Prop étant un propagateur suivant le cadre défini et utilisant l'algorithme du *Best-Fit Decreasing*). Cela nous paraît donc très prometteur pour l'apport que peut avoir notre propagateur pour la recherche de solution optimale sur de grosses instances. On veillera toutefois à distinguer les apports de notre propagateur par rapport à un simple *presolve* (ici l'application du *Best-Fit Decreasing* au noeud racine).

## 4 Conclusion et Perspectives

Dans cet article, nous présentons une façon de compléter les modèles de résolution de Problèmes d'Optimisation sous Contraintes à l'aide de propagateurs basés chacun sur un algorithme d'approximation. Cette approche devrait permettre d'améliorer l'efficacité des solveurs de contraintes pour résoudre ces Problèmes d'Optimisation sous Contraintes. Les premiers résultats expérimentaux sont très prometteurs mais nécessitent d'être étendus - principalement en utilisant une implémentation réelle de notre propagateur - afin de pouvoir mieux qualifier l'apport de cette approche. Il est notamment prévu de montrer la généricité de l'approche en l'appliquant à différentes familles de problèmes.

Enfin, il faudra étudier si cette approche peut être utilisée, et le cas échéant à quel point ses effets varient, lorsque l'on ajoute des contraintes supplémentaires au problème initial.

## Références

[1] I. Castiñeiras and M. De Cauwer and B. O'Sullivan : Weibull-Based Benchmarks for Bin Packing. *In Proceedings CP'12*, pages 207–222, 2012.

[2] G. Derval and P. Schaus and J-C. Régin : Improved filtering for the bin-packing with cardinality constraint. *Constraints*, 3 :251–271, 2018.

[3] W. Fernandez de la Vega and G. S. Lueker : Bin packing can be solved within $1+\epsilon$ in linear time. *Combinatorica*, 1:349–355, 1981.

[4] M. R. Garey and D. S. Johnson : Computers and Intractability ; A Guide to the Theory of NP-Completeness. *W. H. Freeman & Co., New York, NY, USA*, 1979.

[5] D. S. Johnson and A. Demers and J. D. Ullman and M. R. Garey and R. L. Graham : Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM J. Comput*, 3(4):298–325, 1974.

[6] F. Pachet and P. Roy : Automatic Generation of Music Programs. *In Proceedings CP'99*, pages 331–345, 1999.

[7] P. Shaw : A constraint for bin packing. *In Proceedings CP'04*, pages 648–662, 2004.

# Une approche SAT incrémentale pour raisonner efficacement sur les réseaux de contraintes qualitatives

**Gaël Glorian**[1*] **Jean-Marie Lagniez**[1] **Valentin Montmirail**[2] **Michael Sioutis**[3]

[1]CRIL, Université d'Artois et CNRS, F62300, Lens, France
[2]I3S, Université Côte d'Azur et CNRS, Nice Sophia-Antipolis, France
[3]Département d'Informatique, Université d'Aalto, Espoo, Finlande
{glorian,lagniez}@cril.fr vmontmirail@i3s.unice.fr michael.sioutis@aalto.fi

**Résumé**

Le langage $\mathcal{RCC}8$ est un formalisme largement utilisé pour décrire des arrangements topologiques de régions dans l'espace. Deux problèmes fondamentaux sont associés au langage $\mathcal{RCC}8$ : la *satisfiabilité* et la *réalisation*. Soit un réseau de contraintes qualitatives (QCN) de $\mathcal{RCC}8$, le problème de satisfiabilité est de décider s'il est possible d'assigner des régions aux variables du QCN de telle sorte que les contraintes soient satisfaites (*solution*). Le problème de réalisation est le fait de produire un modèle spatial qui peut servir de solution. Dans cet article, nous combinons les deux lignes de recherches mentionnés au-dessus et nous explorons l'idée de relier le problème de satisfiabilité et de réalisation. Nous nous limitons aux QCN qui, quand ils sont satisfiables, sont réalisables avec des rectangles. En effet, nous proposons une approche SAT incrémentale afin d'être capable de raisonner sur le langage $\mathcal{RCC}8$ en nous laissant guider par les contre-exemples. Nous avons expérimentalement évalué notre approche et étudié ses performances face aux solveurs de l'état de l'art pour raisonner en $\mathcal{RCC}8$ en utilisant de nombreux ensembles d'instances. Notre approche tient la charge et est compétitive face aux solveurs de l'état de l'art sur les instances considérées.

**Abstract**

The $\mathcal{RCC}8$ language is a widely-studied formalism for describing topological arrangements of spatial regions. Two fundamental reasoning problems that are associated with $\mathcal{RCC}8$ are the problems of *satisfiability* and *realization*. Given a qualitative constraint network (QCN) of $\mathcal{RCC}8$, the satisfiability problem is deciding whether it is possible to assign regions to the spatial variables of the QCN in such a way that all of its constraints are satisfied (*solution*). The realization problem is producing an actual spatial model that can serve as a solution. In this article, we combine the two aforementioned lines of research and explore the opportunities that surface by interrelating the corresponding reasoning problems, viz., the problems of satisfiability and realization. We restrict ourselves to QCNs that, when satisfiable, are realizable with rectangles. In particular, we propose an *incremental* SAT-based approach for providing a framework that reasons about the $\mathcal{RCC}8$ language in a counterexample-guided manner. We experimentally evaluated our approach and studied its scalability against state-of-the-art solvers for reasoning about $\mathcal{RCC}8$ relations using a varied dataset of instances. The approach scales up and is competitive with the state of the art for the considered benchmarks.

## 1 Introduction

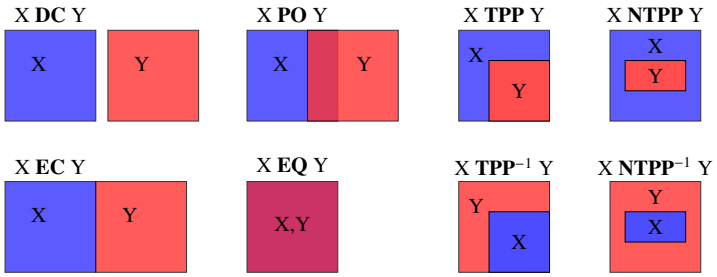
Le raisonnement qualitatif spatial et temporel (QSTR) est un domaine majeur de l'intelligence artificielle, en particulier dans la représentation des connaissances (KR), qui gère les concepts fondamentaux de l'espace et du temps dans un cadre qualitatif, abstrait, comme le fait un humain.

Pour illustrer cela, dans le langage naturel, nous utilisons les expressions telles que *dedans*, *avant*, *au nord de* pour spatialement et temporellement relier un objet à un autre ou à lui même, sans recourir à des informations quantitatives sur ces objets.

Formellement, QSTR limite le vocabulaire très riche des théories mathématiques qui gère l'espace et le temps d'objets à un simple langage de contraintes qualitatives. Ainsi, QSTR fourni un cadre concis qui permet des raisonnements peu coûteux au sujet d'objets localisés dans l'espace et le

*Papier doctorant : Gaël Glorian[1] est auteur principal.

FIGURE 1 – Illustration des relations de base $\mathcal{RCC}8$




temps. Ce cadre améliore la recherche et les applications à une pléthore de domaines incluant (sans être limité à) l'intelligence ambiante, le GIS dynamique, la robotique cognitive, le design spatio-temporel, et la génération de modèles qualitatifs depuis des sources vidéos [3, 25].

Au sujet du raisonnement spatial qualitatif, *Randell et al.* ont développé dans [21] le cadre le plus connu pour les calculs spatiaux dans QSTR, à savoir : Le calcul de connections de régions ($\mathcal{RCC}$). Il étudie les différentes relations qui peuvent être définies entre des régions dans un espace topologique ; ces régions sont basées sur la relation primitive de *connexion*. Par exemple, la relation *déconnectée* entre deux régions $X$ et $Y$ implique qu'aucun des points de la région $X$ n'est connecté à un point de la région $Y$ et réciproquement.

Deux fragments de $\mathcal{RCC}$, à savoir $\mathcal{RCC}8$ et $\mathcal{RCC}5$ (un sous-langage de $\mathcal{RCC}8$) où aucune importance n'est attachée aux frontières des régions), ont été utilisés dans de nombreuses applications de la vie réelle. En particulier, *Bouzy* dans [4] utilise $\mathcal{RCC}8$ dans la programmation du jeu de Go, *Lattner et al.* dans [3] utilise $\mathcal{RCC}5$ pour mettre en place des systèmes d'assistance dans les véhicules intelligents, et *Heintz et al.* dans [12] utilise $\mathcal{RCC}8$ dans le domaine des drones.

$\mathcal{RCC}8$ (qui sera l'objet de cet article) est basé sur les huit relations suivantes : égale (**EQ**), chevauchement partiel (**PO**), connecté à l'extérieur (**EC**), déconnecté (**DC**), la partie propre tangente (**TPP**) et son inverse (**TPP**$^{-1}$), et la partie propre non-tangente (**NTPP**) et son inverse (**NTPP**$^{-1}$). Ces relations spatiales sont illustrées sur la Figure 1.

Soit un réseau de contraintes qualitatives (QCN) défini sur $\mathcal{RCC}8$, nous nous sommes intéressés en particulier au problème de *satisfiabilité*, c'est-à-dire décider s'il existe une interprétation spatiale des variables du QCN qui satisfont ses contraintes. Le problème de satisfiabilité pour $\mathcal{RCC}8$ (et pour $\mathcal{RCC}5$) est NP-complet [23].

Une fois qu'un QCN de $\mathcal{RCC}8$ est connu comme étant satisfiable, c'est-à-dire n'ayant qu'une relation possible pour chacune des arêtes (sans aucun choix possible), nous devons typiquement s'attaquer au *problème de réalisation* (qui est un problème traitable [16]) afin de produire un véritable modèle spatial qui peut servir de solution.

D'autres problèmes de raisonnement fondamentaux que nous pouvons citer sont le problème de *l'étiquetage minimal* (autrement appelé la *fermeture déductive*) et le problème de la *redondance* [22]. Le problème de l'étiquetage minimal consiste à trouver la plus forte contrainte impliquée par le QCN, et le problème de la redondance est de déterminer si une contrainte donnée dans un QCN est impliqué par le reste du réseau (cette contrainte sera appelé *redondante* et sa suppression ne change pas l'ensemble des solutions du QCN). Le problème de la redondance, de l'étiquetage minimale et de satisfiabilité sont tous équivalents calculatoirement sous l'hypothèse de trouver une réduction polynomiale [11].

Les recherches en $\mathcal{RCC}8$ se focalisent habituellement soit sur une vérification symbolique de la satisfiabilité d'un QCN, soit en présentant une méthode pour *réaliser* un QCN satisfiable. À notre connaissance, combiner ces deux lignes de recherche d'une manière inter-reliée n'a pas été considéré dans la littérature. En effet, la première ligne correspond à des méthodes basées sur les contraintes, la seconde correspond à des structures mathématiques beaucoup plus complexes à implémenter.

Dans cet article, nous proposons une approche unifiée et homogène en utilisant une technique SAT incrémentale connue sous le nom de CEGAR, qui signifie **C**ounter-**E**xample **G**uided **A**bstraction **R**efinement [6]. L'idée est la suivante : au lieux de générer une formule propositionnelle équisatisfiable comme le fait l'état-de-l'art [13], nous générons une *sous-abstraction* (une formule qui est sous-contraintes, aussi appelée *relaxation* dans d'autres domaines). Se faisant, si la sous-abstraction n'est pas satisfiable, alors par construction, la formule originale non plus ; sinon, le solveur SAT va fournir en sortie un modèle qui va pouvoir être vérifié. Il se pourrait que l'approche soit chanceuse et que le modèle de la sous-abstraction soit aussi un modèle de la formule originale, et dans ce cas, le problème est décidé. En général, la sous-abstraction est raffinée, *c-à-d*, elle va se rapprocher de la formule originale et, dans le pire cas, devenir équisatisfiable après un nombre fini d'étapes de raffinement.

CEGAR a notamment été proposé dans de nombreux problèmes tel que la vérification bornée de modèles (BMC) [6], la satisfiabilité modulo des théories (SMT) [5], la planification [24] ou plus récemment pour la satisfiabilité minimale en logique modale S5 [14]. CEGAR, de manière globale, peut être vu comme une approche Lazy-SMT [7], où la

connaissance du problème qui est extraite de l'abstraction est utilisée pour guider les étapes des raffinements, au lieu d'un solveur de théorie.

## 2 Préliminaires

Dans cette section, nous supposons que le lecteur est familier avec les notions de la théorie des graphes et des topologies.

### 2.1 Le calcul de connexion de régions

Le calcul de connexion de régions ($\mathcal{RCC}$) [21] est une théorie du premier ordre pour représenter et raisonner au sujet d'informations méréotopologiques entre deux régions d'un même espace topologique. Ces relations sont basées sur la relation de connectivité (**C**). En particulier, en utilisant **C**, un ensemble de relations binaires est défini.

De cet ensemble, le fragment $\mathcal{RCC}8$ peut être extrait : {**DC**, **EC**, **PO**, **EQ**, **TPP**, **NTPP**, **TPP**$^{-1}$, **NTPP**$^{-1}$ }. Ces huit relations sont conjointement exhaustives et disjointes par paires, signifiant qu'une seule de ces relations peut être vraie entre deux régions. Comme indiqué dans l'introduction, ce fragment (illustré par la Figure 1), sera référé en tant que $\mathcal{RCC}8$ pour des raisons de facilité.

Nous pouvons voir les régions de $\mathcal{RCC}8$ comme des sous-ensembles non-vides réguliers d'un certain espace topologique qui ne doit pas être intérieurement connecté et n'a pas de dimension particulière, mais elles sont habituellement exigées *close* (*c-à-d*, que les sous-ensembles égalent la fermeture de leurs intérieurs respectifs).

Soit $R(\mathcal{X})$ qui désigne l'ensemble de toutes les régions d'un espace topologique $\mathcal{X}$. Alors, nous pouvons avoir la représentation suivante des relations de $\mathcal{RCC}8$, où $\mathbf{R}_i$ dénote les interprétations de **R** pour deux régions-variables instanciées. Sémantiquement, la relation binaire **R** contient toutes les instanciations possibles de ces paires de régions-variables.

**Définition 1** (Notations ensemblistes de $\mathcal{RCC}8$)**.** *Soient deux régions X et Y dans $R(\mathcal{X})$, alors :* [1]

| | | |
|---|---|---|
| $\boldsymbol{EQ}_i(X,Y)$ | *ssi* | $X = Y$ |
| $\boldsymbol{DC}_i(X,Y)$ | *ssi* | $X \cap Y = \emptyset$ |
| $\boldsymbol{EC}_i(X,Y)$ | *ssi* | $\mathring{X} \cap \mathring{Y} = \emptyset, X \cap Y \neq \emptyset$ |
| $\boldsymbol{PO}_i(X,Y)$ | *ssi* | $\mathring{X} \cap \mathring{Y} \neq \emptyset, X \not\subseteq Y, Y \not\subseteq X$ |
| $\boldsymbol{TPP}_i(X,Y)$ | *ssi* | $X \subset Y, X \not\subseteq \mathring{Y}$ |
| $\boldsymbol{TPP}_i^{-1}(X,Y)$ | *ssi* | $Y \subset X, Y \not\subseteq \mathring{X}$ |
| $\boldsymbol{NTPP}_i(X,Y)$ | *ssi* | $X \subset \mathring{Y}$ |
| $\boldsymbol{NTPP}_i^{-1}(X,Y)$ | *ssi* | $Y \subset \mathring{X}$ |

1. $\mathring{A}$ dénote l'intérieur de $A$

Soient deux relations de base **R** et **S** dans $\mathcal{RCC}8$ qui implique des paires de variables $(i, j)$ et $(j, k)$ respectivement, la *faible composition* de **R** et **S**, dénotée par *CT(**R**,**S**)*, produit la plus forte relation de $\mathcal{RCC}8$ qui contient $\mathbf{R} \circ \mathbf{S}$, *c-à-d*, qui produit le plus petit ensemble de relations de bases tel que, chacun d'entre eux peut être satisfait par une instanciation des variables $i$ et $k$ pour une instanciation possible des variables $i, j, k$ en fonction des relations **R** et **S**. Nous rappelons la définition de la composition faible :

**Définition 2** (Composition Faible *CT*)**.** *Soit deux relations de base **R**, **S** de $\mathcal{RCC}8$, leur composition faible CT(**R**,**S**) est défini comme le plus petit sous-ensemble $\{\boldsymbol{T}_1, \boldsymbol{T}_2, \ldots, \boldsymbol{T}_n\}$ de $2^{\mathcal{RCC}8}$ telle que $\boldsymbol{T}_i \cap (\boldsymbol{R} \circ \boldsymbol{S}) \neq \emptyset \ \forall i \in \{1, \ldots, n\}$.*

Le résultat de l'opérateur de composition faible pour chaque paire de relations de base de $\mathcal{RCC}8$ est fourni dans une table, appelée la *la table des compositions faibles* [17] ($\mathcal{RCC}8$ *CT* en plus court), illustrée dans la Table 1. Afin de capturer les informations spatiales qualitatives qui sont impliquées par la base de connaissances des relations $\mathcal{RCC}8$, nous allons utiliser la notion de réseaux de contraintes qualitatives (QCN) défini comme suit :

**Définition 3** (Réseaux de contraintes qualitatives (**QCN**))**.** *Un* QCN *de $\mathcal{RCC}8$ est une paire $\mathcal{N} = (V, C)$ où V est un ensemble non-vide fini de variables (chacune correspondant à une région), et C est une application associant une relation $C(v, v') \in 2^{\mathcal{RCC}8}$ avec chaque paire $(v, v')$ of $V \times V$. De plus, C est tel que $C(v, v) \subseteq \{\boldsymbol{EQ}\}$ et $C(v, v') = (C(v', v))^{-1}$.*

Concernant un QCN $\mathcal{N} = (V, C)$, nous avons les définitions suivantes : Une instanciation de $V$ est une application $\sigma$ de $V$ dans $R(\mathcal{X})$. Une solution (*réalisation*) $\sigma$ de $\mathcal{N}$ est une instanciation de $V$ telle que pour chaque paire $(v, v')$ des variable dans $V$, $(\sigma(v), \sigma(v'))$ satisfait $C(v, v')$, c'est-à-dire, qu'il existe une relation de base $b \in C(v, v')$ telle que $(\sigma(v), \sigma(v')) \in b$. $\mathcal{N}$ est satisfiable si et seulement si il admet une solution. Le graphe de contraintes d'un QCN $\mathcal{N}$ est le graphe $(V, E)$, dénoté $G_{\mathcal{N}}$, pour lequel nous avons que $\{v, v'\} \in E$ si et seulement si $C(v, v') \neq \mathcal{RCC}8$ (c'est-à-dire $C(v, v')$ correspond à une relation non-universelle) et $v \neq v'$.

Dans cet article, nous nous restreignons aux QCN qui, quand ils sont satisfiables, sont réalisables avec des rectangles. Comme indiqué dans [19, Exemple 1], il existe des QCN pour lesquels il n'est pas possible de trouver une réalisation rectangulaire en utilisant un seul rectangle par région (cela est toujours possible si plusieurs rectangles par régions sont utilisés). En revanche, pour les instances difficiles à résoudre (ce qui nous concerne ici), qui contiennent des connaissances non-définies et donc, sont plus *flexibles* en terme de réalisations, il est rarement (voire même jamais) le cas qu'une réalisation rectangulaire ne soit pas atteignable pour un QCN satisfiable (voir Section 5).

TABLE 1 – La $\mathcal{RCC}8$ *CT*, où * défini la relation universelle

| *CT* | **DC** | **EC** | **PO** | **TPP** | **NTPP** | **TPP$^{-1}$** | **NTPP$^{-1}$** | **EQ** |
|---|---|---|---|---|---|---|---|---|
| **DC** | * | DC EC PO TPP NTPP | DC EC PO TPP NTPP | DC EC PO TPP NTPP | DC EC PO TPP NTPP | DC | DC | DC |
| **EC** | DC EC PO TPP$^{-1}$ NTPP$^{-1}$ | DC EC PO TPP TPP$^{-1}$ EQ | DC EC PO TPP NTPP | EC PO TPP NTPP | PO TPP NTPP | DC EC | DC | EC |
| **PO** | DC EC PO TPP$^{-1}$ NTPP$^{-1}$ | DC EC PO TPP$^{-1}$ NTPP$^{-1}$ | * | PO TPP NTPP | PO TPP NTPP | DC EC PO TPP$^{-1}$ NTPP$^{-1}$ | DC EC PO TPP$^{-1}$ NTPP$^{-1}$ | PO |
| **TPP** | DC | DC EC | DC EC PO TPP NTPP | TPP NTPP | NTPP | DC EC PO TPP TPP$^{-1}$ EQ | DC EC PO TPP$^{-1}$ NTPP$^{-1}$ | TPP |
| **NTPP** | DC | DC | DC EC PO TPP NTPP | NTPP | NTPP | DC EC PO TPP NTPP | * | NTPP |
| **TPP$^{-1}$** | DC EC PO TPP$^{-1}$ NTPP$^{-1}$ | EC PO TPP$^{-1}$ NTPP$^{-1}$ | PO TPP$^{-1}$ NTPP$^{-1}$ | PO TPP TPP$^{-1}$ EQ | PO TPP NTPP | TPP$^{-1}$ NTPP$^{-1}$ | NTPP$^{-1}$ | TPP$^{-1}$ |
| **NTPP$^{-1}$** | DC EC PO TPP$^{-1}$ NTPP$^{-1}$ | PO TPP$^{-1}$ NTPP$^{-1}$ | PO TPP$^{-1}$ NTPP$^{-1}$ | PO TPP$^{-1}$ NTPP$^{-1}$ | PO TPP NTPP TPP$^{-1}$ NTPP$^{-1}$ EQ | NTPP$^{-1}$ | NTPP$^{-1}$ | NTPP$^{-1}$ |
| **EQ** | DC | EC | PO | TPP | NTPP | TPP$^{-1}$ | NTPP$^{-1}$ | EQ |

### 2.2 Logique Propositionnelle

La logique propositionnelle, dénotée *CPL* permet de raisonner avec ce qui est vrai (`True`) et ce qui est faux (`False`). La syntaxe de *CPL* peut-être formellement défini comme suit :

**Définition 4** (Langage de la logique propositionnelle)**.** *Soit $\mathbb{P}$ un ensemble infini dénombrable de variables propositionnelles. Le langage de la logique propositionnelle est l'ensemble des formules contenant $\mathbb{P}$, fermé sous l'ensemble des connecteurs logiques* $\{\neg, \wedge\}$.

Sans perte de généralité, nous supposons que toutes les formules de *CPL* sont en forme normale conjonctive (CNF) car n'importe quelle formule peut être transformée en une CNF équisatisfiable en utilisant l'algorithme de *Tseitin*. Sur les aspects sémantiques de la logique propositionnelle, la notion d'*interprétation* est importante. Elle est défini comme suit :

**Définition 5** (Interprétation)**.** *Une interprétation est un ensemble de valuations des variables propositionnelles. Formellement, c'est une application* $\mathbb{P} \rightarrow \{\texttt{True}, \texttt{False}\}$.

Une interprétation est un *modèle* de $\phi$ si $\phi$ est vraie pour cette interprétation. Si une formule a *au moins un* modèle $\mathcal{M}$, nous dirons que cette formule est *satisfiable* ; $\mathcal{M} \models \phi$ indiquera que $\mathcal{M}$ satisfait $\phi$. Formellement, la relation de satisfiabilité est défini comme suit :

**Définition 6** (Relation de satisfiabilité dans *CPL*)**.** *La relation $\models$ entre les interprétations $\mathcal{M}$ et les formules $\phi$ dans CPL est défini récursivement comme suit :*

$$\begin{array}{lll} \mathcal{M} \models p & ssi & p \in \mathcal{M} \\ \mathcal{M} \models \neg\phi & ssi & \mathcal{M} \not\models \phi \\ \mathcal{M} \models \phi_1 \wedge \phi_2 & ssi & \mathcal{M} \models \phi_1 \textit{ et } \mathcal{M} \models \phi_2 \\ \mathcal{M} \models \phi_1 \vee \phi_2 & ssi & \mathcal{M} \models \phi_1 \textit{ ou } \mathcal{M} \models \phi_2 \end{array}$$

Si une formule est satisfiable par toutes les interprétations, nous dirons que cette formule est *valide* ; dans ce cas, la formule est une *tautologie* et nous l'indiquerons par $\models \phi$. Si une formule est fausse pour toutes les interprétations, nous dirons que cette formule est *insatisfiable*.

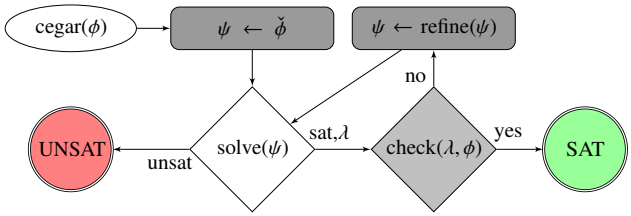
### 2.3 Préliminaires sur CEGAR

*Counter-Example-Guided Abstraction Refinement* (CEGAR) est une manière incrémentale de décider la satisfiabilité de formules en logique propositionnelle. Elle a été proposée initialement pour le problème de la vérification de modèles [6], c'est-à-dire, pour répondre à des questions du type "Est-ce que $S \models P$ est vrai ?" ou dit autrement, "Est-ce que $S \wedge \neg P$ est insatisfiable ?" où $S$ décrit un système et $P$ une propriété. Très souvent, dans des problèmes très structurés, seule une petite partie de la formule est nécessaire pour répondre à la question.

La pierre angulaire de CEGAR est de remplacer $\phi = S \wedge \neg P$ par une abstraction $\phi'$, où $\phi'$ doit être plus facile à résoudre en pratique que $\phi$. Il y a deux types d'abstractions : une sur-abstraction (resp. sous-abstraction) de $\phi$ est une formule $\hat{\phi}$ (resp. $\check{\phi}$) tel que $\hat{\phi} \models \phi$ (resp. $\phi \models \check{\phi}$) est vraie. $\hat{\phi}$ a au plus, autant de modèles que $\phi$ et $\check{\phi}$ a au moins autant de modèle que $\phi$.

Une illustration de l'approche CEGAR utilisant des sous-abstractions est donnée en Figure 2. Pour être correct, complet et pour terminer, une approche CEGAR doit vérifier

FIGURE 2 – Le cadre CEGAR avec des sous-abstractions




certaines hypothèses (les preuves peuvent être obtenus dans [15, Theo. 1,2 and 3])) :

1. "solve" est correcte, complete, et termine ;
2. si $\check{\phi}$ est insatisfiable, alors $\phi$ est insatisfiable ;
3. "check($\lambda, \phi$)" retourne vrai si $\lambda$ est un modèle de $\phi$ ;
4. $\exists$ n $\in \mathbb{N}$ tel que refine$^n(\check{\phi})$ est équisatisfiable avec $\phi$.

Comme nous utilisons un solveur SAT dans notre approche, nous supposons que le solveur SAT intégré est bien codé, qu'il est correct, complet et qu'il termine. Ainsi l'hypothèse (1) de CEGAR est satisfaite.

Maintenant que nous avons introduit les notions nécessaires à la compréhensions des contributions, la section suivante détaille la première d'entre elles, qui est l'encodage de $\mathcal{RCC}8$ en logique propositionnelle d'une telle manière qu'elle va nous permettre de vérifier facilement les hypothèses (2) et (4) de CEGAR.

## 3 Encoder RCC8 en SAT

Pour obtenir un encodage SAT du problème de satisfiabilité en $\mathcal{RCC}8$, nous devons définir comment traduire chacune des relations possibles. Nous allons représenter une région $i$ comme un ensemble de quatre variables $\{x_i^-, y_i^-, x_i^+, y_i^+\}$ comme illustré sur la Figure 3.

Tous les cas possibles pour chacune des relations peuvent être trouvés et sont prouvés, ainsi que leurs liens avec l'algèbre des points, dans [18, Table 6.2]. De cette table, nous pouvons proposer l'encodage SAT suivant pour toutes les arrêtes possibles :

**Définition 7** (Encodage SAT – `tr`)**.** *Pour toutes les relations $R$ dans toutes les arrêtes $(i, j)$ du problème $\mathcal{N}$ en entrée, nous avons :*

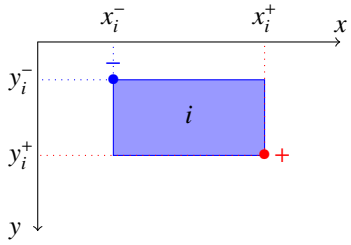
$$\texttt{tr}(\mathcal{N}) := \bigwedge_{\forall (R,i,j) \in \mathcal{N}} \texttt{tr}(R, i, j)$$

Ainsi, depuis [18, Table 6.2], si nous voulons traduire par exemple, la relation **EC** entre les nœuds $i$ et $j$ (la procédure est similaire pour les autres relations $\mathcal{RCC}8$), nous allons avoir l'encodage SAT suivant comme indiqué dans la Définition 7 :

**Définition 8** (Encodage SAT de **EC** sur l'arête i-j)**.**

$$\texttt{tr}(\boldsymbol{EC}, i, j) := \boldsymbol{EC}(i, j) \rightarrow$$
$$(\boldsymbol{EC}_r(i, j) \vee \boldsymbol{EC}_l(i, j) \vee \boldsymbol{EC}_u(i, j) \vee \boldsymbol{EC}_d(i, j))$$

FIGURE 3 – Illustration de la représentation d'une région




De cette définition, nous pouvons voir que la relation **EC** pour l'arête $(i, j)$ ne peut être satisfaite que de quatre manière différentes, à savoir : *par la gauche*, *par la droite*, *par le haut* et *par le bas*. Chacun des cas est défini de la manière suivante :

$$\begin{aligned}
\mathbf{EC}_r(i, j) \rightarrow & ((x_i^- < x_j^-) \wedge (x_i^- < x_j^+)) && \wedge \\
& ((x_i^- = x_j^+) \wedge (x_i^+ < x_j^+)) && \wedge \\
& ((y_i^- < y_j^+) \vee (y_i^- < y_j^+)) && \wedge \\
& ((y_i^+ < y_j^-) \vee (y_i^+ < y_j^-))
\end{aligned}$$

$$\begin{aligned}
\mathbf{EC}_u(i, j) \rightarrow & ((x_i^- < x_j^-) \vee (x_i^- = x_j^-)) && \wedge \\
& ((x_i^- > x_j^+) \vee (x_i^- = x_j^+)) && \wedge \\
& ((y_i^- < y_j^-) \wedge (y_i^- < y_i^+)) && \wedge \\
& ((y_i^+ = y_j^-) \wedge (y_i^+ < y_j^+))
\end{aligned}$$

$$\begin{aligned}
\mathbf{EC}_l(i, j) \rightarrow & ((x_i^- > x_j^-) \wedge (x_i^- = x_j^+)) && \wedge \\
& ((x_i^- > x_j^+) \wedge (x_i^+ > x_j^+)) && \wedge \\
& ((y_i^- < y_j^+) \vee (y_i^- < y_j^+)) && \wedge \\
& ((y_i^+ < y_j^-) \vee (y_i^+ < y_j^-))
\end{aligned}$$

$$\begin{aligned}
\mathbf{EC}_d(i, j) \rightarrow & ((x_i^- < x_j^-) \vee (x_i^- = x_j^-)) && \wedge \\
& ((x_i^- > x_j^+) \vee (x_i^- = x_j^+)) && \wedge \\
& ((y_i^- > y_j^-) \wedge (y_i^- = y_i^+)) && \wedge \\
& ((y_i^+ > y_j^-) \wedge (y_i^+ > y_j^+))
\end{aligned}$$

Les relations inverses sont définies comme d'habitude : $\mathbf{TPP}^{-1}(i, j) = \mathbf{TPP}(j, i)$ et $\mathbf{NTPP}^{-1}(i, j) = \mathbf{NTPP}(j, i)$. Pour chaque nœud dans le QCN avec $N$ nœuds que nous essayons de résoudre, nous allons ajouter la contrainte suivante qui permet de garantir que les coordonnées des points sont dans le bon ordre :

$$\bigwedge_{i=1}^{N} ((x_i^- < x_i^+) \wedge (y_i^- < y_i^+))$$

Nous voulons aussi indiquer que, si la variable propositionnelle $(A < B)$ est vraie, alors les variables $(A > B)$ et

$(A = B)$ sont fausses. Pour exprimer cela, nous ajoutons les clauses suivantes :

$$\texttt{AMO} := \bigwedge_{a\in\{x,y\}} \bigwedge_{c_1\in\{-,+\}} \bigwedge_{c_2\in\{-,+\}} \bigwedge_{i=1}^{N} \bigwedge_{j=1}^{N} \left( \begin{array}{l} ((a_i^{c_1} < a_j^{c_2}) \vee (a_i^{c_1} = a_j^{c_2}) \vee (a_i^{c_1} > a_j^{c_2})) \wedge \\ (\neg(a_i^{c_1} < a_j^{c_2}) \vee \neg(a_i^{c_1} = a_j^{c_2})) \wedge \\ (\neg(a_i^{c_1} < a_j^{c_2}) \vee \neg(a_i^{c_1} > a_j^{c_2})) \wedge \\ (\neg(a_i^{c_1} = a_j^{c_2}) \vee \neg(a_i^{c_1} > a_j^{c_2})) \wedge \end{array} \right) \quad (1)$$

Grâce à l'équation 1 (AMO – At Most One), nous pouvons ainsi remplacer, par exemple, dans **EC** (i,j) (u), $(x_i^- < x_j^+) \vee (x_i^- = x_j^+)$ par $\neg(x_i^- > x_j^+)$. De la même manière, pour toutes les disjonctions dans [18, Table 6.2].

Enfin, nous voulons aussi assurer la transitivité des relations sur toutes les coordonnées possibles. Cela aura l'impact le plus important sur la taille de la CNF générée. Pour chaque triplet $(i, j, k)$ dans la triangulation du graphe de contraintes de l'entrée QCN, nous devons ajouter les règles suivantes pour toutes combinaisons de $(c1, c2)$ qui assurent la transitivité $\in \{(-,-),(-,+),(+,-),(+,+)\}$ et pour les deux axes $a \in \{x, y\}$ :

$$\texttt{transitivity}(i,j,k) := \bigwedge \left( \begin{array}{l} ((a_i^{c1} = a_j^{c1}) \wedge (a_j^{c1} = a_k^{c2})) \rightarrow (a_i^{c1} = a_k^{c2}) \\ ((a_i^{c1} < a_j^{c1}) \wedge \neg(a_j^{c1} > a_k^{c2})) \rightarrow (a_i^{c1} < a_k^{c2}) \\ ((a_i^{c1} > a_j^{c1}) \wedge \neg(a_j^{c1} < a_k^{c2})) \rightarrow (a_i^{c1} > a_k^{c2}) \\ ((a_j^{c2} > a_k^{c2}) \wedge \neg(a_i^{c1} < a_j^{c1})) \rightarrow (a_i^{c1} > a_k^{c2}) \\ ((a_j^{c1} < a_k^{c2}) \wedge \neg(a_i^{c1} > a_j^{c1})) \rightarrow (a_i^{c1} < a_k^{c2}) \end{array} \right)$$

Nous ne rentrerons pas dans le détail sur la manière de trianguler un graphe, c'est une procédure standard. Il est à noter que trianguler un graphe prend un temps linéaire dans la taille du graphe triangulé en sortie. Avant de passer à la partie CEGAR, nous devons prouver que notre encodage est correct et complet.

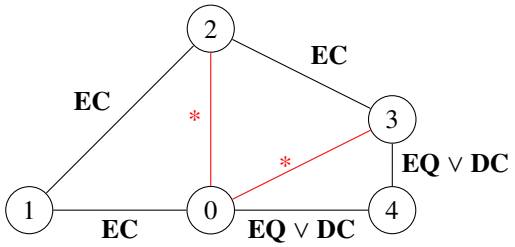
**Théorème 1** (Preuve dans [10])**.** *Soit $\mathcal{N} = (V, C)$ un* QCN *de $\mathcal{RCC}8$, et $G$ un super-graphe, du graphe de contraintes de $\mathcal{N}$, triangulé. Si* $\texttt{toSAT}(\mathcal{N})$ *est défini comme suit :*

$$\texttt{toSAT}(\mathcal{N}) := \texttt{tr}(\mathcal{N}) \wedge \texttt{AMO} \wedge \bigwedge_{i=1}^{N} ((x_i^- < x_i^+) \wedge (y_i^- < y_i^+)) \wedge \bigwedge_{(i,j,k)\in G} \texttt{transitivity}(i,j,k)$$

*alors* $\texttt{toSAT}(\mathcal{N})$ *est équisatisfiable à $\mathcal{N}$.*

Nous venons juste de voir que si nous encodons chacune des transitivités pour chaque triangle dans le graphe triangulé, nous obtenons une formule propositionnelle équisatisfiable. Cependant, quand nous regardons de plus près,

FIGURE 4 – Un exemple de graphe de contraintes d'un problème $\mathcal{RCC}8$ $\phi$, (les labels sur les arrêtes dénotent les relations $\mathcal{RCC}8$ correspondantes), en rouge, les arrêtes ajoutées lors de la triangulation du graphe.




nous pouvons voir que cela est très coûteux en temps, la fonction `transitivity` est exactement ce que nous voulons éviter à tout prix. C'est donc pourquoi nous proposons une approche CEGAR pour l'éviter.

## 4 Traduire parcimonieusement les contraintes de transitivité

Comme expliqué précédemment, la fonction `transitivity` peut être très coûteuse et par conséquent, doit être évitée pour avoir une approche compétitive. C'est exactement l'hypothèse sur laquelle notre approche CEGAR repose. Prenons l'exemple illustré sur la Figure 4, un problème $\mathcal{RCC}8$ donné en entrée avec 5 nœuds et 5 relations (les relations sont mises en caractères **gras** sur la Figure 4)). Ainsi, quand nous traduisons ce problème en logique propositionnelle, nous obtenons les règles suivantes :

$$\begin{aligned} \texttt{under}(\phi) = \bigwedge_{i=0}^{4} ((x_i^- < x_i^+) \wedge (y_i^- < y_i^+)) \\ \wedge\ \texttt{AMO} \wedge EC_{0,1} \wedge EC_{1,2} \wedge EC_{2,3} \\ \wedge\ (EQ_{3,4} \vee DC_{3,4}) \wedge (EQ_{4,0} \vee DC_{4,0}) \\ \wedge\ \texttt{tr}(\mathbf{EC}, 0, 1) \wedge \texttt{tr}(\mathbf{EC}, 1, 2) \\ \wedge\ \texttt{tr}(\mathbf{EC}, 2, 3) \wedge \texttt{tr}(\mathbf{EC}, 0, 1) \\ \wedge\ \texttt{tr}(\mathbf{EQ}, 3, 4) \wedge \texttt{tr}(\mathbf{DC}, 3, 4) \\ \wedge\ \texttt{tr}(\mathbf{EQ}, 4, 0) \wedge \texttt{tr}(\mathbf{DC}, 4, 0) \end{aligned}$$

**Théorème 2.** *Soit $\phi$ un* QCN*, alors* $\texttt{under}(\phi)$ *est une sous-abstraction de $\phi$ (c-à-d, il a au moins autant de modèles).*

*Démonstration.* $\texttt{under}(\phi)$ est un sous-ensemble des clauses de l'encodage équisatisfiable (Théorème 1). Ainsi si under($\phi$) est insatisfiable, alors $\phi$ est aussi insatisfiable par définition de la conjonction logique. □

À ce point, la traduction est une sous-abstraction du problème original, c-à-d, si elle est insatisfiable, alors il est sûr que le problème est insatisfiable, mais un modèle de cette

traduction n'implique pas forcément être un modèle du problème original. L'hypothèse CEGAR (2) est respectée par construction de cette traduction, pour obtenir l'équisatisfiabilité, nous avons besoin de :

$$\begin{aligned}\text{toSAT}(\phi) = \texttt{under}(\phi)\\ &\wedge \texttt{transitivity}(0,1,2)\\ &\wedge \texttt{transitivity}(0,2,3)\\ &\wedge \texttt{transitivity}(0,3,4)\end{aligned}$$

Comme le nombre de triangles dans le graphe triangulé est borné par un nombre $N$ (pire des cas : $N = |V|^3$ quand le graphe est complet), nous pouvons facilement voir qu'après la traduction de la transitivité pour chaque triangle (une opération que nous appellerons maintenant un raffinement), nous pouvons raffiner le problème $N$ fois et obtenir une formule équisatisfiable. Ceci permet de respecter l'hypothèse CEGAR (4).

À propos de l'hypothèse CEGAR (4), nous avons besoin d'une manière efficace pour vérifier si le modèle de la sous-abstraction retournée est aussi un modèle de la formule originale. Pour cela, nous utilisons l'algorithme *Directional Path Consistency* (DPC) présenté dans [8, 20, 27]. Notre fonction check réalise une vérification de modèle et retourne les triangles qui ne peuvent pas être validés par le modèle (qui retourne l'ensemble vide sur leurs relations). À partir de là, si le vérificateur retourne le triangle $(i, j, k)$ et que nous ajoutons ensuite les contraintes de transitivité `transitivity`$(i, j, k)$ dans la formule propositionnelle, alors il est impossible pour le vérificateur de retourner une fois de plus le même triangle. Comme discuté plus tôt, l'ensemble maximal de contraintes de transitivité que nous pouvons ajouter est borné, nous obtenons donc dans le pire des cas, une formule équisatisfiable.

Nous avons maintenant toutes les pièces du puzzle pour créer deux manières différentes pour résoudre le problème de satisfiabilité et de réalisation en $\mathcal{RCC}8$. La première consiste à utiliser un encodage direct (avec la fonction `toSAT`$(\mathcal{N})$). La seconde est d'utiliser une approche CEGAR, comme celle présenté dans l'Algorithme 1, qui dans le pire des cas (le cas où toutes les transitivités ont été ajoutées) va finir comme une version plus lente de l'encodage direct ; Cependant, cela ne s'est jamais produit lors de nos expérimentations (voir Section 5). De plus, à chaque fois qu'une instance a été satisfiable, nous avons pu obtenir une réalisation. En d'autres termes, nous avons résolu le problème de satisfiabilité et de réalisation ensemble.

**Algorithme 1 :** CEGAR-$\mathcal{RCC}8(\mathcal{N})$

**Data :** $\mathcal{N}$=(V,C) with $n$ variables
**Result :** Une réalisation de $\mathcal{N}$ s'il est possible d'en obtenir une, UNSAT sinon

1 $G \leftarrow (V, E \leftarrow E(G_{\mathcal{N}}))$;
2 setOfTriangles $\leftarrow Triangule(G)$;
3 transitivity $\leftarrow \top$;
4 $\psi \leftarrow under(N)$ ; `// étape de sous-abstraction`
5 **while** *(setOfTriangle* $\neq \emptyset$*)* **do**
6 $\quad\lambda \leftarrow$ SAT-Solver($\psi \wedge$ transitivity) ; `// étape de résolution`
7 $\quad$**if** ($\lambda = \bot$) **then return** UNSAT ;
8 $\quad res \leftarrow$ check($\lambda, N$) ; `// étape de vérification`
9 $\quad$**if** ($res = null$) **then return** interpret($\lambda$) ;
10 $\quad$**else**
11 $\quad\quad$setOfTriangle.remove(res);
12 $\quad\quad$transitivity $\leftarrow$ transitivity $\wedge$`transitivity`$(res)$ ; `// étape de raffinement`
13 $\lambda \leftarrow$ SAT-Solver($\psi \wedge$ transitivity) ; `// pire cas: équisatisfiable`
14 **if** ($\lambda = \bot$) **then return** UNSAT ;
15 **else return** interpret($\lambda$) ;

## 5 Résultats Expérimentaux

Maintenant que nous avons notre nouvel encodage SAT et une approche CEGAR pour résoudre les problèmes de satisfiabilité et de réalisation en $\mathcal{RCC}8$, nous voulons comparer cela à l'état de l'art. Pour cela, nous avons implémenté nos approches dans le solveur Churchill [2] et nous avons utilisé Glucose [1, 9] en solveur SAT interne.

Nous comparons Churchill en encodage direct et en mode CEGAR face aux solveurs de l'état de l'art pour le raisonnement spatio-qualitatif en $\mathcal{RCC}8$, que sont : GQR , Renz-Nebel01 , RCC8SAT, PPyRCC8, et Chordal-Phalanx. Chaque solveur est utilisé dans son mode par défaut, à partir GQR pour lequel nous utilisons le flag "-c horn". En utilisant ce flag, GQR décompose une relation $\mathcal{RCC}8$ en sous-relations de Horn (qui est le comportement standard pour le reste des solveurs) ; Cela change son facteur de branchement de 4 à $\sim$ 1.4. De plus, PPyRCC8 et Chordal-Phalanx utilisent `PyPy`, comme recommandé par leurs auteurs, afin d'améliorer les performances globales. Nous comparons ces solveurs sur quatre catégories de benchmarks générés aléatoirement de différentes manières (voir [10] pour en savoir plus).

À propos des ensembles 3 et 4, *scale-free networks* sont des réseaux dont la distribution des degrés suit une loi de puissance [2], ces réseaux structurés ont été utilisés récemment [26]. Les expérimentations ont été réalisées sur un cluster de Xeon 4 cœurs cadencés à 3.3 GHz sous CentOS 7.0 avec une limite mémoire de 32 Go et une limite de temps de 900 secondes par solveur par instance. En vérifiant les

2. Le nom vient du personnage historique qui prenait beaucoup de CEGAR

FIGURE 5 – Distribution des temps d'exécution sur le premier ensemble

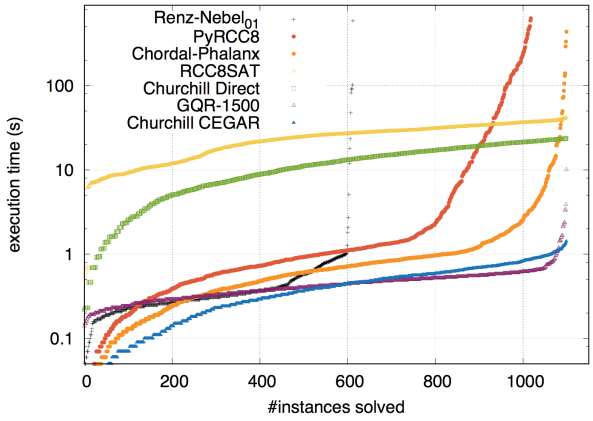
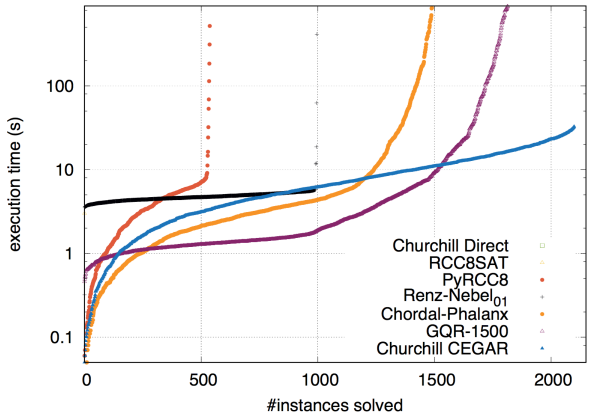



FIGURE 6 – Distribution des temps d'exécution sur le deuxième ensemble




réponses, comme aucune contradiction n'est apparue (tous les solveurs ont répondu la même chose), nous avons conclu que le cas où un QCN est satisfiable mais pas réalisable avec des rectangles n'est pas apparu dans nos ensembles de tests.

En regardant les Figures 5 et 6 qui indiquent les distributions des temps de résolutions des différents solveurs sur le premier et le deuxième ensemble d'instances, nous pouvons constater que Churchill et GQR sont extrêmement rapides pour résoudre ces ensembles. En effet, cela prend au maximum 1.42 seconde à Churchill pour résoudre la plus dure des instances du premier ensemble, 10.10 secondes pour GQR et 32.70 secondes pour le deuxième ensemble pour Churchill. Pour Churchill, cette rapidité vient du fait que nous effectuons en moyenne qu'un petit nombre de boucles CEGAR (moy : 8.90 pour le premier ensemble et 11.87 pour le deuxième ensemble). Cependant quand nous regardons les traductions directes (RCC8SAT et Churchill Direct), tenir dans la mémoire autorisée est difficile à partir de 500 nœuds.

La table 3 montre le nombre d'instances résolues pour les ensembles 3 et 4. Les meilleurs résultats d'une ligne sont présentés en gras et le nombre d'instances qui ne peuvent pas être résolues dû à la limite mémoire est indiqué entre parenthèses (si aucun dépassement de mémoire, un trait est affiché). La ligne VBS représente le *Virtual Best Solver* (une borne supérieure pratique représentant les performances atteignables en choisissant à chaque fois les résultats du meilleur solveur pour chaque instance).

Sur le troisième ensemble, nous pouvons voir la mise à l'échelle de l'approche CEGAR face à l'encodage direct (Churchill Direct) ou via une représentation CP. Les résultats sont clairs, quand le nombre de nœuds devient trop important, les approches SAT demandent trop de mémoire ou trop de temps pour effectuer la traduction et donc, deviennent inefficaces. Plus le réseau est gros, plus Churchill CEGAR prend de temps pour vérifier le modèle retourné par le solveur SAT et plus cela prend de mémoire pour ajouter les contraintes de transitivités. Dans certains cas, nous atteignons la limite de mémoire et sommes incapable de résoudre l'instance.

Sur le quatrième ensemble, nous étudions la mise à l'échelle sur des instances très difficiles mais de taille raisonnable. Nous pouvons voir ici que les approches basées sur SAT sont en difficulté avec la taille de l'entrée et qu'utiliser une approche CEGAR au lieu d'un encodage direct permet d'améliorer grandement les performances. En effet, Churchill CEGAR a réussi à résoudre toutes les instances, mais, malheureusement, il prend plus de temps que Chordal-Phalanx dans beaucoup de cas (médian : 37.97s pour Churchill contre 16.78s pour Chordal-Phalanx) ; Cependant il est plus rapide dans le pire cas (max : 163.10s pour Churchill contre 714.12s pour Chordal-Phalanx). Cela est clairement dû au fait que vérifier les modèles plusieurs fois, ce qui est typiquement le cas quand le réseau a une taille entre 2 000 et 3 500 nœuds, prend énormément de temps. En effet, dans la Table 2, un résumé du découpage en trois étapes du temps de résolution de Churchill : la triangulation, la vérification des modèles et la résolution des problèmes SAT est donné. Quand nous analysons les résultats donnés dans la Table 2, les résultats sont clairs : quand le réseau est petit (1$^{er}$ et 2$^{ème}$ ensemble) la plupart du temps est passé dans la triangulation du graphe. Quand le réseau est gros (3$^{ème}$ et 4$^{ème}$ ensemble) la plupart du temps est passé dans la vérification des modèles. Dans tous les cas, le solveur SAT n'est jamais coûteux. Pour chaque résultat, il est important de se rappeler que nous résolvons les problèmes de satisfiabilité et de réalisation, pas simplement la satisfiabilité comme les autres solveurs.

## 6 Conclusion

Dans ce papier, une nouvelle approche pour résoudre les problèmes de satisfiabilité et de réalisation en $\mathcal{RCC}8$

TABLE 2 – Résumé des temps des trois étapes dans Churchill

| Temps (s) | Triangulation | | | Vérification | | | Résolution | | |
|---|---|---|---|---|---|---|---|---|---|
| | min | med | max | min | med | max | min | med | max |
| Premier Ensemble | 0.220 | 0.390 | 0.630 | 0.015 | 0.030 | 0.151 | 0.002 | 0.003 | 0.010 |
| Deuxième Ensemble | 3.910 | 12.910 | 20.153 | 0.430 | 1.170 | 2.057 | 0.015 | 0.030 | 0.370 |
| Troisième Ensemble | 8.708 | 55.96 | 128.96 | 7.900 | 222.3 | 668.57 | 0.015 | 0.860 | 16.38 |
| Quatrième Ensemble | 3.765 | 21.530 | 68.230 | 0.560 | 24.410 | 58.950 | 0.012 | 0.250 | 15.73 |

TABLE 3 – Résultats sur les ensembles 3 (gauche) et 4 (droit)

| | *random-scale-free-like-instances* | | | | | | *random-scale-free-like-np8-instances* | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| #Nœuds (x1000)<br>#Instances | < 6<br>150 | 6<br>30 | 7<br>30 | 8<br>30 | 9<br>30 | 10<br>30 | 0.5<br>10 | 1<br>10 | 1.5<br>10 | 2<br>10 | 2.5<br>10 | 3<br>10 | 3.5<br>10 |
| Renz-Nebel01 | 0<br>- | 0<br>- | 0<br>- | 0<br>- | 0<br>- | 0<br>- | 0<br>- | 0<br>- | 0<br>- | 0<br>- | 0<br>- | 0<br>- | 0<br>- |
| RCC8SAT | 0<br>(150) | 0<br>(30) | 0<br>(30) | 0<br>(30) | 0<br>(30) | 0<br>(30) | 0<br>(10) | 0<br>(10) | 0<br>(10) | 0<br>(10) | 0<br>(10) | 0<br>(10) | 0<br>(10) |
| PPyRCC8 | 134<br>(14) | 19<br>(8) | 20<br>(7) | 21<br>(7) | 23<br>(7) | 23<br>(4) | **10**<br>- | 9<br>- | **10**<br>- | 8<br>- | 7<br>(1) | 3<br>(5) | 3<br>(7) |
| Chordal-Phalanx | **150**<br>- | **30**<br>- | **30**<br>- | **30**<br>- | **30**<br>- | **30**<br>- | **10**<br>- | **10**<br>- | **10**<br>- | **10**<br>- | **10**<br>- | **10**<br>- | **10**<br>- |
| GQR-1500 | **150**<br>- | **30**<br>- | **30**<br>- | **30**<br>- | **30**<br>- | **30**<br>- | **10**<br>- | **10**<br>- | 9<br>- | 6<br>- | **10**<br>- | 8<br>- | 9<br>- |
| Churchill Direct | 0<br>(150) | 0<br>(30) | 0<br>(30) | 0<br>(30) | 0<br>(30) | 0<br>(30) | 0<br>(10) | 0<br>(10) | 0<br>(10) | 0<br>(10) | 0<br>(10) | 0<br>(10) | 0<br>(10) |
| Churchill CEGAR | **150**<br>- | **30**<br>- | **30**<br>- | 18<br>(10) | 8<br>(20) | 6<br>(24) | **10**<br>- | **10**<br>- | **10**<br>- | **10**<br>- | **10**<br>- | **10**<br>- | **10**<br>- |
| VBS | 150 | 30 | 30 | 30 | 30 | 30 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |

en utilisant une approche basée sur les sous-abstractions dans le cadre CEGAR a été proposée. Nous avons démontré que notre encodage est correct et complet pour les QCN satisfiables qui sont réalisables avec des rectangles et nous l'avons instancié au travers du solveur Churchill.

Nous avons comparé notre approche face aux solveurs représentant, à notre connaissance, l'état de l'art pour la résolution pratique en $\mathcal{RCC}8$ dans un large éventail d'instances de différentes tailles et difficultés. Nous avons conclu qu'un encodage basique de notre approche n'est pas du tout compétitif, beaucoup des instances disponibles sont de taille importante et demande un grand nombre de contraintes de transitivités dans l'encodage SAT. Cependant, notre approche CEGAR mixant à la fois des courts-circuits SAT et UNSAT surpasse les autres solveurs sur les benchmarks considérés.

Comme travaux futurs, en considérant la table 2, nous pouvons améliorer notre vérificateur de modèle. En effet, éviter de vérifier les sous-graphes non modifiés en mettant des marqueurs sur certains nœuds, c'est-à-dire, en ne vérifiant que les parties qui ont été modifiées par la précédente assignation. De plus, afin de rendre notre approche correcte, même en dehors de réalisation rectangulaire, nous pouvons étendre l'approche CEGAR en une approche RECAR [15] où la sur-abstraction serait de considérer des formes de plus en plus complexes (d'abord des points, ensuite des rectangles, ensuite des rectangles qui permettent un trou au milieu, puis deux trous, etc.)

## 7 Remerciements

Les auteurs aimeraient remercier les relecteurs de *CP'18* pour leurs commentaires pertinents. Une partie de ce travail a été supportée par le Ministère de l'Enseignement Supérieur et la Recherche, par le projet ANR Investissement d'Avenir UCA$^{\text{JEDI}}$ (ANR-15-IDEX-01) et par le Conseil Régional des Haut-de-France au travers du "Contrat de Plan État Région (CPER) DATA".

## Références

[1] Gilles Audemard, Jean-Marie Lagniez et Laurent Simon : Improving Glucose for Incremental SAT Solving with Assumptions : Application to MUS Extraction. *In Proc. of SAT'13*, 2013.

[2] Albert-László Barabási et Réka Albert : Emergence of Scaling in Random Networks. *Science*, 286(5439), 1999.

[3] Mehul Bhatt, Hans W. Guesgen, Stefan Wölfl et Shyamanta Moni Hazarika : Qualitative Spatial and Temporal Reasoning : Emerging Applications, Trends, and Directions. *Spatial Cognition & Computation*, 11(1):1–14, 2011.

[4] Bruno Bouzy : Les concepts spatiaux dans la programmation du go. *Revue d'Intelligence Artificielle*, 15(2):143–172, 2001.

[5] Robert Brummayer et Armin Biere : Effective Bit-Width and Under-Approximation. *In Proc. of EUROCAST'09*, volume 5717 de *LNCS*. Springer, 2009.

[6] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu et Helmut Veith : CounterExample-Guided Abstraction Refinement For Symbolic Model Checking. *Journal of ACM*, 50(5), 2003.

[7] Leonardo Mendonça de Moura, Harald Ruess et Maria Sorea : Lazy Theorem Proving for Bounded Model Checking over Infinite Domains. *In Proc. of CADE'02*, 2002.

[8] Rina Dechter, Itay Meiri et Judea Pearl : Temporal Constraint Networks. *Artificial Intelligence*, 49(1-3), 1991.

[9] Niklas Eén et Niklas Sörensson : An Extensible SAT-solver. *In Proc. of SAT'03*, volume 2919 de *LNCS*. Springer, 2003.

[10] Gael Glorian, Jean-Marie Lagniez, Valentin Montmirail et Michael Sioutis : An Incremental SAT-Based Approach to Reason Efficiently on Qualitative Constraint Networks. *In Proc. of CP'18*, volume 11008, pages 160–178. Springer, 2018.

[11] Martin Charles Golumbic et Ron Shamir : Complexity and algorithms for reasoning about time : A graph-theoretic approach. *J. ACM*, 40(5):1108–1133, 1993.

[12] Fredrik Heintz et Daniel de Leng : Spatio-Temporal Stream Reasoning with Incomplete Spatial Information. *In Proc. of ECAI'14*, volume 263, pages 429–434. IOS Press, 2014.

[13] Jinbo Huang, Jason Jingshi Li et Jochen Renz : Decomposition and tractability in qualitative spatial and temporal reasoning. *Artificial Intelligence*, 195, 2013.

[14] Jean-Marie Lagniez, Daniel Le Berre, Tiago de Lima et Valentin Montmirail : An Assumption-Based Approach for Solving the Minimal S5-Satisfiability Problem. *In Proc. of IJCAR'18*, volume 10900, pages 1–18. Springer, 2018.

[15] Jean-Marie Lagniez, Daniel Le Berre, Tiago de Lima et Valentin Montmirail : A Recursive Shortcut for CEGAR : Application To The Modal Logic K Satisfiability Problem. *In Proc. of IJCAI'17*, 2017.

[16] Sanjiang Li : On Topological Consistency and Realization. *Constraints*, 11(1):31–51, 2006.

[17] Sanjiang Li et Mingsheng Ying : Region Connection Calculus : Its models and composition table. *Artif. Intell.*, 145(1-2):121–146, 2003.

[18] Zhiguo Long : *Qualitative Spatial And Temporal Representation And Reasoning : Efficiency in Time And Space*. Thèse de doctorat, University of Technology Sydney (UTS), January 2017.

[19] Zhiguo Long, Steven Schockaert et Sanjiang Li : Encoding Large RCC8 Scenarios Using Rectangular Pseudo-Solutions. *In Proc. of KR'16*, 2016.

[20] Zhiguo Long, Michael Sioutis et Sanjiang Li : Efficient Path Consistency Algorithm for Large Qualitative Constraint Networks. *In Proc. of IJCAI'16*, 2016.

[21] David A. Randell, Zhan Cui et Anthony G. Cohn : An interval logic for space based on "connection". *In ECAI*, pages 394–398, 1992.

[22] Jochen Renz : Qualitative spatial and temporal reasoning : Efficient algorithms for everyone. *In Proc. ocf IJCAI'07*, pages 526–531, 2007.

[23] Jochen Renz et Bernhard Nebel : On the Complexity of Qualitative Spatial Reasoning : A Maximal Tractable Fragment of the Region Connection Calculus. *Artificial Intelligence*, 108(1-2), 1999.

[24] Jendrik Seipp et Malte Helmert : Counterexample-Guided Cartesian Abstraction Refinement. *In Proc. of ICAPS'13*. AAAI, 2013.

[25] Michael Sioutis, Marjan Alirezaie, Jennifer Renoux et Amy Loutfi : Towards a synergy of qualitative spatio-temporal reasoning and smart environments for assisting the elderly at home. *In IJCAI Workshop on Qualitative Reasoning*, pages 901–907, 2017.

[26] Michael Sioutis, Jean-François Condotta et Manolis Koubarakis : An Efficient Approach for Tackling Large Real World Qualitative Spatial Networks. *IJAIT*, 25(2):1–33, 2016.

[27] Michael Sioutis, Zhiguo Long et Sanjiang Li : Leveraging Variable Elimination for Efficiently Reasoning about Qualitative Constraints. *IJAIT*, 27(4):1860001, 2018.

# Heuristiques exploitant la relaxation linéaire pour l'optimisation dans les réseaux de fonction de coût

**Fulya Trösser**[1*] **Simon de Givry**[1] **George Katsirelos**[2]
[1] MIAT, UR-875, INRA, F-31320 Castanet Tolosan, France
[2] UMR MIA-Paris, INRA, AgroParisTech, Université Paris-Saclay, 75005 Paris, France
{fulya.ural,simon.de-givry}@inra.fr gkatsi@gmail.com

**Résumé**

Les solveurs exacts pour le problème d'optimisation dans les modèles graphiques, comme par exemple les réseaux de fonctions de coût ou les champs aléatoires de Markov, utilisent généralement l'algorithme par séparation et évaluation. L'efficacité de la recherche dépend principalement de deux facteurs : la qualité de la borne calculée à chaque nœud de l'arbre de recherche et les heuristiques de branchement. Dans ce contexte, il y a un compromis à trouver entre la qualité de la borne et son coût de calcul. En particulier, l'algorithme de Cohérence d'Arc Virtuelle (*Virtual Arc Consistency or VAC*) produit des bornes de bonne qualité, mais avec un coût élevé, il est donc utilisé généralement pendant le prétraitement uniquement, au lieu de l'appliquer à chaque nœud de l'arbre de recherche.

Dans ce travail, nous identifions une faiblesse des solveurs par séparation et évaluation dans l'utilisation de VAC. C'est-à-dire qu'ils ignorent l'information sur la relaxation linéaire du problème que VAC fournit, à l'exception de la borne duale. Notamment, il se peut que l'heuristique de branchement prenne des décisions qui sont clairement inefficaces à la lumière de cette information. En éliminant ces décisions inefficaces, nous réduisons considérablement la taille de l'arbre de recherche. De plus, nous pouvons supposer de façon optimiste, que la relaxation linéaire est correcte en grande partie au niveau des affectations qu'elle effectue, ce qui aide à trouver des solutions de bonne qualité rapidement. La combinaison de ces méthodes montre une très bonne performance pour certaines familles d'instances et surpasse l'état de l'art.

**Abstract**

Exact solvers for optimization problems on graphical models, such as Cost Function Networks and Markov Random Fields, typically use branch-and-bound. Its efficiency relies mainly on two factors: the quality of the bound computed at each node of the branch-and-bound tree and the branching heuristics. In this respect, there is a trade-off between quality of the bound and computational cost. In particular, the Virtual Arc Consistency (VAC) algorithm computes high quality bounds but at a significant cost, so it is mostly used in preprocessing, rather than in every node of the search tree.

In this work, we identify a weakness in the use of VAC in branch-and-bound solvers, namely that they ignore the information that VAC produces on the linear relaxation of the problem, except for the dual bound. In particular, the branching heuristic may make decisions that are clearly ineffective in light of this information. By eliminating these ineffective decisions, we significantly reduce the size of the branch-and-bound tree. Moreover, we can optimistically assume that the relaxation is mostly correct in the assignments it makes, which helps find high quality solutions quickly. The combination of these methods shows great performance in some families of instances, outperforming the previous state of the art.

## 1 Introduction

Les modèles graphiques non dirigés, comme le problème de satisfaction de contraintes pondérées (*Weighted Constraint Satisfaction Problem or WCSP*) et les champs aléatoires de Markov (*Markov Random Field or MRF*), peuvent être utilisés afin de donner une représentation factorisée d'une fonction, dans lesquels les noeuds du graphe correspondent aux variables de la fonction et les (hyper-)arêtes correspondent aux facteurs. Les facteurs peuvent être, par exemple, des fonctions de coût, auquel cas le modèle graphique représente la factorisation d'une fonction de coût ; ou des tables de probabilités jointes, auquel cas le modèle représente une distribution globale de probabilité non normalisée.

*Papier doctorant : Fulya Trösser[1] est auteur principal.

Les deux modèles, WCSP et MRF, sont équivalents sous une transformation $-\log$, donc la tâche NP-difficile de minimisation de coût en WCSP est équivalente à la tâche d'affectation de probabilité maximum a posteriori en MRF. Ce problème d'optimisation a des applications diverses telles qu'en bioinformatique, en vision par ordinateur, *etc.*

Les méthodes de résolution exactes pour ce problème sont généralement basées sur l'algorithme par séparation et évaluation, sauf dans quelques exceptions notables. Par exemple, il est possible de modéliser l'optimisation d'un WCSP en tant qu'un problème linéaire en nombres entier (PLNE) et d'utiliser un solveur pour ce problème. Or, les solveurs PLNE doivent résoudre la relaxation linéaire de façon exacte afin d'obtenir un minorant à chaque noeud de l'arbre de recherche, ce qui est une opération très coûteuse en fonction de la taille des problèmes rencontrés dans de nombreuses applications. En revanche, les solveurs dédiés les plus performants utilisent des algorithmes qui résolvent la relaxation linéaire approximativement et donc potentiellement sous-optimalement. Spécifiquement, les algorithmes comme EDAC [9], VAC [7], TRWS [15] et d'autres, produisent des solutions faisables pour le dual de la relaxation linéaire du WCSP, qui peuvent être utilisées comme minorants. Ces algorithmes compensent la perte de précision en gagnant considérablement en efficacité de calcul. A l'inverse de la PLNE, non seulement la relaxation linéaire est approchée, mais c'est l'algorithme le plus faible, EDAC, qui est préféré durant la recherche, tandis que VAC ou TRWS sont principalement utilisés en prétraitement.

Parmi les exceptions à l'approche précédente, nous sommes intéressés par la méthode CombiLP [11], qui résout la relaxation linéaire et décompose le problème en deux parties : la partie "facile" correspond à l'ensemble des variables intégrales dans la relaxation et la partie combinatoire contient les variables restantes. Ceci est fait en identifiant une condition appelée cohérence d'arc stricte (*strict arc consistency or strictAC*) à partir de la solution duale produite par l'algorithme TRWS. La partie combinatoire est résolue de façon exacte par l'algorithme par séparation et évaluation, et si la solution peut être fusionnée avec la partie facile sans induire de coût supplémentaire, l'algorithme a prouvé l'optimalité. Sinon, il transfère certaines variables de la partie facile vers la partie combinatoire et réitère le processus.

Ici, nous faisons plusieurs contributions. Premièrement, nous améliorons la condition qu'utilise CombiLP afin de détecter l'intégralité. Nous montrons en Section 3 que cette condition est trop stricte et nous donnons une condition relâchée qui admet un ensemble plus grand de variables intégrales. Deuxièmement, nous montrons que l'on peut détecter un ensemble de variables intégrales de la relaxation à partir d'une solution duale réalisable sans avoir à favoriser la condition d'intégralité. Sur le plan pratique, nous introduisons deux heuristiques simples qui exploitent cette propriété au sein d'un algorithme par séparation et évaluation. La première heuristique présentée en Section 4 modifie l'heuristique de choix de branchement pour éviter de brancher sur les variables strictAC. La deuxième heuristique, présentée en Section 5, est une variante de l'heuristique RINS [8] connue en PLNE, qui fait l'hypothèse que les variables strictAC sont affectées à leur valeur intégrale dans la solution optimale et effectue une exploration arborescente dans le sous-problème des variables restantes dans le but de trouver un bon majorant. En Section 6, nous montrons que l'intégration de ces techniques dans le solveur toulbar2 [7] surpasse considérablement l'état de l'art pour certaines familles d'instances.

## 2 Préliminaires

**Definition 1.** Un *Problème de Satisfaction de Contraintes (CSP)* [6] est un triplet $\langle X, D, C\rangle$. $X = \{1, \ldots, n\}$ est un ensemble de $n$ variables. Chaque variable $i \in X$ a un domaine de valeurs $D_i \in D$ et peut être affectée à n'importe quelle valeur $a \in D_i$, également notée $(i, a)$. $C$ est un ensemble de contraintes.

Chaque contrainte $c_S \in C$ est définie sur un ensemble de variables $S \subseteq X$ par un sous-ensemble du produit Cartésien $\ell(S) = \prod_{i \in S} D_i$ qui définit tous les tuples de valeurs cohérents vis à vis de la contrainte. Nous supposons, sans perte de généralité, qu'au plus une contrainte est définie sur un ensemble de variables donné. La contrainte unaire sur la variable $i$ sera notée $c_i$, et les contraintes binaires $c_{ij}$. Par souci de simplification et en raison de limitations techniques de mise en oeuvre, nous nous focalisons sur des CSP binaires, c'est à dire pour lesquels chaque contrainte porte sur au plus deux variables. Le problème est de trouver une solution qui satisfait toutes les contraintes dans $C$. C'est un problème NP-complet.

**Definition 2.** Un CSP est *arc-cohérent* (AC) si $\forall c_{ij} \in C$, $\forall a \in D_i$, $\exists b \in D_j$ avec $\{(i, a), (j, b)\} \in c_{ij}$.

**Definition 3.** Un *Problème de Satisfaction de Contraintes Pondérées (WCSP)* [7] est un quadruplet $\langle X, D, C, k\rangle$ où $X$ est un ensemble de $n$ variables, chaque variable $i \in X$ a un domaine de valeurs possibles $D_i \in D$, comme pour un CSP. $C$ est un ensemble de fonctions de coût et $k$ est un entier positif ou infini qui sert de majorant.

Chaque fonction de coût $c_S \in C$ est définie sur un ensemble de variables $S \subseteq X$ et la fonction fait

correspondre à chaque affectation des variables de $S$ un coût entier positif ou nul.

Nous supposons que tous les WCSPs contiennent une fonction de coût unaire pour chaque variable et une fonction de coût $c_\emptyset$, qui représente une constante dans la fonction objectif. Puisque tous les coûts sont positifs, cette constante est un minorant du coût des affectations admises par le WCSP.

Nous nous restreignons à des WCSP binaires, bien que toutes les définitions et algorithmes présentés par la suite puissent se généraliser à des fonctions d'arité supérieure.

Un WCSP binaire peut être représenté de façon graphique comme illustré en tête de la Figure 1. Chaque variable $i \in X$ correspond à une cellule. Chaque valeur $a \in D_i$ correspond à un point dans la cellule. Un coût unaire $c_i(a)$ est écrit à côté du point s'il est non nul. S'il existe une fonction de coût binaire $c_{ij}$ ayant un coût non nul entre $(i, a)$ et $(j, b)$, alors une arête est mise entre les points qui correspondent à ces affectations.

Le problème est de trouver une solution $t$ qui minimise la somme de toutes les fonctions de coût dans $C$, notée $c_P(t) = c_\emptyset + \sum_{i \in X} c_i(t[i]) + \sum_{c_{ij} \in C} c_{ij}(t[i], t[j])$, et qui soit de coût total inférieur à $k$. C'est un problème NP-difficile.

Étant donné deux WCSPs $P = (X, D, C, k)$ et $P' = (X, D, C', k)$ tels que $\forall c_S \in C, \exists c'_S \in C'$ et vice versa, on dit qu'ils ont la même structure. Si $c_P(t) = c_{P'}(t)\ \forall t \in \ell(X)$, alors $P$ et $P'$ sont équivalents et ils sont des reparamétrisations de l'un à l'autre. Il a été montré dans [7] que la reparamétrisation optimale qui maximise le facteur constant $c_\emptyset$ correspond au dual du programme linéaire ci-dessous, appelé le polytope local du WCSP binaire :

$$\min \sum_{i \in X, a \in D_i} c_i(a)\, x_{i,a} + \sum_{c_{ij} \in C, a \in D_i, b \in D_j} c_{ij}(a,b)\, y_{i,a,j,b}$$

s.t.

$$\begin{aligned}
&\sum_{a \in D_i} x_{i,a} = 1 && i \in X \\
&x_{i,a} = \sum_{b \in D_j} y_{i,a,j,b} && i \in X, a \in D_i, c_{ij} \in C \\
&0 \le x_{i,a} \le 1 && i \in X, a \in D_i \\
&0 \le y_{i,a,j,b} \le 1 && c_{ij} \in C, a \in D_i, b \in D_j
\end{aligned} \tag{1}$$

La reparamétrisation est extraite à partir des coûts réduits des variables $x_{i,a}$ et $y_{i,a,j,b}$ dans la solution optimale du PL ci-dessus et en ajoutant à $c_\emptyset$ l'optimum de ce PL.

**Definition 4** ([7])**.** Soit le WCSP binaire $P = \langle X, D, C, k\rangle$. On note le CSP $Bool(P) = \langle X, \overline{D}, \overline{C}\rangle$ (resp. $Bool_\theta(P) = \langle X, \overline{D_\theta}, \overline{C_\theta}\rangle$) tel que pour tout $i \in X$, $a \in \overline{D}$ (resp. $\overline{D_\theta}$) si et seulement si $\exists a \in D_i, c_i(a) = 0$ (resp. $c_i(a) < \theta$) et pour tout $i, j \in X^2$, $r_{ij} \in \overline{C}$ (resp. $\overline{C_\theta}$) si et seulement si $\exists c_{ij} \in C$, avec $r_{ij}$ la relation $\forall a \in D_i, b \in D_j, \{(i,a),(j,b)\} \in r_{ij} \Leftrightarrow c_{ij}(a,b) = 0$ (resp. $c_{ij}(a,b) < \theta$).

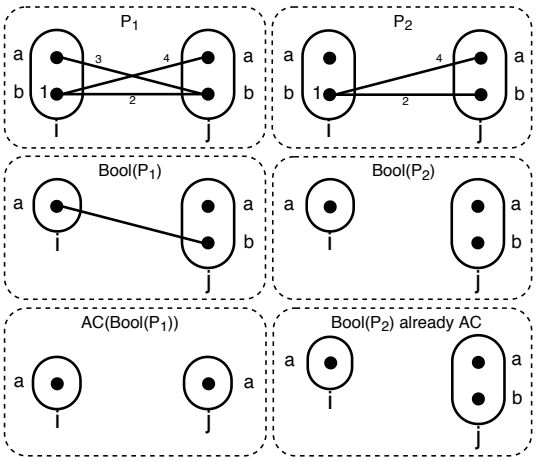



FIGURE 1 – Deux instances WCSP $P_1$, $P_2$, leurs transformations correspondantes en CSP par $Bool(P)$ et leurs fermetures arc-cohérentes de $Bool(P)$.

Un exemple du $Bool(P)$ est illustré dans la Figure 1. Ici une arête indique un tuple interdit.

**Definition 5.** Un WCSP $P$ est *virtuellement arc-cohérent* (VAC) si la fermeture arc-cohérente de $Bool(P)$ n'est pas vide [7].

Dans l'exemple, les deux fermetures sont arc-cohérentes, avec la valeur $b$ supprimée du domaine de $\overline{D_j}$ dans la fermeture de $Bool(P_1)$.

L'algorithme VAC construit itérativement $Bool(P)$ et si sa fermeture par cohérence d'arc est vide, il extrait une reparamétrisation améliorant le minorant [1]. Il termine lorsque la fermeture par cohérence d'arc n'est pas vide. Il converge ainsi vers un point-fixe qui n'est pas unique et qui peut être inférieur à l'optimum du PL 1.

Par la suite, nous supposons que $\forall c_S \in C$, $S \neq \emptyset$, $\min_{t \in \ell(S)} c_S(t) = 0$, ou sinon l'instance peut être triviallement reparamétrée pour augmenter le minorant.

## 3 Cohérence d'arc stricte

La cohérence d'arc stricte (*strictAC*) [18] est une façon de partitionner un WCSP en une partie "facile", qui

1. Le fait de garder des coûts entiers peut conduire à ne pas améliorer le minorant, posant un problème de terminaison de l'algorithme. Ceci est réglé en augmentant localement à une variable ou à une fonction de coût le seuil $\theta$ [7].

est résolue de façon exacte par un solveur de programmation linéaire et une partie combinatoire "difficile" résolue par un solveur d'optimisation combinatoire.

**Definition 6** (Cohérence d'arc stricte [18])**.** *Une variable $i \in X$ est arc-cohérente stricte s'il existe une valeur unique $a \in D_i$ telle que $c_i(a) = 0$ et $\forall c_{ij} \in C$, il existe un tuple unique $\{(i, a), (j, b)\}$ qui satisfait $c_{ij}(a, b) = 0$ parmi tous les tuples possibles de la fonction de coût. La valeur $a$ est appelée la valeur* strict AC *de $i$.*

Étant donné un WCSP $P$ et un sous-ensemble $S$ de ses variables tel que toutes les variables dans $S$ sont strict AC, nous pouvons résoudre $P$ restreint à $S$ de manière exacte en affectant la valeur strict AC à chaque variable. Cette affectation partielle aura un coût nul. Cette propriété donne une partition naturelle d'un WCSP en l'ensemble des variables strict AC et le reste. Cette bi-partition a été utilisée dans [18] et dans un algorithme raffiné qui a été introduit plus tard [11]. Ces algorithmes exploitent la solvabilité du sous-ensemble des variables strict AC et doivent résoudre seulement la partie plus petite composée des variables non strict AC avec un solveur combinatoire. Une affectation complète est obtenue en combinant les solutions trouvées pour les deux parties. Si les coûts entre les deux parties sont nuls alors l'optimalité est prouvée, sinon les variables strict AC impliquées dans ces coûts non nuls sont ajoutées à la partie combinatoire qui est résolue à nouveau jusqu'à obtenir une preuve d'optimalité du problème global.

Notre première contribution ici est de noter que la propriété strict AC peut être plus forte que nécessaire. En particulier, nous pouvons modifier la deuxième condition comme ci-dessous :

**Definition 7** (Cohérence d'arc stricte revisitée strict* AC)**.** *Une variable $i \in X$ est strict* AC s'il existe une valeur unique $a \in D_i$ telle que $c_i(a) = 0$ et $\forall c_{ij} \in C$, il existe au moins un tuple $\{(i, a), (j, b)\}$ qui satisfait $c_{ij}(a, b) + c_j(b) = 0$. La valeur $a$ est appelée la valeur strict* AC de $i$.*

La différence entre strict* AC et strict AC est que pour strict* AC, la seconde condition nécessite que la valeur témoin apparaisse dans au moins un tuple de coût nul et non dans un unique tuple pour chaque fonction de coût incidente. En contre partie, nous imposons que le coût unaire du support sur l'autre variable doit aussi être nul ($c_j(b) = 0$). Cela permet de reconstruire une affectation partielle de coût nul pour les variables strict* AC en affectant simplement ces variables à leur valeur strict* AC, à l'instar de la propriété strict AC.

Il est intéressant de noter que l'existence d'un tuple unique $\{(i, a), (j, b)\}$ de coût nul pour la propriété strict AC implique que les coûts unaires associés $c_i(a), c_j(b)$ sont nuls également dans le cas où les deux variables $i$ et $j$ sont strict AC. Si toutes les variables voisines de $i$ et $j$ sont aussi strict AC alors les variables $i$ et $j$ sont également strict* AC. L'inverse n'est pas forcément vrai. Par contre l'ensemble des variables à la frontière, *i.e.,* incidentes à des variables de la partie combinatoire, est incomparable d'une propriété à l'autre.

Dans l'exemple du WCSP $P_1$ en Figure 1, la variable $i$ est strict AC et strict* AC, avec pour valeur témoin la valeur $a$. Dans le cas de $P_2$, la variable $i$ est strict* AC mais pas strict AC.

Une difficulté présente pour les deux propriétés, strict AC et strict* AC, est qu'un même minorant $c_\emptyset$ peut être produit par différentes reparamétrisations issues de plusieurs solutions duales du polytope local n'induisant pas le même ensemble de variables strict* AC. Une façon d'y remédier est d'inciter le solveur de programmation linéaire vers des solutions qui maximisent l'ensemble strict* AC [11]. Ici nous proposons une autre méthode, à partir de l'observation suivante.

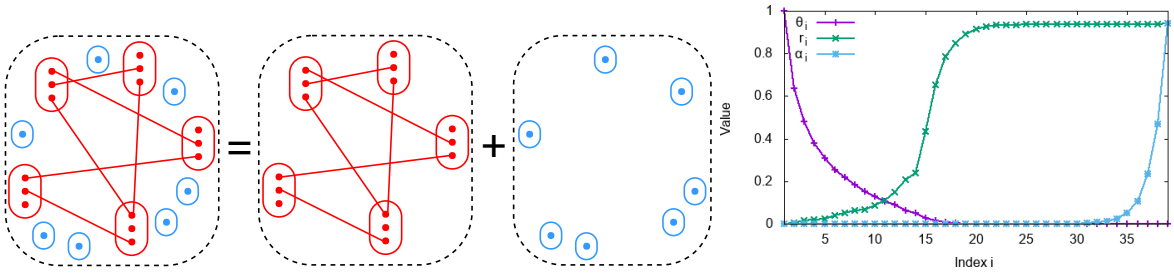
**Proposition 1.** *Soit une instance WCSP $P$ qui est VAC et une variable $i$. S'il existe une unique valeur $a \in \overline{D}_i$ dans $Bool(P)$, alors la variable $i$ est strict* AC avec pour valeur témoin $a$.*

D'après la définition de VAC et $Bool(P)$, nous savons que $Bool(P)$ est arc-cohérent et si une seule valeur reste dans le domaine de $i$ dans $Bool(P)$, elle a nécessairement un coût nul dans $P$ et un support de coût nul dans toutes les variables voisines. A contrario, si une valeur est supprimée dans $Bool(P)$, soit elle a un coût non nul dans $P$, soit il est possible de déplacer un coût non nul vers cette valeur [7]. Voir l'exemple en Figure 1 où nous montrons la fermeture par cohérence d'arc de $Bool(P)$ pour deux instances. Ici, les variables $i$ et $j$ sont strict* AC dans la fermeture $Bool(P_1)$.

Cette observation nous permet de construire un ensemble de variables strict* AC plus grand que celui obtenu en vérifiant la propriété pour une reparamétrisation particulière. Ainsi nous n'avons pas à modifier le solveur de programmation linéaire pour favoriser des solutions duales spécifiques et c'est plus facile à utiliser au sein de l'algorithme VAC qui maintient la fermeture arc-cohérente de $Bool(P)$ explicitement.

**Complexité.** Il est naturel de se demander si la présence d'un grand ensemble de variables strict* AC rend le problème plus facile à résoudre au sens de la complexité paramétrée [10]. Malheureusement, ce n'est pas le cas. Soit Almost-S-AC-WCSP la classe des WCSPs qui sont VAC avec $n - 1$ strict* AC variables.

**Theorème 1.** Almost-S-AC-WCSP *est NP-complet.*




(a) Partition des variables en deux groupes : difficile et facile.

(b) Évolution de $\theta_i$, $r_i$ et $\alpha_i$ en fonction du nombre d'itérations.

FIGURE 2 – strict* AC et l'heuristique fondée sur RINS.

*Démonstration.* L'appartenance à NP est triviale puisque c'est une sous-classe de WCSP. Pour la complétude, nous effectuons une réduction à partir d'un WCSP binaire. Soit $P = \langle X, D, C, k\rangle$ une instance WCSP arbitraire. Nous construisons une instance $P'\langle X \cup X' \cup \{q\}, D, C', k\rangle$ de ALMOST-S-AC-WCSP, où $X'$ est une copie des variables de $X$, avec les mêmes coûts unaires, et $q$ est une variable supplémentaire de domaine $\{a, b\}$ avec $c_q(a) = c_q(b) = 0$. Pour chaque fonction de coût $c_{ij} \in C$, $P'$ a deux fonctions de coût ternaires portant sur $\{i, j, q\}$ et sur $\{i', j', q\}$.

Chaque variable de $P$ a au moins une valeur de coût unaire nul puisque $P$ est VAC. Sans perte de généralité, supposons que cette valeur est $a$ pour toutes les variables. Nous définissons les fonctions de coût ternaires sur les variables de $X$ par $c_{ijq}(a, a, a) = 0$, $c_{ijq}(u, v, a) = k$ quand $u \neq a$ ou $v \neq a$, $c_{ijq}(u, v, b) = c_{ij}(u, v) + 1 \forall u, v$. De même, $c_{i'j'q}(a, a, b) = 0$, $c_{i'j'q}(u, v, b) = k$ quand $u \neq a$ ou $v \neq a$, $c_{i'j'q}(u, v, a) = c_{ij}(u, v) + 1 \forall u, v$.

$P'$ est une instance ALMOST-S-AC-WCSP avec $q$ la variable non-strict* AC. En effet, $c_i(a) = 0$ pour toutes les variables $i \in X \cup X' \cup \{q\}$ et chaque variable $i \in X$ (resp. $i \in X'$) est supportée par l'unique tuple de coût nul $(a, a, a)$ (resp. $(a, a, b)$) dans chaque fonction de coût ternaire. Toutes les autres valeurs appartiennent à des tuples de fonctions ternaires ayant un coût non nul et seront supprimées dans la fermeture par cohérence d'arc de $Bool(P')$.

$P$ a une solution optimale de coût $c$ si et seulement si $P'$ a une solution optimale de coût $c + |C|$. En effet, quand nous affectons $q$ à $a$ ou $b$, le problème se décompose en deux WCSPs binaires indépendants sur $X$ et sur $X'$. L'un deux admet une solution que de $a$ de coût nul et l'autre est identique à $P$ avec un coût supplémentaire de 1 par fonction de coût. □

Bien que cette construction utilise des fonctions de coût ternaires, nous pouvons convertir celles-ci en fonctions binaires en utilisant l'encodage caché [3] qui préserve la cohérence d'arc et donc aussi VAC, démontrant ainsi le théorème dans le cas de WCSPs binaires.

## 4 Heuristique de choix de variable

Dans un algorithme de recherche par séparation et évaluation, l'ordre dans lequel les variables sont affectées a un impact crucial sur les performances. En général, une décision de branchement doit aider le solveur à élaguer rapidement des sous-arbres qui ne contiennent pas de solutions meilleures, en créant des sous-problèmes ayant un minorant augmenté dans toutes les branches [1].

A partir de cette observation et de la connexion entre strict* AC et l'intégralité expliquée en Section 3, nous observons que brancher sur une variable strict* AC $i$ créera une branche qui contiendra la valeur strict* AC $a$ de $i$. Puisque $a$ est la seule valeur du domaine de $i$ dans $Bool(P)$ et que son coût unaire ne change pas lors du branchement, $Bool(P\ |_{i=a})$ est identique à $Bool(P)$, ainsi le minorant ne sera pas augmenté dans cette branche. Il semble donc préférable de ne pas brancher sur les variables strict* AC.

Pour implanter ceci, nous partitionnons les variables suivant la taille de leur domaine dans la fermeture arc-cohérente de $Bool(P)$ : celles ayant un domaine singleton correspondent aux variables faciles en bleu sur la Figure 2a et le reste correspond aux variables difficiles en rouge.

Nous donnons alors la priorité aux variables ayant plus d'une valeur, *i.e.,* les variables en rouge, pour le branchement. Le choix parmi ces variables est fait en utilisant l'heuristique par défaut dans le solveur. Dans le cas de TOULBAR2, utilisé dans nos expérimentations, il s'agit de l'heuristique DOM+WDEG [4] combinée avec l'heuristique last conflict [16].

## 5 Heuristique de production de majorants

Une approche appelée *recherche dans le voisinage induit par la relaxation* (Relaxation Induced Neighborhood Search or RINS) [8] consiste à construire un

sous-problème prometteur en utilisant l'information contenue dans la relaxation continue d'un modèle de programmation linéaire à variables entières (PLNE). L'exploration du sous-problème est formulée à nouveau comme un modèle PLNE et résolu à son tour avec une limite de temps [2]. Pour cela, le sous-problème est construit en affectant les variables qui ont la même valeur dans la meilleure solution trouvée et dans la relaxation courante.

En suivant cette approche, nous proposons une heuristique de production de majorants qui s'exécute en prétraitement. Dans ce cas, nous n'avons pas encore trouvé de meilleure solution et nous affectons seulement les variables qui sont intégrales dans la relaxation : affectation des variables strict* AC à leur valeur témoin et suppression des valeurs dans le reste des variables qui ont été supprimées dans la fermeture par cohérence d'arc de $Bool(P)$. De plus, comme nous allons l'expliquer ensuite, nous exploitons la fermeture par cohérence d'arc construite par des itérations intermédiaires de VAC, en examinant $Bool_\theta(P)$ pour une valeur appropriée de $\theta$.

Cette heuristique RINS peut également s'appliquer durant la recherche à une fréquence donnée, comme par exemple tous les $f$ noeuds de l'arbre de recherche [3]. Dans ce cas, les variables strict* AC sont affectées à leur valeur témoin si et seulement si leur valeur strict* AC est la même que dans la meilleure solution trouvée, sinon nous conservons ces deux valeurs dans le domaine de la variable, à l'instar de l'approche RINS [8].

De la même façon, nous rajoutons dans le domaine des variables non-strict* AC la valeur présente dans la meilleure solution trouvée après avoir réalisé le filtrage des domaines par maintien de cohérence d'arc sur $Bool_\theta(P)$.

Nous avons observé que la qualité des majorants trouvés par cette heuristique dépend fortement de la valeur du seuil $\theta$ dans la construction de $Bool_\theta(P)$. Rappelons d'abord la façon dont est implanté l'algorithme VAC$_\theta$ dans le solveur TOULBAR2 [7]. Cette implémentation ne traite que les WCSP binaires. Premièrement, tous les coûts binaires non nuls $c_{ij}(a,b)$ sont triés par ordre décroissant et repartis dans un nombre fixe de $g$ sous-ensembles de coûts. Les coûts minimums $\theta_i$ de chaque sous-ensemble $i \in \{1,\ldots,g\}$ définissent une séquence de seuils $(\theta_1,\ldots,\theta_g)$. L'algorithme VAC$_\theta$ commence avec le seuil le plus grand $\theta_1$ et itère à ce seuil fixe ses étapes de reparamétrisation jusqu'à ce que le problème $Bool_{\theta_1}(P)$ soit arc-cohérent. Puis il recommence avec le seuil suivant $\theta_{i+1}$. Après avoir terminé avec le seuil $\theta_g$, l'algorithme suit une suite géométrique décroissante avec $\theta_{i+1} = \frac{\theta_i}{2}$ jusqu'à atteindre $\theta_i = 1$.

Plus $\theta_i$ est petit, plus $Bool_{\theta_i}(P)$ est restreint, *i.e.,* les domaines sont réduits. Cela conduit globalement à un ensemble de variables strict* AC globalement croissant suivant les itérations $i$ de VAC$_\theta$. L'observation que nous faisons de manière informelle est que cet ensemble arrête de croître et semble saturer à partir d'un certain seuil, souvent avant d'atteindre $\theta_i = 1$. Cependant même après ce seuil de saturation, l'algorithme continue à réduire les domaines des variables non-strict* AC. Notre idée est de choisir un seuil $\theta$ où le nombre de variables strict* AC est maximisé et où les domaines des variables non-strict* AC sont suffisamment réduits mais pas trop. De cette manière, nous augmentons la taille de la partie facile et diminuons celle de la partie difficile, mais en même temps, nous ne restreignons pas trop les domaines des variables non-strict* AC par rapport au problème original.

En Figure 2b, nous montrons l'évolution du seuil normalisé $\frac{\theta_i}{\theta_1}$, le pourcentage $r_i$ de variables strict* AC et le rapport de ces deux courbes $\alpha_i = \frac{r_i\theta_1}{\theta_i}$ en fonction des itérations allant de $i = 1$ à $i = 39$ pour l'instance cnd1threeL1_1228061 du benchmark Worms. Ici, $r_i$ varie de 0 pour $\theta_1 = 705.867.889$ à 0.94 pour $\theta_{39} = 1$.

La courbe montrant l'évolution du rapport $\alpha$ en Figure 2b est utilisée pour fixer le seuil $\theta$ dans notre heuristique RINS. L'idée est que lorsque le nombre de variables strict* AC sature, le seuil $\theta_i$ normalisé continue à diminuer et donc le rapport $\alpha$ augmente de manière plus rapide, c'est à ce moment que nous identifions le bon seuil au point d'inflexion. En pratique, nous sélectionnons le seuil $\theta_i$ lorsque l'angle de la courbe $\alpha$ atteint 10 degrés. Sur notre exemple, cela survient à l'itération 32, avec $\theta_{32} = 627$.

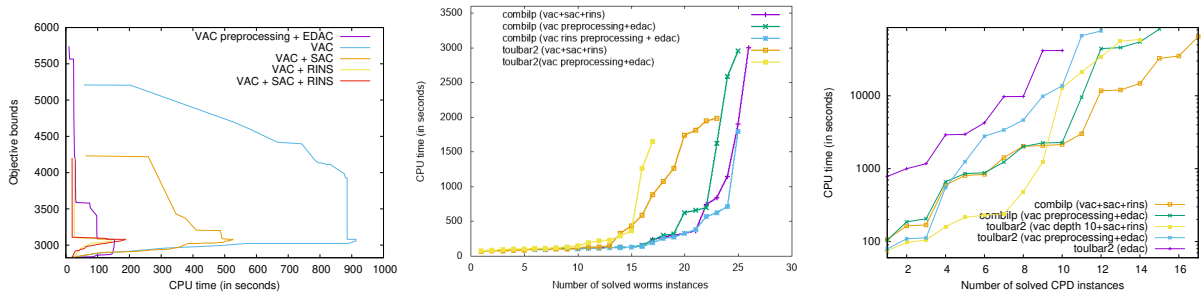
## 6 Résultats expérimentaux

Nous avons implantés les heuristiques strict* AC et RINS dans TOULBAR2 (`github.com/toulbar2/toulbar2` version 1.0.1 avec les options par défaut), un solveur WCSP open-source écrit en C++ et ayant remporté plusieurs compétitions sur les modèles graphiques [4]. Les calculs ont été réalisés sur un seul coeur d'un processeur Intel Xeon à 2.1 GHz et 8 Go de mémoire vive avec un temps limite d'une heure (sauf pour les instances CPD, temps limite fixé à une journée).

### 6.1 Description des benchmarks

Les expérimentations ont porté sur des benchmarks de modèles graphiques probabilistes et déterministes issus de différentes communautés. D'abord, nous avons

2. Dans les expérimentations, nous limitons la recherche à 1000 retours-arrières.
3. Dans les expérimentations, $f = 100$.
4. `www.inra.fr/mia/T/toulbar2`




(a) CELAR graph11 instance. (b) Worms instances. (c) CPD instances.

FIGURE 3 – Comportement des bornes au cours du temps et nombre d'instances entièrement résolues (optimum et preuve d'optimalité) pour un temps de calcul donné sur les familles de problèmes CELAR, Worms et CPD.

considéré une large collection de 3016 instances [12] [5] pour lesquelles nous avons sélectionné les instances binaires. Cela inclut 251 instances MRF des compétitions *UAI 2008 Evaluation* et *Probabilistic Inference Challenge* (PIC 2011) [6], 853 instances en analyse d'images (Computer Vision and Pattern Recognition or CVPR) du benchmark OpenGM2 [7] [14], 503 instances MaxCSP de la compétition *CSP 2008 Competition* [8] et 251 instances d'une librairie de WCSPs (aka, Cost Function Network or CFN) [9].

Nous avons aussi collecté 30 instances Worms [13] [10] où COMBILP est l'état de l'art [11] et 21 instances de conception de protéines (Computational Protein Design or CPD) où TOULBAR2 est l'état de l'art [17] [11].

## 6.2 Analyse des résultats

Dans la Table 1, nous présentons les résultats par famille d'instance en nombre d'instances résolues et en temps CPU moyenné sur les instances résolues. Nous comparons différentes versions de TOULBAR2 avec :

— VAC en prétraitement et EDAC durant la recherche avec l'heuristique de choix de variable par défaut DOM+WDEG (VACpreprocessing+EDAC)
— VAC et RINS en prétraitement et EDAC durant la recherche avec l'heuristique de choix de variable par défaut (VAC+RINSpreprocessing+EDAC)
— VAC en prétraitement et durant la recherche avec l'heuristique de choix de variable par défaut (VAC)
— VAC en prétraitement et durant la recherche avec l'heuristique de choix de variables exploitant strict* AC (VAC+sAC)
— VAC et RINS en prétraitement et VAC seul durant la recherche avec l'heuristique de choix de variables exploitant strict* AC (RINSpreprocessing+VAC+sAC)
— VAC et RINS en prétraitement et durant la recherche avec l'heuristique de choix de variables exploitant strict* AC (VAC+sAC+RINS)

Bien que l'approche VAC en prétraitement et EDAC durant la recherche soit souvent la meilleure option, nous observons des gains de performance avec nos heuristiques sur les familles Worms, CPD, EHI et QCP. L'écart de performance entre VAC+RINS et/ou strict* AC et EDAC vient principalement du temps mis pour maintenir VAC durant la recherche. Si l'on se compare à VAC durant la recherche, alors l'apport de nos heuristiques est le plus souvent positif. En particulier le nombre de noeuds de l'arbre de recherche de VAC+sAC ou VAC+sAC+RINS comparé à VAC durant la recherche est en moyenne réduit sur toutes les familles sauf CFN/Auction, MaxCSP/BlackHole, Composed, Geometric, QCP et MRF/DBN (résultats non communiqués par faute de place). Cependant, dans ces cas défavorables, l'augmentation du nombre de noeuds est inférieur à 40%. Pour le reste des benchmarks, ce nombre est réduit de quelque pourcent à plusieurs ordres de magnitude (CVPR/ColorSeg, MRF/ImageAlignment, ProteinFolding, Segmentation et Worms).

Pour les instances CELAR et ProteinDesign, quand VAC est utilisé en prétraitement, l'ajout de RINS améliore considérablement les temps de résolution. Si nous ajoutons strict* AC et RINS avec VAC durant la recherche, nous pouvons surpasser VAC en prétraitement + EDAC. Voir Figure 3a, l'évolution des bornes au cours du temps pour l'instance CELAR graph11 [5].

Nous avons également comparé TOULBAR2 et COMBILP (qui utilise la même version de TOULBAR2 pour son solveur PLNE interne) avec différentes cohérences

5. genoweb.toulouse.inra.fr/~degivry/evalgm
6. graphmod.ics.uci.edu/uai08/Evaluation/Report/Benchmarks, www.cs.huji.ac.il/project/PASCAL
7. hci.iwr.uni-heidelberg.de/opengm2
8. www.cril.univ-artois.fr/CPAI08
9. forgemia.inra.fr/thomas.schiex/cost-function-library
10. genoweb.toulouse.inra.fr/~degivry/evalgm/CVPR/Worms_uai.tar.xz
11. genoweb.toulouse.inra.fr/~degivry/evalgm/CFN/ProteinDesignUAI2017_wcsp_xz.zip

locales, montrant les améliorations apportées par l'exploitation des heuristiques strict* AC et RINS.

En Figure 3b, nous donnons les résultats pour le benchmark Worms qui est composé de 30 instances. CombiLP résout 25 instances d'après [11] en 1 heure de temps CPU. Ici, nous comparons CombiLP avec les paramètres pris dans [11] (VAC en prétraitement et EDAC durant la recherche) et utilisant notre version de toulbar2. De plus, nous testons l'apport des heuristiques strict* AC et RINS. toulbar2 seul résout 23 instances. Cependant, en mettant notre version de toulbar2 dans CombiLP, nous avons résolu 26 instances, soit une de plus que dans [11]. Bien qu'il soit coûteux de maintenir VAC durant la recherche, c'est la meilleure approche obtenue pour ce problème. En effet, la réduction de la taille de l'arbre de recherche surpasse le coût de maintenir VAC.

La même amélioration est observée pour le benchmark CPD. En Figure 3c, CombiLP utilisant VAC durant la recherche avec strict* AC et RINS résout 17 instances parmi les 21, au lieu de 15 seulement pour CombiLP utilisant VAC en prétraitement + EDAC. Indépendemment de CombiLP, toulbar2 par défaut avec VAC en prétraitement en résout seulement 12. Pour pouvoir exploiter VAC durant la recherche en dehors de CombiLP, sans avoir un coût excessif en temps de calcul pour maintenir VAC, nous avons dû limiter l'application de VAC uniquement lors de l'expansion des noeuds ouverts dans l'algorithme Hybrid-Best First Search utilisé par défaut dans toulbar2 [2] et seulement à une profondeur de recherche inférieure à 10. Dans ce cas, toulbar2 exploitant partiellement VAC avec l'ajout des heuristiques strict* AC et RINS résout 14 instances. Plusieurs instances de taille intermédiaire sont résolues avec un gain en temps d'un ordre de grandeur par rapport à toulbar2 par défaut ou à CombiLP. Pour comparaisons, dans [17], toulbar2 n'utilisant ni VAC ni nos heuristiques n'a résolu que 10 instances avec un temps limite de quatre jours.

## 7 Conclusions

Nous avons revisité la propriété Strict AC de telle façon qu'elle soit plus simple à utiliser au sein d'un algorithme par séparation et évaluation en conjonction avec l'algorithme VAC. Cette nouvelle propriété intègre des informations calculées par VAC sur la relaxation du problème pour l'utiliser dans des heuristiques.

Nous avons présenté deux heuristiques qui exploitent cette information, l'une pour le choix de la prochaine variable à brancher et l'autre pour trouver des bons majorants. Une évaluation expérimentale montre que ces heuristiques peuvent améliorer l'état de l'art sur certaines familles d'instances.

**Remerciements.**

Ce travail a été en partie financé par l'Agence nationale de la Recherche, référence ANR-16-C40-0028. Nous sommes reconnaissant à la plateforme bioinformatique Genotoul de Toulouse pour l'accès au cluster.

## Références

[1] T Achterberg : Scip : solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

[2] D Allouche, S de Givry, G Katsirelos, T Schiex et M Zytnicki : Anytime Hybrid Best-First Search with Tree Decomposition for Weighted CSP. *In Proc. of CP-15*, pages 12–28, Cork, Ireland, 2015.

[3] Fahiem Bacchus, Xinguang Chen, Peter van Beek et Toby Walsh : Binary vs. non-binary constraints. *Artif. Intell.*, 140(1/2):1–37, 2002.

[4] F Boussemart, F Hemery, C Lecoutre et L Sais : Boosting systematic search by weighting constraints. *In Proc. of ECAI-04*, pages 146–150, Valencia, Spain, 2004.

[5] B Cabon, S de Givry, L Lobjois, T Schiex et JP Warners : Radio Link Frequency Assignment. *Constraints*, 4(1):79–89, 1999.

[6] M Cooper et T Schiex : Arc consistency for soft constraints. *Artificial Intelligence*, 154(1-2):199–227, 2004.

[7] M C Cooper, S De Givry, M Sánchez, T Schiex, M Zytnicki et T Werner : Soft arc consistency revisited. *Artificial Intelligence*, 174(7-8):449–478, 2010.

[8] E Danna, E Rothberg et C Le Pape : Exploring relaxation induced neighborhoods to improve mip solutions. *Mathematical Programming*, 102(1):71–90, 2005.

[9] S de Givry, M Zytnicki, F Heras et J Larrosa : Existential arc consistency : getting closer to full arc consistency in weighted CSPs. *In Proc. of IJCAI-05*, pages 84–89, Edinburgh, Scotland, 2005.

[10] R G. Downey et M R. Fellows : *Fundamentals of Parameterized Complexity.* Texts in Computer Science. Springer, 2013.

[11] S Haller, P Swoboda et B Savchynskyy : Exact map-inference by confining combinatorial search with LP relaxation. *In Proc. of AAAI-18*, pages 6581–6588, New Orleans, Louisiana, USA, 2018.

[12] B Hurley, B O'Sullivan, D Allouche, G Katsirelos, T Schiex, M Zytnicki et S de Givry : Multi-Language Evaluation of Exact Solvers in Graphical Model Discrete Optimization. *Constraints*, 21(3):413–434, 2016.

[13] D Kainmueller, F Jug, C Rother et G Myers : Active graph matching for automatic joint segmentation and annotation of c. elegans. *In Medical Image Computing and Computer-Assisted Intervention (MICCAI-14)*, pages 81–88, Boston, USA, 2014.

[14] J Kappes, B Andres, F Hamprecht, C Schnörr, S Nowozin, D Batra, S Kim, B Kausler, T Kröger, J Lellmann, N Komodakis, B Savchynskyy et C Rother : A comparative study of modern inference techniques for structured discrete energy minimization problems. *Int. J. of Computer Vision*, 115(2):155–184, 2015.

[15] V Kolmogorov : Convergent tree-reweighted message passing for energy minimization. *IEEE transactions on pattern analysis and machine intelligence*, 28(10):1568–1583, 2006.

[16] C Lecoutre, L Sais, S Tabary et V Vidal : Reasoning from last conflict(s) in constraint programming. *Artificial Intelligence*, 173(18):1592–1614, 2009.

[17] A Ouali, D Allouche, S de Givry, S Loudni, Y Lebbah, F Eckhardt et L Loukil : Iterative Decomposition Guided Variable Neighborhood Search for Graphical Model Energy Minimization. *In Proc. of UAI-17*, pages 550–559, Sydney, Australia, 2017.

[18] B Savchynskyy, JH Kappes, P Swoboda et C Schnörr : Global map-optimality by shrinking the combinatorial search area with convex relaxation. *In Proc. of NIPS-13*, pages 1950–1958, Lake Tahoe, Nevada, USA, 2013.

| Problem/$s$/$d$ | VAC preprocessing + EDAC | VAC and RINS preprocessing + EDAC | VAC | VAC + sAC | RINS preprocessing + VAC + sAC | VAC + sAC + RINS |
|---|---|---|---|---|---|---|
| RFC/251/300 | 248 (92.40) | 246 (97.79) | 242 (100.94) | 243 (116.87) | 244 (140.95) | 242 (126.01) |
| Auction/170/2 | 170 (34.77) | 170 (34.78) | 170 (50.04) | 170 (68.45) | 170 (68.00) | 170 (68.98) |
| CELAR/16/44 | 14 (345.35) | 13 (72.85) | 13 (524.02) | 14 (416.11) | 13 (276.99) | 13 (469.92) |
| ProteinDesign/10/198 | 9 (4.90) | 9 (1.89) | 9 (12.21) | 9 (9.08) | 9 (2.59) | 9 (4.50) |
| Warehouse/55/300 | 55 (220.45) | 54 (318.16) | 50 (179.95) | 50 (217.09) | 52 (369.37) | 50 (252.37) |
| CVPR/853/20 | 742 (42.25) | 742 (35.45) | 724 (6.35) | 743 (40.40) | 742 (34.59) | 742 (38.03) |
| ColorSeg/21/12 | 17 (1322.43) | 17 (1086.87) | 3 (1198.70) | 17 (1244.88) | 17 (1093.99) | 17 1142.04 |
| InPainting/4/4 | 2 (1083.39) | 2 (983.25) | 2 (419.48) | 3 (952.89) | 2 (520.26) | 2 (1226.70) |
| Matching/4/20 | 4 (3.46) | 4 (4.50) | 4 (28.21) | 4 (32.66) | 4 (42.54) | 4 (39.72) |
| ObjectSeg/5/8 | 4 (1658.67) | 4 (1447.96) | 0 (-) | 4 (1454.08) | 4 (1452.63) | 4 (1533.74) |
| SceneDecomp/715/8 | 715 (0.07) | 715 (0.07) | 715 (0.07) | 715 (0.07) | 715 (0.07) | 715 (0.07) |
| MaxCSP/503/50 | 419 (139.75) | 419 (138.91) | 422 (217.74) | 421 (196.67) | 421 (196.54) | 419 (190.64) |
| BlackHole/37/50 | 10 (0.08) | 10 (0.08) | 10 (0.06) | 10 (0.13) | 10 (0.13) | 10 (0.15) |
| Coloring/22/6 | 17 (7.07) | 17 (6.80) | 17 (11.45) | 17 (11.06) | 17 (10.90) | 17 (11.78) |
| Composed/80/10 | 80 (0.12) | 80 (0.12) | 80 (0.83) | 80 (1.44) | 80 (1.42) | 80 (1.44) |
| EHI/200/7 | 198 (231.73) | 198 (229.34) | 200 (344.84) | 200 (296.08) | 200 (298.64) | 200 (297.25) |
| Geometric/100/20 | 96 (84.64) | 96 (85.24) | 91 (115.43) | 91 (143.44) | 91 (139.96) | 91 (183.05) |
| Langford/4/29 | 2 (0.36) | 2 (0.35) | 2 (0.47) | 2 (1.29) | 2 (1.12) | 2 (1.15) |
| QCP/60/9 | 16 (275.93) | 16 (280.32) | 22 (552.39) | 21 (486.64) | 21 (474.97) | 19 (181.72) |
| MRF/297/503 | 213 (37.32) | 213 (38.26) | 209 (56.55) | 208 (36.60) | 209 (49.79) | 209 (48.29) |
| DBN/108/2 | 82 (90.22) | 82 (93.86) | 78 (130.98) | 77 (87.97) | 78 (126.87) | 78 (123.00) |
| ImageAlignment/10/93 | 10 (2.46) | 10 (2.28) | 10 (4.28) | 10 (2.39) | 10 (2.39) | 10 (2.28) |
| ProteinFolding/21/503 | 21 (23.91) | 21 (19.29) | 21 (72.62) | 21 (37.65) | 21 (21.93) | 21 (21.52) |
| Segmentation/100/21 | 100 (0.24) | 100 (0.25) | 100 (0.35) | 100 (0.24) | 100 (0.25) | 100 (0.25) |
| Worms/30/128 | 17 (308.14) | 19 (472.92) | 14 (606.06) | 22 (668.48) | 23 (579.13) | 23 (585.56) |

TABLE 1 – Nombre d'instances WCSP binaires résolues en moins d'une heure. En parenthèses, temps CPU en secondes pour les instances résolues. La colonne 1 contient le nom des familles d'instances suivi du nombre d'instances par famille ($s$) et de la taille du plus grand domaine ($d$). Les familles d'instances CVPR/ChineseChars/100/2, MatchingStereo/2/20, PhotoMontage/2/7, MRF/Grid/21/2 et ObjectDetection/37/21 pour lesquelles aucune instance n'est résolue quelque soit la méthode ont été retirées de cette table.

# Introduction de contraintes structurelles pour la résolution du problème du voyageur de commerce

**Nicolas Isoart*** **Jean-Charles Régin**
Université Côte d'Azur, CNRS, I3S, France
nicolas.isoart@gmail.com jcregin@gmail.com

**Résumé**

Plusieurs modèles basés sur la programmation par contraintes ont été proposés pour résoudre le problème du voyageur de commerce (TSP). Les plus efficaces, telle que la weighted circuit constraint (WCC), s'appuient principalement sur la relaxation lagrangienne du TSP, basée sur la recherche d'arbres recouvrants ou plus précisément de "one-trees". Le défaut de cette méthode est qu'elle n'inclut pas assez de contraintes structurelles et se base presque exclusivement sur les coûts des arêtes. L'objectif de cet article est de corriger ce défaut. Aussi, nous cherchons des motifs empêchant l'existence d'un cycle hamiltonien dans un graphe ou, à l'inverse, des motifs imposant que certaines arêtes soient dans l'ensemble de solutions du TSP. Nous proposons un propagateur basé sur la recherche de $k$-cutsets pour la contrainte de cycle hamiltonien. Sa combinaison avec la contrainte WCC permet d'obtenir, pour la résolution du TSP, des gains d'un ordre de magnitude pour le nombre de backtracks ainsi qu'une forte réduction du temps de calcul.

**Abstract**

Several models based on constraint programming have been proposed to solve the travelling salesman problem (TSP). The most efficient, such as weighted circuit constraint (WCC), mainly rely on the Lagrangian relaxation of TSP, based on the search for trees covering or more precisely "one-trees". The weakness of this method is that it does not include enough structural constraints and is based almost exclusively on edge costs. The purpose of this article is to correct this defect. Also, we are looking for reasons preventing the existence of a Hamiltonian cycle in a graph or, conversely, reasons requiring that certain edges be in the TSP solution set. We propose a propagator based on the research of $k$-cutsets for the Hamiltonian cycle constraint. Its combination with the WCC constraint allows us to obtain, for the resolution of the TSP, gains of an order of magnitude for the number of backtracks as well as a strong reduction of the computation time.

## 1 Introduction

Le problème du voyageur de commerce ou TSP (travelling salesman problem) est un problème NP-Difficile ou NP-Complet selon les variantes. Il a de nombreuses applications et a été motivé par des problèmes concrets, tels que le parcours des bus scolaires, la logistique, le routage, etc. A peu près tous les types de méthodes de résolution (MIP, SAT, CP, algorithme évolutionnaire, etc.) ont été testés pour le résoudre. Lorsque le graphe est Euclidien, le programme le plus efficace est le logiciel ad hoc Concorde [1]. Malheureusement, il ne peut pas prendre en compte des contraintes additionnelles. Or, en pratique, elles sont bien souvent nécessaires pour un grand nombre de problèmes, comme le Pickup & Delivery, des problèmes de transport à la demande (Dial and Ride), les vendanges et moissonnages automatiques, etc. La résolution du TSP est difficile puisqu'il s'agit de trouver un unique cycle passant par tous les sommets d'un graphe, tel que la somme des coûts des arêtes qui le composent soit minimale. On peut assez facilement modéliser le fait que chaque sommet appartienne à un cycle. En effet, il suffit que chaque sommet ait au moins deux voisins distincts, autrement dit, chaque sommet doit être l'extrémité d'au moins deux arêtes. On obtient ce résultat en modélisant notre problème comme un problème d'affectation, qui se résout en temps polynomial. Malheureusement, ce modèle n'est pas suffisant pour obtenir un seul et unique cycle dans le graphe. En effet, l'affectation correspond à un recouvrement des sommets par un ensemble de cycles disjoints. Pour ce modèle, on obtient des solutions où chaque sommet appartient à un cycle, mais pas à un seul et unique cycle. Habituellement, pour parvenir

*Papier doctorant : Nicolas Isoart est auteur principal.

à ce résultat on impose que le sous-graphe engendré par les arêtes sélectionnées soit connexe. C'est cette combinaison des deux aspects qui rend le problème aussi difficile. Contrairement à l'approche précédente, on peut construire un modèle basé sur la notion de sous-graphe connexe. C'est exactement l'idée de Held et Karp [6, 7] qui ont représenté cette notion par un 1-tree, qui est un arbre recouvrant particulier. La notion de cycle est alors introduite en imposant que chaque nœud ait un degré 2 dans le 1-tree. Puis, ils ont utilisé une relaxation lagrangienne en relâchant la contrainte de degré. Plus précisément, ils résolvent une succession de problèmes du minimum spanning tree (polynomial) en ajoutant la contrainte de degré à la fonction objectif tel que si un nœud a un degré $> 2$, on diminue le coût des arêtes voisines, et si le degré est $< 2$ on augmente le coût des arêtes voisines pour obtenir une convergence vers la solution optimale.

Comme mentionné dans [3], la relaxation Lagrangienne proposée par Held et Karp semble être la meilleure : "The best approach regarding the number of instances solved and quality of the bound is the Held and Karp's filtering" [3]. D'ailleurs, la *weighted circuit constraint* (WCC) est essentiellement basée sur cette relaxation.

Nous proposons dans ce papier d'améliorer la WCC en lui ajoutant des méthodes de résolutions de cycles hamiltoniens (=TSP sans les coûts). Pour cela, nous nous inspirons des travaux de Cohen et Coudert sur la structure des cycles hamiltoniens effectués pour le FHCP Challenge [5]. La figure 1 présente un exemple où la structure du graphe est importante pour la recherche de cycle hamiltonien. Il n'existe pas de cycle hamiltonien dans ce graphe, car il est impossible de trouver un cycle visitant tous les sommets ne passant qu'une seule fois par le nœud C. On dit d'un tel graphe qu'il est 1-connexe : il existe un sommet dans le graphe tel que sa suppression le déconnecte. On peut donc définir une nouvelle contrainte structurelle : si un graphe est 1-connexe, alors il ne contient pas de cycle hamiltonien. Dans la pratique, nous n'allons que très rarement avoir un graphe 1-connexe, alors nous proposons d'étendre l'idée de cette contrainte aux arêtes et d'étudier les graphes $k$-arete-connexe pour $k > 1$. Nous étudierons particulièrement les valeurs $k = 2$ et $k = 3$.

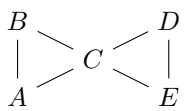



FIGURE 1 – Graphe papillon.

Cet article est organisé de la manière suivante : tout d'abord, on rappelle certains concepts de la théorie des graphes. Ensuite, nous définissons formellement les contraintes structurelles utilisées dans notre méthode de résolution du TSP. Puis, nous définissons une nouvelle structure de données appelée cycled spanning tree, elle est utilisée pour définir un nouvel algorithme visant à exploiter les contraintes structurelles. La dernière partie vérifie expérimentalement l'intérêt de notre méthode. Pour finir, nous concluons.

## 2 Préliminaires

### 2.1 Définitions

Les définitions traitant la théorie des graphes sont issues de [10].

Un **graphe orienté** $G = (X, U)$est composé d'un **ensemble de sommets** X et d'un **ensemble d'arcs** U, où chaque arc $(u, v)$ est une paire ordonnée de sommets distincts. On notera $X(G)$ l'ensemble des sommets de $G$ et $U(G)$ l'ensemble des arcs de $G$. Le **coût** d'un arc est la valeur associée à l'arc. Un **graphe non orienté** est un graphe orienté tel que pour chaque arc $(u, v) \in U$, $(u, v) = (v, u)$. Si $G_1 = (X_1, U_1)$ et $G_2 = (X_2, U_2)$ sont des graphes, $G_1$ est un **sous-graphe** de $G_2$ si $V_1 \subseteq V_2$ et $U_1 \subseteq U_2$. Un **chemin** allant du nœud $v_1$ au nœud $v_k$ dans $G$ est une liste de nœuds $[v_1, ..., v_k]$ telle que $(v_i, v_{i+1})$ est une arête pour $i \in [1..k-1]$. Le chemin est **élémentaire** si tous ses nœuds sont distincts. Un chemin est un **cycle** si $k > 1$ et $v_1 = v_k$. Un cycle est **hamiltonien** si $[v_1, ..., v_{k-1}]$ est un chemin élémentaire et contient tous les sommets de U. La **longueur** d'un chemin $p$, noté $length(p)$, est la somme des coûts des arêtes contenues dans $p$. Pour un graphe $G$, une solution au **problème du voyageur de commerce** ou **travelling salesman problem (TSP)** dans $G$ est un cycle hamiltonien $HC \in G$ minimisant $length(HC)$. Un graphe non orienté $G$ est **connecté ou connexe** s'il existe un chemin entre chaque paire de sommets, sinon il est **déconnecté**. Les sous-graphes maximaux connectés de $G$ sont ses **composantes connexes**. Un **arbre** est un graphe connexe et sans cycle. Un arbre $T = (X', U')$ est un **arbre couvrant** de $G = (X, U)$ si $X' = X$ et $U' \subseteq U$. Les arêtes de $U'$ sont les **arêtes de l'arbre** $T$ et les arêtes de $U - U'$ sont les **arêtes hors de l'arbre** $T$. Un **isthme** est une arête telle que sa suppression augmente le nombre de composantes connexes. Une partition des sommets de $G = (X, U)$ telle que $S \subseteq X$ et $T = X - S$ est une **coupe ou cut**. L'ensemble des arêtes $(u, v) \in U$ ayant $u \in S$ et $v \in T$ est la **cutset** de la coupe $(S, T)$. Une **$k$-cutset** est une cutset de taille $k$.

### 2.2 HCWME : Le problème du cycle hamiltonien avec des arêtes obligatoires

Les algorithmes de résolution de TSP tendent à :

• Éliminer des arêtes ne pouvant pas appartenir à la solution optimale.

• Définir des arêtes appartenant à la solution optimale, appelées arêtes obligatoires.

Du fait que chacune des solutions du TSP soit un cycle hamiltonien (HC), l'ensemble des solutions du TSP est un sous-ensemble des solutions du problème du cycle hamiltonien (HCP).

**Propriété 1.** *Soit $G = (X, U)$. Si $a \in U$ appartient à toutes les solutions de HCP(G), alors a appartient nécessairement à toutes les solutions de TSP(G).*

**Propriété 2.** *Soit $G = (X, U)$. Si HCP(G) n'a pas de solution, alors TSP(G) n'a pas de solution.*

Afin d'utiliser les propriétés 1 et 2, nous formulons le problème Hamiltonian Cycle With Mandatory Edges (HCWME) tel que :
**DONNÉES :** Un graphe $G = (X, U)$ et un ensemble d'arêtes obligatoires $M \subseteq U$.
**PROBLÈME :** Existe-il un cycle hamiltonien dans $G$ passant par toutes les arêtes de $M$ ?

Ce problème nous permet d'exploiter le fait qu'un solveur de contraintes définit des arêtes obligatoires. La propriété 1 permet de définir de nouvelles arêtes obligatoires à partir d'existantes. Aussi, la propriété 2 permet de détecter si étant donné un graphe, il existe une solution du TSP dans celui-ci.

### 2.3 Complexité du problème HCWME

Soit $G = (X, U)$. Le problème du $HCWME(G, M)$ se réduit au $HCP(G)$. Si on résout $HCWME(G, M)$ avec $M = \emptyset$ alors on résout $(HC(G))$. On vérifie en temps polynomial si une solution donnée $G' = (X', U')$ résout le problème $HCWME(G, M)$. Pour cela, il faut que $X'$ soit égal à $X$ et que $M \subseteq U'$ : vérification polynomiale. Puis, il faut vérifier que $G'$ forme un cycle hamiltonien : il doit être connexe et chaque sommet doit avoir exactement 2 voisins, vérification polynomiale [10]. Comme le problème du cycle hamiltonien est un problème NP-Complet, le problème $HCWME(G, M)$ est NP-Complet.

Sans perte de généralité, on note $G = (X, U)$, $n = |X|$, $m = |U|$, $M \subseteq U$ l'ensemble des arêtes obligatoires et supposons $G$ connexe pour la suite.

## 3 Contraintes structurelles

### 3.1 Présentation de contraintes structurelles

Lorsque ce n'est pas spécifié, une $k$-cutset est maximale, c'est-à-dire qu'il n'existe pas de sous-ensemble de la $k$-cutset déconnectant le graphe.

**Proposition 1.** *Soit une $k$-cutset dans un graphe, alors tout cycle hamiltonien emprunte un nombre pair et strictement positif d'arêtes de la $k$-cutset.*

*Démonstration.* Soit $G = (X, U)$ tel que $S \subset X$ et $T = \{X - S\}$. Pour trouver un cycle hamiltonien dans $G$ il est nécessaire que $S$ et $T$ soient connectés, autrement dit qu'ils aient une cutset de taille strictement positive. Du fait que l'on cherche un cycle, si on emprunte un chemin allant de $S$ à $T$ alors on doit emprunter un chemin disjoint allant de $T$ à $S$, ce qui donne un nombre pair de chemins. □

Soient $G = (X, U)$, M ⊆ U et $\mathcal{P} = HCMWE(G, M)$. D'après la proposition 1, on définit les propriétés 3, 4, 5, 6 et 7.

**Propriété 3.** *S'il existe une 2-cutset dans $G$, alors les deux arêtes notées $a_1, a_2$ de la cutset deviennent obligatoires : $M \leftarrow M + \{a_1, a_2\}$.*

**Propriété 4.** *S'il existe dans $G$ une $k$-cutset avec $k$ impair contenant $k - 1$ arêtes obligatoires, alors l'arête non obligatoire notée $a$ est supprimée car elle ne peut pas appartenir à un cycle hamiltonien : $E \leftarrow E - \{a\}$.*

**Propriété 5.** *S'il existe dans $G$ une $k$-cutset avec $k$ pair contenant $k - 1$ arêtes obligatoires, alors l'arête non obligatoire notée $a$ le devient : $M \leftarrow M + \{a\}$.*

**Propriété 6.** *S'il existe dans $G$ une 1-cutset, alors $\mathcal{P}$ n'a pas de solution.*

**Propriété 7.** *S'il existe dans $G$ une $k$-cutset avec $k$ impair contenant $k$ arêtes obligatoires, alors $\mathcal{P}$ n'a pas de solution.*

**Définition 1.** *On dit que deux problèmes $\mathcal{P}$ et $\mathcal{P}'$ sont équivalents si leurs ensembles de solutions sont en bijection. On note alors $\mathcal{P}' = \mathcal{P}$.*

**Corollaire 1.** *Soit $\mathcal{P}' = \mathcal{P}$. Si une ou plusieurs des propriétés 3, 4, 5, 6 ou 7 sont appliquées sur $\mathcal{P}$, alors $\mathcal{P}$ et $\mathcal{P}'$ restent équivalents.*

*Démonstration.* Immédiat par la proposition 1. □

On note $\boldsymbol{a}^* \in U$ une arête obligatoire.

### 3.2 Exemple

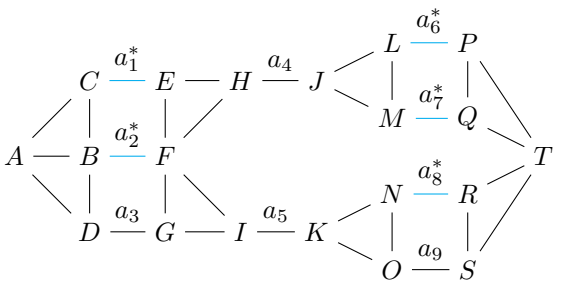



FIGURE 2 – Graphe $G_1$.

D'après les propriétés 3, 4 et 5, on peut supprimer certaines arêtes de la figure 2 et en rendre obligatoires :

• $\{a_4, a_5\}$ forme une 2-cutset : si on veut relier la partie gauche de H I à la partie droite de J K par un cycle il faut obligatoirement prendre (H,J) et (I,K) donc $a_4$ et $a_5$ deviennent obligatoires.

• $\{a_1^*, a_2^*, a_3\}$ forme une 3-cutset avec $\{a_1^*, a_2^*\}$ obligatoires : c'est une cutset de taille impaire avec un nombre pair d'arêtes obligatoires. Alors on peut supprimer $a_3$, car en la choisissant la cutset deviendrait composée uniquement d'arêtes obligatoires et de taille impaire.

• $\{a_6^*, a_7^*, a_8^*, a_9\}$ forme une 4-cutset avec $\{a_6^*, a_7^*, a_8^*\}$ obligatoires : c'est une cutset de taille paire avec 3 arêtes obligatoires. Si on ne rend pas $a_9$ obligatoire, alors la cutset n'aura jamais un nombre pair d'arêtes obligatoires, elle est donc nécessaire.

La figure 3 représente l'application des propriétés 3, 4 et 5 sur $G_1$.

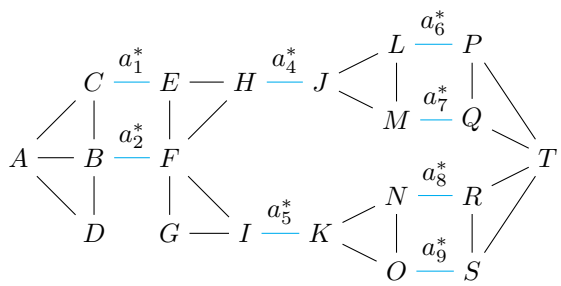



FIGURE 3 – Application des propriétés 3, 4 et 5 sur $G_1$.

D'après la propriété 7 il ne peut pas exister de cycle hamiltonien dans la figure 4 :

• $\{a_4\}$ est une 1-cutset : il ne peut pas exister de cycle hamiltonien reliant $\{I, J, K\}$ au reste du graphe.

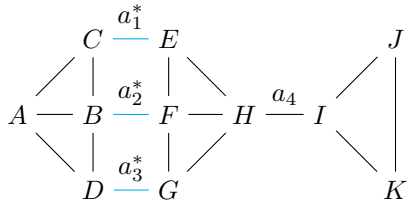



FIGURE 4 – Graphe $G_2$.

• $\{a_1^*, a_2^*, a_3^*\}$ forme une 3-cutset où les trois arêtes sont obligatoires : on veut un cycle hamiltonien passant par ces trois arêtes mais il ne peut pas en exister.

Nous allons maintenant montrer comment appliquer les propriétés 3, 4, 5, 6 ou 7. Etant donné que ces propriétés sont basées sur la taille des cutsets, on peut légitimement se demander combien de cutsets un graphe peut posséder.

**Propriété 8.** *Le nombre de cutset d'un graphe de $n$ sommets est $2^n$.*

*Démonstration.* Soit $G = (X, U)$. N'importe quelle partie $S \subseteq X$ forme une coupe $(S, T)$ où $T = X - S$ avec une cutset. La taille de l'ensemble des parties de $S \subseteq X$ est $2^n$, il y a donc $2^n$ cutsets. □

Dans le cas du graphe non orienté, on remarque qu'une coupe $(S, T)$ a la même cutset qu'une coupe $(T, S)$ : on peut aller par les mêmes chemins de $S$ à $T$ et de $T$ à $S$. Le nombre de cutsets différentes est alors $2^n/2 = 2^{n-1}$.

Dans le but d'intégrer cette idée à un solveur de contraintes, on va donc ne chercher que certaines $k$-cutsets et appliquer les propriétés 3, 4, 5, 6 ou 7. Si on est dans le cas d'un graphe très dense, alors on a peu de chance de satisfaire les propriétés 3, 4, 5, 6 ou 7 pour une petite valeur de $k$. Il n'apparaît non plus pas très raisonnable d'appliquer ces propriétés avec une valeur de $k$ élevée pour au moins deux raisons :

• La complexité des algorithmes de $k$-cutset augmente avec $k$ parce que ce sont des algorithmes d'énumérations [12].

• La relation entre la taille de la cutset et le nombre d'arêtes obligatoires est forte. Plus $k$ augmente et moins on a de chance de satisfaire une des propriétés 3, 4, 5, 6 ou 7. En conséquence, nous proposons d'étudier $k = 1, 2$ et 3 avec les comportements suivants :

• Les 1-cutsets : provoquer une erreur.

• Les 2-cutsets : rendre obligatoire les arêtes de la 2-cutset.

• Les 3-cutsets : considérer uniquement les 3-cutsets avec au moins 2 arêtes obligatoires. Si elle contient une

arête non obligatoire, alors il faut la supprimer, sinon une erreur est provoquée.

## 4 L'algorithme

Nous devons être capables de trouver les $k$-cutsets avec $k = 1, 2, 3$ et deux arêtes obligatoires pour $k = 3$. Si on décompose le problème, trouver les 1-cutset, qui sont en fait des isthmes, peut être fait avec l'algorithme de Tarjan [9] en $O(m + n)$ ; trouver des 2-cutset peut être fait avec l'algorithme de Tsin [11] en $O(m + n)$. L'avantage de l'algorithme de Tsin est qu'il permet aussi de trouver les isthmes, on peut alors gérer $k = 1, 2$ uniquement avec ce dernier. Il nous faut maintenant gérer le cas de $k = 3$ avec au moins deux arêtes obligatoires.

D'après la définition de coupe, si on supprime une arête d'une $k$-cutset, alors elle devient une $(k - 1)$-cutset. On peut alors proposer un premier algorithme simple :

Pour chaque arête obligatoire $a^* \in M$, on cherche les 2-cutsets de $G - \{a^*\}$. De cette façon, chacune des 2-cutset trouvées forme une 3-cutset avec au moins une arête obligatoire. Il suffit alors de conserver uniquement les 3-cutsets avec 2 arêtes obligatoires.

On peut réduire le nombre d'arêtes obligatoires considérées. Pour cela, nous allons construire une structure particulière, nommée CST.

### 4.1 CST : Cycled Spanning Tree

Soit $G = (X, U)$. Un CST est un sous-graphe connexe de G tel que chacune de ses arêtes appartient à un cycle dans le CST.

Une manière de construire un CST est de calculer un arbre couvrant $T$, puis d'ajouter certaines arêtes à $T$ jusqu'à ce que toutes les arêtes, celles de T et celles hors de T, appartiennent à un cycle du CST. Ainsi, on introduit l'idée de Cycled Spanning Tree.

**Propriété 9.** *Soit $T$ un arbre couvrant de $G$. Toute arête qui n'est pas dans $T$ appartient à un cycle composé de cette arête et uniquement d'arêtes de $T$.*

*Démonstration.* Par définition de l'arbre couvrant, $\forall i, j \in X$, il existe un chemin élémentaire $p = [i, ..., j]$ dans T. Choisissons une arête dans $G - T$, notée $(i, j)$. Ajouter $(i, j)$ à $p$ donne $p = [i, ..., j, i]$, par définition p est maintenant un cycle où $(i, j)$ est la seule arête $\notin T$. □

**Définition 2.** *Soit $G' = (X', U')$ un sous-graphe de $G$. On dit que $G'$* **supporte** *les arêtes de $G" = (X, U - U')$ si chacune des arêtes de $G"$ forment un cycle avec uniquement des arêtes de $G'$.*

Cela signifie qu'un arbre couvrant supporte toutes les arêtes qui ne sont pas dans $T$. Mais qu'en est-il des arêtes de $T$ ?

Dans ce cas il nous faut d'autres arêtes pour les supporter. On va donc ajouter des arêtes à notre arbre couvrant. On prend une arête $a_T \in T$ puis on cherche une autre arête $\notin T$ permettant d'introduire dans $T$ un cycle contenant $a_T$. Puis on contracte toutes les arêtes du cycle dans $T$ contenant $a_T$. On répète jusqu'à avoir contracté toutes les arêtes de T. À la fin on aura ajouté au plus $n - 1$ arêtes, puisqu'on peut considérer individuellement chaque arête de $T$ et que pour chaque arête, l'ajout d'une arête suffit. La nouvelle structure obtenue est le CST, elle contient au plus $2n - 2$ arêtes.

Sans perte de généralité, on admet $G$ connexe et sans isthme, il existe alors nécessairement un CST dans $G$.

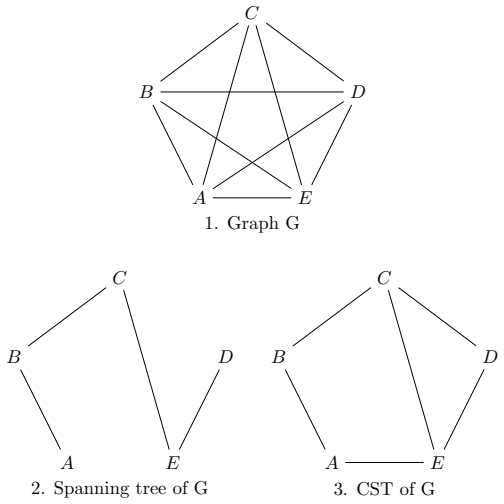



FIGURE 5 – Exemple de construction d'un CST.

**Corollaire 2.** *Soit $k > 1$. S'il existe une $k$-cutset dans $G$, alors au moins deux arêtes de la cutset sont dans le CST.*

*Démonstration.* Par construction, le CST est connexe et couvre le graphe par des cycles. Comme un cycle à une coupe $\geq 2$, chaque $k$-cutset dans G avec $k > 1$ ont une coupe de taille $\geq 2$. □

**Corollaire 3.** *S'il existe une 3-cutset contenant au moins deux arêtes obligatoires, alors au moins une arête obligatoire appartient au CST.*

*Démonstration.* Immédiat par le corollaire 2. □

Via le corollaire 3, on peut améliorer l'algorithme simple en réduisant le nombre d'arêtes obligatoires considérées. L'algorithme devient : pour chaque arête

obligatoire $a^*$ du CST, appelée **arête d'identification**, chercher les 2-cutsets appartenant à $G - \{a^*\}$. Pour chaque 3-cutset trouvée, on obtient soit une 3-cutset avec trois arêtes obligatoires, soit une 3-cutset avec deux arêtes obligatoires ou alors une 3-cutset avec une arête obligatoire. Puis, il suffit de leur appliquer les propriétés 3, 4, 5, 6 et 7.

Puisque les arêtes obligatoires hors du CST ne sont pas considérées comme arêtes d'identifications et qu'on choisit à la construction les arêtes présentes dans le CST, on a intérêt à minimiser son nombre d'arêtes obligatoires.

### 4.2 Amélioration supplémentaire

L'algorithme proposé dépend fortement du nombre d'arêtes d'identifications. D'après le corollaire 2, si deux arêtes appartiennent à la même 2-cutset et qu'elles sont obligatoires, alors ce sont des arêtes d'identifications. Cependant, lorsqu'on a cherché les 3-cutsets avec une arête d'identification, il n'est pas nécessaire de répéter la recherche pour toutes les arêtes formant une 2-cutset avec celle-ci. Plus précisément, le problème de la recherche des 3-cutsets avec $a^*$ comme arête d'identification a le même ensemble de solutions que le problème de la recherche des 3-cutsets avec chacune des arêtes formant une 2-cutset avec $a^*$. La figure 6 l'illustre bien puisque la 2-cutset est un chemin.

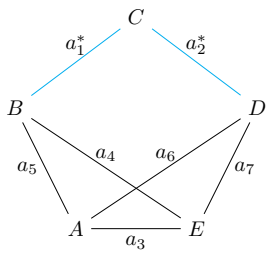



FIGURE 6 – $\{a_1^*,a_2^*\}$ est une 2-cutset. $\{a_1^*,a_4,a_5\}$ et $\{a_1^*,a_6,a_7\}$ sont les 3-cutsets incluant $a_1^*$. $\{a_2^*,a_4,a_5\}$ et $\{a_2^*,a_6,a_7\}$ sont les 3-cutsets incluant $a_2^*$. .

Afin de généraliser, on définit la propriété 10.

**Propriété 10.** *Soient $S_1$ une $k$-cutset et $S_2$ une 2-cutset telles que $k > 1$ et $S_2 \not\subseteq S_1$, si $\exists a \in S_1$ tel que $a \in S_2$ alors $(S_1 \cup S_2) - \{a\}$ forme une $k$-cutset.*

*Démonstration.* Soient $S_2 = \{a_1, a_2\}$ une 2-cutset et $a_1 \in S_1$. Supprimer la cutset $S_1$ du graphe le déconnecte en deux composantes connexes. Par la définition de coupe, $S_2 - \{a_1\} = \{a_2\}$ est un isthme et déconnecter $a_2$ augmente le nombre de composantes connexe à trois. Si on reconnecte $a_1$, G est déconnecté en deux composantes connexes, sa cutset est $(S_1 - \{a_1\}) \cup \{a_2\} = (S_1 - \{a_1\}) \cup (S_2 - \{a_1\}) = (S_1 \cup S_2) - \{a_1\}$. Comme $S_1$ est une $k$-cutset, il n'existe pas de sous-ensemble de celle-ci déconnectant le graphe autre que la $k$-cutset elle même. Si $(S_1 \cup S_2) - \{a_1\}$ déconnecte le graphe alors c'est une $k$-cutset car on supprime et on ajoute une arête dans un ensemble de cardinalité initial $k$. □

Soient $S_1$ une 3-cutset, $S_2 = \{a_1, a_2\}$ et $S_3$ deux 2-cutsets distinctes. La propriété 10 permet d'améliorer notre algorithme car le nombre d'arêtes d'identifications est réduit. On a :

- Si $a_1 \in S_1$, alors $(S_1 - \{a_1\}) \cup \{a_2\}$ est une 3-cutset.
- Si $a_1 \in S_3$, alors $(S_3 - \{a_1\}) \cup \{a_2\}$ est une 2-cutset.

Ainsi, on considère comme arêtes d'identification, les arêtes obligatoires du CST n'appartenant à aucune 2-cutset et pour l'ensemble des 2-cutsets de $G$, son sous-ensemble qui maximise sa cardinalité tel qu'il n'existe pas de combinaison de celui-ci formant une 2-cutset.

Pour ne pas avoir d'incohérence, il faut chercher toutes les 2-cutsets avant d'effectuer la recherche des 3-cutsets. Si ce n'est pas fait, il se peut qu'il existe une 2-cutset contenant au moins une arête non obligatoire. Cela peut avoir pour effet que lors de la recherche des 3-cutsets une arête soit désignée comme supprimable alors qu'elle est nécessaire à l'existence d'un cycle hamiltonien.

Aussi, la suppression d'une arête dans une 3-cutset peut créer une 2-cutset : il faut attendre la fin de la recherche de toutes les 3-cutsets pour rendre effectives les suppressions. Pour avoir une suppression instantanée, il faudrait effectuer une recherche des 2-cutsets à chaque suppression d'arêtes lors de la recherche des 3-cutsets.

Puis, lorsque l'on a peu d'arêtes obligatoires dans le graphe, il y a de grandes chances qu'on ne puisse pas les prendre dans le CST. En revanche, lorsque ce nombre grandit, elles deviennent nécessaires à l'existence d'un CST et forment des chemins, donc des 2-cutsets ! Alors, lorsque $|M|$ tend vers $n$, le nombre d'arêtes d'identifications tend vers 1. Du fait que notre complexité dépend fortement de ce nombre, l'impact est fort sur le temps de résolution.

Enfin, le CST a un autre avantage : il est incrémental. En effet, tant qu'aucune arête du CST n'est supprimée, toutes les arêtes hors du CST appartiennent à un cycle composé d'arêtes du CST, il n'est alors pas nécessaire de le reconstruire.

### 4.3 Implémentation

Nous proposons une implémentation du filtrage $k$-cutset dans l'algorithme 1. La fonction principale est FILTRAGE(G). Les fonctions CHERCHER2CUTSET(G) et CHERCHER3CUTSET(G) appellent la fonction CHERCHERCUTPAIRS(G,isthmAction,cutpairAction), elle fait

une recherche des 2-cutsets comme proposé dans [11] avec une complexité en $O(n+m)$. Ses arguments sont respectivement un graphe, une fonction spécifiant ce qui doit être fait lorsqu'un isthme est trouvé et une fonction spécifiant ce qui doit être fait lorsqu'une 2-cutset est trouvée. Lors de l'exécution de FILTRAGE(G), CHERCHER2CUTSET(G) trouve donc toutes les 2-cutsets. S'il n'existe aucun isthme, la fonction MERGECUTPAIRS(set) remplit le tableau *id* afin que chaque arête appartenant à la même 2-cutset ait la même valeur dans *id*. Si deux 2-cutsets ont une arête en commun, alors chacune de leurs arêtes ont la même valeur dans *id*. On peut alors exploiter l'amélioration proposée en 4.2 qui réduit la complexité en $O(|M|)$. Un CST est ensuite calculé en $O(n)$ avec la méthode proposé en 4.1, l'ensemble S représente les arêtes obligatoire de ce dernier. Enfin, pour chaque arête $a^*$ de S on considère celles n'appartenant soit à aucune cutset, soit à une cutset dont la recherche de $id[a^*]$ n'a pas été effectuée. Il faut ensuite trouver les 3-cutsets contenant $a^*$. Pour cela, on considère $G'$ le graphe duquel l'arête $a^*$ est retiré et on cherche les 2-cutsets dans $G'$ avec la fonction CHERCHER3CUTSET(G'). Lors des itérations, si une 3-cutset avec trois arêtes obligatoires est trouvée, alors l'exécution est stoppée (la variable fail est mise à true), sinon, les cutsets avec l'identifiant $id[a^*]$ sont marquées comme traités.

**Algorithm 1:** $k$-CUTSET($G=(X,U)$,M)

```
global fail
FILTRAGE(G, M) :
  fail ← false
  set ← ∅
  ∀e ∈ U : id[e] ← nil
  CHERCHER2CUTSET(G,M,set)
  if fail then return ;
  CST ← CALCULERCST()
  S ← CST.ARETESOBLIGATOIRES()
  MERGECUTPAIRS(S,set,id)
  for each a* ∈ S do
    if id[a*] = nil or visited[id[a*]] = 0 then
      G' ← (X, U − {a*})
      CHERCHER3CUTSET(G', M)
      if fail then return ;
      if id[a*] ≥ 0 then visited[id[a*]] ← 1;

CHERCHER2CUTSET(G,M,SET) :
  define isthmCutpairs(){ fail ← true}
  define cutpairsFound(M, a1, a2){
      if a1 ∉ M then M ← M ∪ {a1};
      if a2 ∉ M then M ← M ∪ {a2};
      set ← set ∪{a1, a2}
    }
  CHERCHERCUTPAIRS(G,isthmCutpairs,cutpairsFound)

CHERCHER3CUTSET(G,M) :
  define isthm3cutset(){ return }
  define 3cutsetFound(M, a1, a2){
      if a1 ∈ M and a2 ∈ M then fail ← true;
      else if a1 ∈ M then U ← U − {a2};
      else if a2 ∈ M then U ← U − {a1};
    }
  CHERCHERCUTPAIRS(G,isthm3cutset,3cutsetFound)

MERGECUTPAIRS(S,set,id) :
  cpt ← 0
  for each s ∈ set do
    if id[s.a1] ≠ nil and id[s.a2] ≠ nil then
      for each s' ∈ S do
        if id[s'] = id[s.a1] then
          id[s'] ← id[s.a2]
    if id[s.a1] = nil and id[s.a2] = nil then
      id[s.a1] ← id[s.a2] ← cpt
      cpt ← cpt + 1
    if id[s.a1] ≠ nil and id[s.a2] = nil then
      id[s.a2] ← id[s.a1]
    if id[s.a1] = nil and id[s.a2] ≠ nil then
      id[s.a1] ← id[s.a2]
```

## 5 Expérimentations

Les algorithmes ont été implémentés en Java 11 dans un solveur de programmation par contraintes développé localement, appelé Talos. Les expérimentations ont été effectuées sur une machine Windows 10 avec un processeur Intel Core i7-7820HQ CPU @ 2.90GHz et 32Go de RAM. Les instances de références sont issues de la TSPLib [8], une librairie de graphes de références pour le TSP. Toutes les instances considérées sont des graphes symétriques.

Nous présentons les résultats sous la forme de tables. Chacune d'entre elles rapporte le temps de résolution en millisecondes. S'il est strictement supérieur à 30 minutes il est spécifié *t.o.* Le nombre de backtracks est noté #bk. Elles comportent une colonne exprimant le ratio du temps de résolution et du nombre de backtracks.

Afin de vérifier l'intérêt de l'ajout de contraintes structurelles à la WCC, nous allons dans un premier temps vérifier que le filtrage $k$-cutset est efficace en utilisant la stratégie statique "maxCost", qui sélectionne les arêtes selon les coûts décroissants.

La table 1 représente les performances de l'ajout du filtrage $k$-cutset à la WCC. Choisir une stratégie statique telle que maxCost permet de bien comparer les performances des filtrages en évitant une perturbation due à la stratégie. Ces résultats montrent qu'utiliser un filtrage structurel est très intéressant. Par exemple, le temps de résolution de kroE100 a été réduit d'un

facteur 39.24 et le nombre de backtracks d'un facteur 131.96. En effet, le nombre de backtracks est en général réduit d'un gros facteur, ce qui permet d'obtenir une bonne réduction du temps de résolution.

Nous considérons maintenant différentes stratégies, notamment LCFirst maxCost introduite par Fages *et al.* [4]. Elle conserve pour la dernière arête branchée une de ses deux extrémités et sélectionne les arêtes parmi le voisinage du nœud par leurs coûts décroissants. Elle est considérée comme la meilleure stratégie actuelle pour résoudre le TSP en PPC.

| | (1) Talos WCC statique maxCost | | (2) Talos WCC + $k$-cutset statique maxCost | | Ratios (1) / (2) | |
|---|---|---|---|---|---|---|
| Instance | time(ms) | #bk | time(ms) | #bk | time | #bk |
| gr96 | 9997 | 14988 | 2438 | 1512 | 4.1 | 9.91 |
| rat99 | 60 | 36 | 73 | 40 | 0.82 | 0.9 |
| kroA100 | 67044 | 96414 | 14829 | 9444 | 4.52 | 10.21 |
| kroB100 | 178435 | 294686 | 12319 | 7178 | 14.48 | 41.05 |
| kroC100 | 4743 | 4246 | 2702 | 1754 | 1.76 | 2.42 |
| kroD100 | 509 | 474 | 496 | 282 | 1.03 | 1.68 |
| kroE100 | 863352 | 1609168 | 22004 | 12194 | 39.24 | 131.96 |
| rd100 | 29 | 12 | 26 | 4 | 1.12 | 3.0 |
| eil101 | 126 | 122 | 124 | 88 | 1.02 | 1.39 |
| lin105 | 9 | 2 | 10 | 0 | 0.9 | Inf |
| pr107 | 338 | 10 | 339 | 10 | 1.0 | 1.0 |
| gr120 | 4493 | 3790 | 1811 | 960 | 2.48 | 3.95 |
| pr124 | 1065 | 544 | 919 | 304 | 1.16 | 1.79 |
| bier127 | 391 | 398 | 295 | 124 | 1.33 | 3.21 |
| ch130 | 13476 | 11172 | 5178 | 2290 | 2.6 | 4.88 |
| pr136 | t.o | 2234970 | 114673 | 35584 | $\geq$ 15.7 | $\geq$ 62.81 |
| gr137 | 21419 | 11814 | 17634 | 6822 | 1.21 | 1.73 |
| pr144 | 1203 | 454 | 1285 | 418 | 0.94 | 1.09 |
| kroA150 | 1025157 | 601084 | 171299 | 59034 | 5.98 | 10.18 |
| kroB150 | t.o | 1087292 | 939634 | 302318 | $\geq$ 1.92 | $\geq$ 3.6 |
| ch150 | 8626 | 5362 | 4188 | 1532 | 2.06 | 3.5 |
| brg180 | 2843 | 1400 | 3238 | 1390 | 0.88 | 1.01 |
| rat195 | 1656448 | 662298 | 915603 | 287322 | 1.81 | 2.31 |
| d198 | 343600 | 166470 | 170325 | 54512 | 2.02 | 3.05 |
| kroA200 | t.o | 913968 | t.o | 402802 | NaN | 2.27 |
| kroB200 | t.o | 880048 | 1415194 | 345516 | $\geq$ 1.27 | $\geq$ 2.55 |
| gr202 | 15187 | 9636 | 6585 | 2316 | 2.31 | 4.16 |

TABLE 1 – Résultats de la WCC avec le filtrage $k$-cutset et stratégie statique maxCost.

De façon surprenante, la table 2 montre que le filtrage $k$-cutset n'est pas intéressant pour la stratégie LCFirst maxCost. D'après nos expériences, la stratégie semble très ad hoc par rapport au propagateur de la contrainte WCC et notamment à la relaxation lagrangienne. Elle semble corriger une partie du manque de contraintes structurelles de la WCC. Nous observons des résultats similaires à la table 1 pour d'autres stratégies dynamiques, telle que LCFirst minCost.

Aussi, nous proposons d'utiliser la stratégie minRepCost proposée dans [4] qui est mieux adaptée à notre modèle. Elle consiste à sélectionner les arêtes par leurs coûts de remplacement [2] croissant. La table 3 permet d'observer une réduction conséquente de l'espace de recherche et une réduction moindre du temps de calcul. Par exemple, gr137 a un facteur de gain du temps de résolution et du nombre de backtracks respectivement de 3.1 et 6.8.

| | (1) Talos WCC LCFirst maxCost | | (2) Talos WCC + $k$-cutset LCFirst maxCost | | Ratios (1) / (2) | |
|---|---|---|---|---|---|---|
| Instance | time(ms) | #bk | time(ms) | #bk | time | #bk |
| kroB100 | 4844 | 4864 | 28446 | 16948 | 0.17 | 0.29 |
| kroC100 | 872 | 730 | 996 | 538 | 0.88 | 1.36 |
| kroD100 | 479 | 412 | 633 | 378 | 0.76 | 1.09 |
| ch130 | 3501 | 2268 | 7568 | 3282 | 0.46 | 0.69 |
| pr136 | 218637 | 149034 | 146158 | 52208 | 1.5 | 2.85 |
| gr137 | 3570 | 1878 | 8546 | 3312 | 0.42 | 0.57 |
| pr144 | 1109 | 304 | 1259 | 358 | 0.88 | 0.85 |
| kroB200 | 240725 | 91708 | 1583689 | 390622 | 0.15 | 0.23 |
| gr202 | 5916 | 3004 | 6217 | 2084 | 0.95 | 1.44 |

TABLE 2 – Résultats de la WCC avec le filtrage $k$-cutset avec la stratégie LCFirst maxCost.

| | (1) Talos WCC LCFirst maxCost | | (2) Talos WCC + $k$-cutset minRepCost | | Ratios (1) / (2) | |
|---|---|---|---|---|---|---|
| Instance | time(ms) | #bk | time(ms) | #bk | time | #bk |
| gr96 | 1040 | 964 | 1093 | 482 | 0.95 | 2.0 |
| rat99 | 63 | 44 | 67 | 20 | 0.94 | 2.2 |
| kroA100 | 3161 | 2836 | 4735 | 1778 | 0.67 | 1.6 |
| kroB100 | 4844 | 4864 | 1248 | 482 | 3.88 | 10.09 |
| kroC100 | 872 | 730 | 320 | 96 | 2.72 | 7.6 |
| kroD100 | 479 | 412 | 161 | 52 | 2.98 | 7.92 |
| kroE100 | 3362 | 3354 | 3671 | 1422 | 0.92 | 2.36 |
| rd100 | 39 | 18 | 30 | 8 | 1.3 | 2.25 |
| eil101 | 90 | 68 | 42 | 24 | 2.14 | 2.83 |
| lin105 | 10 | 2 | 9 | 0 | 1.11 | Inf |
| pr107 | 343 | 10 | 342 | 10 | 1.0 | 1.0 |
| gr120 | 769 | 540 | 590 | 210 | 1.3 | 2.57 |
| pr124 | 981 | 432 | 1073 | 228 | 0.91 | 1.89 |
| bier127 | 317 | 236 | 1670 | 318 | 0.19 | 0.74 |
| ch130 | 3501 | 2268 | 2731 | 820 | 1.28 | 2.77 |
| pr136 | 218637 | 149034 | 105412 | 23926 | 2.07 | 6.23 |
| gr137 | 3570 | 1878 | 1150 | 276 | 3.1 | 6.8 |
| pr144 | 1109 | 304 | 642 | 74 | 1.73 | 4.11 |
| kroA150 | 21560 | 11510 | 15164 | 3234 | 1.42 | 3.56 |
| kroB150 | 394760 | 217934 | 821804 | 176294 | 0.48 | 1.24 |
| ch150 | 4355 | 1958 | 4417 | 1000 | 0.99 | 1.96 |
| brg180 | 816 | 344 | 612 | 226 | 1.33 | 1.52 |
| rat195 | 114889 | 40060 | 207379 | 29938 | 0.55 | 1.34 |
| d198 | 23976 | 9132 | 456885 | 78262 | 0.05 | 0.12 |
| kroA200 | t.o | 722550 | t.o | 248232 | NaN | 2.91 |
| kroB200 | 240725 | 91708 | 236642 | 32338 | 1.02 | 2.84 |
| gr202 | 5916 | 3004 | 2916 | 366 | 2.03 | 8.21 |

TABLE 3 – Résultats de la WCC avec le filtrage $k$-cutset et la stratégie minRepCost.

L'interaction du filtrage $k$-cutset avec la relaxation lagrangienne est peu clair, une étude plus approfondie devra être menée pour mieux la comprendre. Néanmoins nous observons que la contrainte WCC est construite autour de la relaxation lagrangienne. Rajouter un filtrage peut alors perturber la convergence de cette dernière et parfois, la ralentir. Cela explique pourquoi le facteur de gain du nombre de backtracks est toujours bien supérieur à celui du temps de résolution.

A titre indicatif, la table 4 compare notre méthode avec [4].

| | (1) Choco WCC LCFirst maxCost | | (2) Talos WCC + $k$-cutset minRepCost | | Ratios (1) / (2) | |
|---|---|---|---|---|---|---|
| Instance | time(ms) | #bk | time(ms) | #bk | time | #bk |
| kroB100 | 11576 | 4427 | 1248 | 482 | 9.28 | 9.18 |
| kroC100 | 2735 | 1005 | 320 | 96 | 8.55 | 10.47 |
| kroD100 | 696 | 231 | 161 | 52 | 4.32 | 4.44 |
| ch130 | 5547 | 1341 | 2731 | 820 | 2.03 | 1.64 |
| pr136 | 554465 | 138457 | 105412 | 23926 | 5.26 | 5.79 |
| gr137 | 10616 | 1907 | 1150 | 276 | 9.23 | 6.91 |
| pr144 | 2262 | 307 | 642 | 74 | 3.52 | 4.15 |
| kroA150 | 59529 | 11533 | 15164 | 3234 | 3.93 | 3.57 |
| kroB200 | 867058 | 95941 | 236642 | 32338 | 3.66 | 2.97 |
| gr202 | 15125 | 2667 | 2916 | 366 | 5.19 | 7.29 |

TABLE 4 – Comparatif de l'état de l'art et de Talos avec l'ajout du filtrage $k$-cutset à la WCC et la stratégie minRepCost.

## 6 Conclusion

Nous avons introduit une nouvelle contrainte structurelle dans la WCC basée sur la recherche de $k$-cutsets dans le graphe. Les résultats expérimentaux montrent l'intérêt de notre approche en pratique. Nous avons observé que le nombre de backtracks est réduit d'un ordre de magnitude en fonction de la stratégie choisie, ce qui nous permet d'obtenir un facteur de gain intéressant du temps de résolution. Les interactions entre cette contrainte et la stratégie de recherche, ainsi qu'entre cette contrainte et le modèle lagrangien de la WCC méritent une étude plus poussée.

## Références

[1] David L APPLEGATE, Robert E BIXBY, Vasek CHVATAL et William J COOK : *The traveling salesman problem : a computational study.* Princeton university press, 2006.

[2] Pascal BENCHIMOL, Jean-Charles RÉGIN, Louis-Martin ROUSSEAU, Michel RUEHER et Willem-Jan van HOEVE : Improving the held and karp approach with constraint programming. *In* Andrea LODI, Michela MILANO et Paolo TOTH, éditeurs : *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 40–44, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[3] Sylvain DUCOMMAN, Hadrien CAMBAZARD et Bernard PENZ : Alternative filtering for the weighted circuit constraint : Comparing lower bounds for the tsp and solving tsptw. *In AAAI*, 2016.

[4] Jean-Guillaume FAGES, Xavier LORCA et Louis-Martin ROUSSEAU : The salesman and the tree : the importance of search in cp. *Constraints*, 21(2): 145–162, 2016.

[5] Michael HAYTHORPE : Fhcp challenge set : The first set of structurally difficult instances of the hamiltonian cycle problem, 2019.

[6] Michael HELD et Richard M. KARP : The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6):1138–1162, 1970.

[7] Michael HELD et Richard M. KARP : The traveling-salesman problem and minimum spanning trees : Part ii. *Mathematical Programming*, 1(1):6–25, 1971.

[8] Gerhard REINELT : Tsplib—a traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991.

[9] Robert E. TARJAN : A note on finding the bridges of a graph. *Inf. Process. Lett.*, 2:160–161, 1974.

[10] Robert E. TARJAN : *Data Structures and Network Algorithms.* CBMS-NSF Regional Conference Series in Applied Mathematics, 1983.

[11] Yung H. TSIN : Yet another optimal algorithm for 3-edge-connectivity. *Journal of Discrete Algorithms*, 7(1):130 – 146, 2009. Selected papers from the 1st International Workshop on Similarity Search and Applications (SISAP).

[12] Li-Pu YEH, Biing-Feng WANG et Hsin-Hao SU : Efficient algorithms for the problems of enumerating cuts by non-decreasing weights. *Algorithmica*, 56(3):297–312, 2010.

# Contrainte globale *abstractXOR* : résultats de complexité et algorithmes de propagation

**Loïc Rouquette*** **Christine Solnon**
Université de Lyon, INSA-Lyon, CNRS, LIRIS, UMR5205, F-69621, France
{loic.rouquette, christine.solnon}@insa-lyon.com

**Résumé**

Les algorithmes de chiffrement symétrique par bloc utilisent une clé secrète pour chiffrer un texte. Le but de l'attaque *related-key* est d'évaluer s'il est possible de retrouver la clé en étudiant la propagation des différences entre des textes et des clés donnés. Pour cela, les cryptanalystes doivent calculer des chemins différentiels maximaux. Ce problème est résolu en deux étapes : dans une première étape, chaque octet est abstrait par un booléen indiquant si l'octet contient une différence ou non et on recherche des solutions abstraites ; dans une seconde étape, on recherche une solution concrète optimale pour chaque solution abstraite. Certaines solutions abstraites peuvent ne pas admettre de solution concrète, et un enjeu important est de limiter au maximum le nombre de solutions abstraites incohérentes au niveau concret. La perte de précision au niveau abstrait vient essentiellement du fait que le chiffrement utilise massivement des XOR (ou exclusif bit à bit), et que cette opération est très mal abstraite lors de la première étape. Pour améliorer cela, nous proposons d'introduire une contrainte permettant de propager un ensemble de XOR de façon globale.

Nous étudions tout d'abord la complexité du problème consistant à décider si un ensemble de XOR admet une solution concrète. Nous montrons que si le problème est polynomial lorsque les variables ne sont pas contraintes, il devient NP-complet si on contraint certaines variables à prendre une valeur comprise entre 1 et une borne supérieure donnée. Nous introduisons ensuite la contrainte globale $abstractXOR$ qui est satisfaite s'il existe une solution abstraite pouvant être concrétisée dans le cas où les variables ne sont pas contraintes par une borne supérieure, et nous introduisons des algorithmes polynomiaux pour propager cette contrainte. Nous montrons comment utiliser cette contrainte globale pour modéliser un problème de recherche de chemin différentiel maximal pour Midori, et nous comparons les performances du modèle utilisant cette contrainte globale avec d'autres modèles.

*Papier doctorant : Loïc Rouquette est auteur principal.

**Abstract**

Block cipher algorithms use shared secret keys to cipher texts. The aim of the *related-key* attack is to evaluate if it is possible to find out the key by studying difference propagations. To this aim, cryptanalysts need to compute maximum differential paths. This problem is solved in two steps : in a first step, each byte is abstracted by a boolean indicating if the byte contains a difference or not and we search for abstract solutions , in a second step, we search for an optimal concrete solution for each abstract solution. Some abstract solutions may not accept concrete solutions and our goal is to limit as much as possible the number of inconsistent abstract solutions at the concrete level. These inconsistencies mainly come from the fact that ciphering massively relies on XOR operations (bitwise exclusive or), and this operation is poorly abstracted during the first step. To improve that, we introduce a constraint for propagating a set of XORs in a global way.

First we study the complexity of the problem aiming at deciding if a XOR set accepts a concrete solution. We show that if the problem is polynomial when variables are not constrained, it becomes NP-complete if we constrain some variables to take values between 1 and a given upper bound. Then, we introduce the global constraint $abstractXOR$ which is satisfied if there exists an abstract solution that can be concretized in the case where variables are not constrained by an upper bound and we introduce polynomial algorithms to propagate this constraint. We show how to use this global constraint to model a maximal differential path problem for Midori and we compare the performance of the model using this global constraint with other models.

## 1 Motivations

Les algorithmes de chiffrement symétrique par bloc (tels que AES [8] ou Midori [1]) garantissent la confidentialité des communications en utilisant une clé secrète $K$ pour chiffrer un texte initial $X$ en un texte chiffré $f_K(X)$, de telle sorte que le texte chiffré puisse être déchiffré en utilisant la même clé.

Le but de la cryptanalyse est de vérifier que la confidentialité est bien garantie. En particulier, la *cryptanalyse différentielle* évalue s'il est possible de retrouver la clé en moins de $2^{|K|}$ essais en étudiant la propagation des différences entre deux textes $X$ et $X'$ pendant le chiffrement [4]. La différence $\delta X$ entre les deux textes $X$ et $X'$ est obtenue en appliquant l'opérateur XOR (ou exclusif entre les bits de $X$ et $X'$), noté $\delta X = X \oplus X'$.

Les algorithmes de chiffrement ont été prouvés robustes pour ces attaques différentielles. Ainsi, l'attaque *related-key* introduite dans [3] autorise l'attaquant à injecter des différences non seulement entre deux textes $X$ et $X'$ mais aussi entre deux clés $K$ et $K'$ (même si la clé secrète $K$ reste inconnue pour l'attaquant). Afin de concevoir cette attaque, les cryptanalystes doivent trouver des *chemins différentiels maximaux*, *i.e.*, des différences en entrée ($\delta X = X \oplus X'$ et $\delta K = K \oplus K'$) et en sortie ($\delta X_{out} = f_K(X) \oplus f_{K'}(X')$) maximisant la probabilité d'observer $\delta X_{out}$ étant données $\delta X$ et $\delta K$.

**Résolution en deux étapes.** Pour réduire la taille de l'espace de recherche à explorer, Knudsen a introduit la notion de *différentielle tronquée* [13]. L'idée est de regrouper les bits (de $\delta X$, $\delta K$ et $\delta X_{out}$, mais aussi de tous les états intermédiaires du processus de chiffrement) par séquences de $b$ bits consécutifs : typiquement, $b = 8$ et chaque séquence $\delta_i$ de $b$ bits correspond à un octet. Chacune de ces séquences $\delta_i$ est abstraite par un booléen $\Delta_i$ indiquant si la séquence contient une différence ou non, *i.e.*,
$\Delta_i = faux \Leftrightarrow \delta_i = 0$ et $\Delta_i = vrai \Leftrightarrow \delta_i \in [1, 2^b - 1]$.

Le problème est alors résolu en deux étapes : dans une première étape, on recherche un ensemble de *solutions abstraites* où les variables booléennes satisfont une abstraction des opérations de chiffrement ; dans une seconde étape, pour chaque solution abstraite, on recherche une *solution concrète* où chaque variable concrète $\delta_i$ est affectée à 0 (resp. une valeur comprise entre 1 et $2^b - 1$) si $\Delta_i = faux$ (resp. $\Delta_i = vrai$) dans la solution abstraite.

L'ensemble des solutions abstraites calculé lors de la première étape doit permettre de trouver toutes les solutions concrètes, *i.e.*, pour chaque solution concrète il doit exister une solution abstraite telle que, pour chaque variable concrète $\delta_i$ affectée à 0 (resp. une valeur entre 1 et $2^b - 1$), la variable abstraite $\Delta_i$ soit affectée à *faux* (resp. *vrai*). En revanche, certaines solutions abstraites calculées lors de la première étape peuvent ne pas correspondre à une solution concrète. Ces solutions abstraites sont dites *b-incohérentes* car il n'existe pas de solution concrète dans le cas où le nombre de bits utilisés pour représenter les valeurs des variables concrètes $\delta_i$ est égal à $b$. Pour que le processus de résolution soit efficace, il est fondamental de limiter au maximum le nombre de solutions abstraites *b-incohérentes* calculées lors de la première étape.

**Abstraction du XOR lors de la première étape.** L'opération principale des algorithmes de chiffrement symétrique par bloc tels qu'AES ou Midori est le XOR. Lors de la première étape, le XOR concret $\oplus$ est abstrait par l'opérateur $\oplus_{\mathcal{A}}$ dont la table de vérité est :

| $\Delta_1$ | $\Delta_2$ | $\Delta_1 \oplus_{\mathcal{A}} \Delta_2$ |
|---|---|---|
| *faux* | *faux* | *faux* |
| *faux* | *vrai* | *vrai* |
| *vrai* | *faux* | *vrai* |
| *vrai* | *vrai* | *vrai* ou *faux* |

Cet opérateur abstrait implique une perte en précision car lorsque $\Delta_1$ et $\Delta_2$ sont affectées à *vrai*, on ne sait pas si $\Delta_1 \oplus_{\mathcal{A}} \Delta_2$ est égal à *vrai* ou *faux*. Cela provient du fait que le résultat d'un XOR entre deux variables $\delta_1$ et $\delta_2$ différentes de zéro peut être soit zéro (si $\delta_1 = \delta_2$) soit différent de zéro (si $\delta_1 \neq \delta_2$). Considérons par exemple les problèmes concret (à gauche) et abstrait (à droite) :

$$\begin{array}{ll} \delta_1 \oplus \delta_2 = \delta_3 & \Delta_1 \oplus_{\mathcal{A}} \Delta_2 = \Delta_3 \\ \delta_2 \oplus \delta_4 = \delta_5 & \Delta_2 \oplus_{\mathcal{A}} \Delta_4 = \Delta_5 \\ \delta_3 \oplus \delta_5 = \delta_6 & \Delta_3 \oplus_{\mathcal{A}} \Delta_5 = \Delta_6 \end{array}$$

L'affectation $\Delta_1 = \Delta_2 = \Delta_3 = \Delta_5 = vrai$ et $\Delta_4 = \Delta_6 = faux$ est solution du problème abstrait, mais elle est $b$-incohérente car si $\delta_4 = \delta_6 = 0$, alors nous pouvons en déduire que $\delta_2 = \delta_3 = \delta_5$ et donc que $\delta_1 = 0$. Par conséquent, il n'existe pas de solution concrète où $\delta_4 = \delta_6 = 0$ et où chaque autre variable prend une valeur dans $[1, 2^b - 1]$, quel que soit $b$.

**Contributions et organisation de l'article.** Dans cet article, nous introduisons une contrainte globale permettant de calculer efficacement une meilleure approximation (contenant moins de solutions $b$-incohérentes) d'un ensemble de contraintes XOR au niveau abstrait, pendant la première étape.

Dans la section 2, nous définissons le problème $XOR_\infty$ visant à décider s'il existe une solution à un ensemble de contraintes XOR dans le cas où les variables $\delta_i$ peuvent prendre n'importe quelle valeur entière (non bornée par $2^b - 1$). Nous introduisons un algorithme polynomial permettant de résoudre ce problème.

Dans la section 3, nous étudions la complexité de deux cas particuliers de $XOR_\infty$ :

— cas où certaines variables sont contraintes à prendre pour valeur 0 et les autres à prendre une valeur différente de 0, dénoté $XOR_{\infty,\neq 0}$ ;

— cas où certaines variables sont contraintes à prendre pour valeur 0 et les autres à prendre une valeur dans $[1, 2^b - 1]$, dénoté $XOR_{b,\neq 0}$.

Nous montrons que $XOR_{\infty,\neq 0}$ est polynomial, et $XOR_{b,\neq 0}$ est NP-complet. A notre connaissance, ce résultat est nouveau. Dans [7], les auteurs s'intéressent au problème *reachability*, consistant à décider s'il est possible de déduire un terme secret $s$ à partir d'un ensemble de message $T$ pour un protocole de cryptographie donné, et montrent que ce problème est décidable dans le cas où $T$ ne contient que des contraintes XOR.

Dans la section 4, nous introduisons la contrainte globale *abstractXOR* permettant de propager le fait qu'il doit exister une affectation d'un ensemble de variables booléennes qui soit $\infty$-cohérente pour un ensemble donné d'équations XOR, et nous introduisons des algorithmes polynomiaux pour propager cette contrainte.

Dans la section 5, nous montrons comment utiliser cette contrainte pour modéliser très simplement un problème de cryptanalyse différentielle pour Midori, et nous comparons les performances du modèle utilisant cette contrainte avec le modèle introduit dans [9].

## 2 Définition du problème $XOR_\infty$

Le problème $XOR_\infty$ vise à décider s'il existe une solution à un ensemble de contraintes XOR dans le cas où les variables peuvent prendre n'importe quelle valeur entière. On impose en plus qu'au moins une variable soit différente de 0 afin d'interdire la solution triviale où toutes les variables sont égales à 0.

**Définition 1** *Une instance de $XOR_\infty$ est définie par un couple $(X_\delta, C)$ tel que $X_\delta = \{\delta_1, \delta_2, \ldots, \delta_m\}$ est un ensemble de $m$ variables entières, et $C$ est un ensemble de $n$ équations, chaque équation étant de la forme $\delta_{i_1} \oplus \delta_{i_2} \oplus \ldots \oplus \delta_{i_x} = 0$. Le problème consiste à décider s'il existe une solution non triviale,* i.e., *une affectation d'une valeur entière à chaque variable $\delta_i \in X_\delta$ telle qu'au moins une variable soit différente de 0 et que chaque équation de $C$ soit satisfaite.*

Une instance $(X_\delta, C)$ est représentée par une matrice $M$ comportant $n$ lignes et $m$ colonnes telle que $M[i,j] = 1$ si $\delta_j$ apparaît dans la $i^{\text{ème}}$ équation, et $M[i,j] = 0$ sinon. Un exemple d'une telle représentation matricielle est donné dans la figure 1. Nous considérons dans la suite que chaque ligne et chaque colonne de la matrice comporte au moins un 1 (*i.e.*, chaque variable apparaît dans au moins une contrainte, et chaque contrainte porte sur au moins une variable).

| | $\delta_1$ | $\delta_2$ | $\delta_3$ | $\delta_4$ | $\delta_5$ | $\delta_6$ | $\delta_7$ |
|---|---|---|---|---|---|---|---|
| $\delta_3 \oplus \delta_6 \oplus \delta_4 = 0$ | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| $\delta_5 \oplus \delta_7 \oplus \delta_6 = 0$ | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| $\delta_2 \oplus \delta_5 \oplus \delta_3 = 0$ | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| $\delta_2 \oplus \delta_7 \oplus \delta_1 = 0$ | 1 | 1 | 0 | 0 | 0 | 0 | 1 |

FIGURE 1 – Exemple d'instance de $XOR_\infty$.

**Algorithme 1 :** normalisation($M$)

1 $r \leftarrow 1$ ; $j \leftarrow 1$
2 **tant que** $r \leq n$ **faire**
3  **si** *il existe une ligne* $i \in [r, n]$ *tq* $M[i,j] = 1$ **alors**
4  **pour** *chaque ligne* $i' \in [1, n]$ *tq* $i' \neq i$ *et* $M[i', j] = 1$ **faire**
5  **pour** *chaque colonne* $j' \in [r, m]$ **faire**
6  $M[i', j'] \leftarrow M[i', j'] \oplus M[i, j']$
7  **si** *la ligne* $i'$ *ne comporte que des 0* **alors**
8  Supprimer la ligne $i$ et décrémenter $n$
9  Echanger les lignes $i$ et $r$ et incrémenter $r$
10  Incrémenter $j$

**Forme échelonnée réduite d'une matrice.** Le premier élément non nul de chaque ligne de la matrice est appelé *pivot*, et on note $c_i$ le numéro de colonne du pivot de la ligne $i$. Une matrice est sous forme *échelonnée* si pour chaque couple de lignes $(i, i')$ tel que $i < i'$, et pour chaque colonne $j \leq c_i$, $M[i', j] = 0$. Les variables associées aux colonnes contenant des pivots sont appelées *variables de base* ; les autres variables sont appelées *variables hors-base*. Une matrice échelonnée est *réduite* si, pour chaque ligne $i$, la colonne $c_i$ contient un seul 1.

L'algorithme 1 s'inspire de l'algorithme d'élimination de Gauss pour transformer une matrice quelconque en une matrice échelonnée réduite admettant le même ensemble de solutions. A chaque itération de la boucle lignes 2-9, les $r - 1$ premières lignes de la matrice $M$ sont sous forme échelonnée réduite, et si la colonne courante $j$ contient un élément $M[i,j]$ pouvant devenir le pivot de la ligne $i$, alors la ligne $i$ est XORée avec chaque ligne $i' \neq i$ comportant un 1 en colonne $j$, puis les lignes $i$ et $r$ sont échangées et $r$ est incrémenté. Si certaines équations du système initial sont redondantes (*i.e.*, elles peuvent être obtenues en combinant d'autres équations de telle sorte qu'elles s'annulent lorsqu'elles sont XORées avec ces équations), alors elles sont supprimées (lignes 7-8).

Cet algorithme ne change pas l'ensemble des solutions car les seules opérations effectuées sur la matrice

(en dehors des échanges de lignes) sont des XOR entre lignes (lignes 5-6), et l'équation $\delta_{i_1} \oplus \ldots \oplus \delta_{i_x} = 0$ admet les mêmes solutions que l'équation $\delta_{i_1} \oplus \ldots \oplus \delta_{i_x} \oplus \delta_{j_1} \oplus \ldots \oplus \delta_{j_y} = 0$ dès lors que $\delta_{j_1} \oplus \ldots \oplus \delta_{j_y} = 0$. L'algorithme se termine lorsque $r = n + 1$, et les $n$ lignes de la matrice sont sous forme échelonnée réduite.

La complexité de cet algorithme est $\mathcal{O}(mn^2)$.

L'exécution de cet algorithme sur la matrice de la figure 1 permet d'obtenir la matrice suivante :

| $\delta_1$ | $\delta_2$ | $\delta_3$ | $\delta_4$ | $\delta_5$ | $\delta_6$ | $\delta_7$ |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 |

Les variables de base sont $\delta_1$, $\delta_2$, $\delta_3$ et $\delta_5$, et les pivots sont coloriés en gris.

**Solutions d'une forme échelonnée réduite.** Si la matrice sous forme échelonnée réduite comporte autant de lignes que de colonnes, alors il n'existe pas de solution non triviale car chaque équation contraint une variable de base à prendre pour valeur 0. Sinon, il existe autant de solutions que d'affectations différentes pour les variables hors-base : pour chaque affectation, on peut construire une solution en affectant à chaque variable $\delta_i$ de la base le résultat du XOR des valeurs des variables hors base apparaissant dans la même ligne que $\delta_i$.

Sur notre exemple, étant donnée une affectation des variables $\delta_4$, $\delta_6$ et $\delta_7$, on déduit les valeurs de $\delta_1$, $\delta_2$, $\delta_3$ et $\delta_5$ en exécutant les opérations suivantes :

$$\delta_1 = \delta_4 \qquad \delta_2 = \delta_4 \oplus \delta_7$$
$$\delta_3 = \delta_4 \oplus \delta_6 \qquad \delta_5 = \delta_6 \oplus \delta_7$$

Par conséquent, pour décider si une instance du problème $XOR_\infty$ admet une solution non triviale, il suffit de vérifier que la matrice en sortie de l'algorithme 1 a plus de colonnes que de lignes.

## 3 Complexité de variantes de $XOR_\infty$

Rappelons que lors de la première étape de résolution d'un problème de cryptanalyse différentielle, on ne recherche pas une solution à une instance $(X_\delta, C)$ de $XOR_\infty$, mais on résout un problème abstrait où chaque variable entière $\delta_i$ est abstraite par une variable booléenne $\Delta_i$ et où l'opérateur $\oplus$ entre entiers est abstrait par l'opérateur $\oplus_{\mathcal{A}}$ entre booléens. Dans ce contexte, pour vérifier la correction d'une solution abstraite, nous devons décider si une instance du problème $XOR_\infty$ possède une solution dans le cas où certaines variables sont contraintes à prendre pour valeur 0 et les autres à prendre une valeur dans $[1, 2^b - 1]$. Dans la section 3.2, nous montrons que ce problème, dénoté $XOR_{b,\neq 0}$, est NP-complet. Mais auparavant nous nous intéressons au cas intermédiaire entre $XOR_\infty$ et $XOR_{b,\neq 0}$, et nous montrons que le problème est polynomial quand certaines variables sont contraintes à prendre pour valeur 0 et les autres à prendre une valeur différente de 0 (problème dénoté $XOR_{\infty,\neq 0}$).

### 3.1 Complexité de $XOR_{\infty,\neq 0}$

$XOR_{\infty,\neq 0}$ est la variante de $XOR_\infty$ où certaines variables sont contraintes à prendre des valeurs non nulles, et d'autres la valeur 0.

**Définition 2** *Une instance de $XOR_{\infty,\neq 0}$ est définie par un triplet $(X_\delta, C, X_{\neq 0})$ où $(X_\delta, C)$ est une instance de $XOR_\infty$, et $X_{\neq 0} \subseteq X_\delta$ est un sous-ensemble de variables. Le problème consiste à décider s'il existe une solution de $(X_\delta, C)$ telle que chaque variable $\delta_i \in X_{\neq 0}$ est affectée à une valeur différente de zéro et chaque variable $\delta_i \in X_\delta \setminus X_{\neq 0}$ est affectée à zéro.*

Pour décider si une instance $(X_\delta, C, X_{\neq 0})$ de $XOR_{\infty,\neq 0}$ possède une solution, nous allons utiliser la propriété suivante :

**Propriété 1** *Toute matrice sous forme échelonnée réduite comportant au moins deux 1 par ligne admet au moins une solution où toutes les variables ont une valeur strictement positive.*

**Preuve.** L'algorithme 2 montre comment construire une telle solution : il affecte une à une les variables hors-base, en choisissant à chaque fois la première valeur positive possible. Le point clé consiste à maintenir, pour chaque variable de base, le résultat du XOR sur les valeurs des variables hors-base dont elle dépend et qui sont déjà affectées : ces valeurs sont initialisées à 0 (ligne 2) ; à chaque fois qu'une variable hors-base $\delta_j$ est affectée à une valeur $v_j$, pour chaque variable de base $\delta_{c_i}$ dépendant de $\delta_j$, on met à jour $v_{c_i}$ en le XORant avec $v_j$ (lignes 6-7). Au moment de choisir une valeur pour une variable hors-base $\delta_j$, on cherche l'ensemble $S$ des valeurs interdites pour $\delta_j$ (ligne 4) : une valeur $v$ est interdite si elle contraint une variable de base $\delta_{c_i}$ à prendre pour valeur 0, *i.e.*, $M[i, j]$ est le dernier 1 de la ligne $i$ et le résultat du XOR sur les variables hors-base déjà affectées dont dépend $\delta_{c_i}$ (*i.e.*, $v_{c_i}$) est égal à $v$. □

Considérons par exemple la matrice échelonnée réduite de notre exemple, pour laquelle les numéros de colonne des pivots sont $P = \{1, 2, 3, 5\}$.

— Pour choisir $v_4$, on construit $S = \{v_1\} = \{0\}$, et on affecte $v_4$ à la plus petite valeur de $\mathbb{N}^+ \setminus \{0\}$, *i.e.*, $v_4 = 1$. On a $v_1 = v_2 = v_3 = 1 \oplus 0 = 1$.

**Algorithme 2 :** Concrétisation($M$)

**Input :** Une matrice $M$ sous forme échelonnée réduite telle que, pour chaque ligne $i \in [1,n]$,$\sum_{j=1}^{m} M[i,j] > 1$

**Output :** Une valeur $v_j > 0$ pour chaque colonne $j \in [1,m]$ telle que $\delta_1 = v_1, \ldots \delta_m = v_m$ soit une solution de $M$

1 Soit $P = \{c_i : i \in [1,n]\}$ l'ensemble des numéros de colonne des pivots de $M$
2 **pour** *chaque colonne* $j \in P$ **faire** initialiser $v_j$ à 0 ;
3 **pour** *chaque colonne* $j \in [1,m] \setminus P$ **faire**
4 $\quad S \leftarrow \{v_{c_i} : i \in [1,n], M[i,j]{=}1 \wedge \sum_{j'=j+1}^{m} M[i,j']{=}0\}$
5 $\quad$ Affecter à $v_j$ la plus petite valeur de $\mathbb{N}^+ \setminus S$
6 $\quad$ **pour** *chaque ligne* $i \in [1,n]$ *tq* $M[i,j] = 1$ **faire**
7 $\quad\quad v_{c_i} \leftarrow v_{c_i} \oplus v_j$

— Pour choisir $v_6$, on construit $S = \{v_3\} = \{1\}$ et on affecte $v_6$ à la plus petite valeur de $\mathbb{N}^+ \setminus \{1\}$, *i.e.*, $v_6 = 2$. On a $v_3 = 1 \oplus 2 = 3$ et $v_5 = 0 \oplus 2 = 2$.
— Pour choisir $v_7$, on construit $S = \{v_2, v_5\} = \{1,2\}$ et on affecte $v_7$ à la plus petite valeur de $\mathbb{N}^+ \setminus \{1,2\}$, *i.e.*, $v_7 = 3$. On a $v_2 = 1 \oplus 3 = 2$ et $v_5 = 2 \oplus 3 = 1$.

L'affectation finale est $v_1 = 1$, $v_2 = 2$, $v_3 = 3$, $v_4 = 1$, $v_5 = 1$, $v_6 = 2$, et $v_7 = 3$.

La propriété 1 nous permet de montrer la propriété suivante :

**Propriété 2** *Le problème $XOR_{\infty,\neq 0}$ est polynomial.*

**Preuve.** Pour décider si une instance $(X_\delta, C, X_{\neq 0})$ de $XOR_{\infty,\neq 0}$ admet une solution, nous commençons par supprimer de chaque équation de $C$ les variables n'appartenant pas à $X_{\neq 0}$, et nous supprimons les équations ne comportant que des variables n'appartenant pas à $X_{\neq 0}$. Cette transformation ne change pas l'ensemble de solutions de l'instance car $\forall X \in \mathbb{N}, X \oplus 0 = X$. Nous construisons ensuite la matrice $M$ associée aux équations résultantes (telle que chaque colonne de $M$ correspond à une variable de $X_{\neq 0}$ devant prendre une valeur différente de 0) et nous la mettons sous forme échelonnée réduite. Si la matrice échelonnée réduite possède une ligne comportant exactement un 1, cela implique qu'il existe une équation de la forme $\delta_i = 0$, et cette équation ne peut pas être satisfaite. Dans ce cas, l'instance n'a pas de solution. Sinon, l'instance admet au moins une solution du fait de la propriété 1.

Étant donné que la matrice peut être mise sous forme échelonnée réduite en temps polynomial en utilisant l'algorithme 1, le problème $XOR_{\infty,\neq 0}$ est polynomial. □

### 3.2 Complexité de $XOR_{b,\neq 0}$

$XOR_{b,\neq 0}$ est la variante de $XOR_\infty$ où certaines variables sont contraintes à prendre des valeurs comprises entre 1 et $2^b - 1$, et d'autres la valeur 0.

**Définition 3** *Une instance de $XOR_{b,\neq 0}$ est définie par un quadruplet $(X_\delta, C, b, X_{\neq 0})$ où $(X_\delta, C)$ est une instance de $XOR_\infty$, $b \geq 1$ est un entier positif, et $X_{\neq 0} \subseteq X_\delta$ est un sous-ensemble de variables. Le problème consiste à décider s'il existe une solution de $(X_\delta, C)$ telle que chaque variable $\delta_i \in X_{\neq 0}$ a une valeur comprise entre 1 et $2^b - 1$ et chaque variable $\delta_i \in X_\delta \setminus X_{\neq 0}$ est affectée à zéro.*

La propriété 1 n'est plus valide dans le cas où les variables sont bornées, et une matrice sous forme échelonnée réduite comportant au moins deux 1 par ligne peut ne pas admettre de solution. Considérons par exemple l'ensemble d'équations suivant :

$$\delta_1 \oplus \delta_2 \oplus \delta_5 = 0 \qquad \delta_1 \oplus \delta_3 \oplus \delta_6 = 0$$
$$\delta_1 \oplus \delta_4 \oplus \delta_7 = 0 \qquad \delta_2 \oplus \delta_3 \oplus \delta_8 = 0$$
$$\delta_2 \oplus \delta_4 \oplus \delta_9 = 0 \qquad \delta_3 \oplus \delta_4 \oplus \delta_{10} = 0$$

La matrice $M$ associée à cet ensemble d'équations est :

| $\delta_5$ | $\delta_6$ | $\delta_7$ | $\delta_8$ | $\delta_9$ | $\delta_{10}$ | $\delta_1$ | $\delta_2$ | $\delta_3$ | $\delta_4$ |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |

$M$ est sous forme échelonnée réduite et respecte bien la condition d'avoir au moins deux 1 par ligne. Cependant, si $X_{\neq 0}$ contient toutes les variables, alors l'instance n'a pas de solution car chaque équation $\delta_i \oplus \delta_j \oplus \delta_k = 0$ implique que $\delta_i \neq \delta_j$ (si $\delta_i = \delta_j$ alors $\delta_k = \delta_i \oplus \delta_j = 0$). Ainsi, les variables $\delta_1$, $\delta_2$, $\delta_3$ et $\delta_4$ doivent prendre des valeurs toutes différentes. Si $b = 2$, alors il n'existe pas de solution où chaque variable est affectée à une valeur comprise entre 1 et $2^2 - 1 = 3$ car il n'y a pas suffisamment de valeurs dans $[1,3]$ pour affecter des valeurs toutes différentes à $\delta_1$, $\delta_2$, $\delta_3$ et $\delta_4$.

De fait, la propriété 3 nous montre que $XOR_{b,\neq 0}$ ne peut être résolu en temps polynomial si P≠NP.

**Propriété 3** *Le problème $XOR_{b,\neq 0}$ est NP-complet.*

**Preuve.** Le problème appartient à la classe NP car nous pouvons vérifier en temps polynomial qu'une affectation donnée des variables de $X_\delta$ est solution. Montrons qu'il est NP-complet en réduisant le problème de coloriage d'un graphe en $XOR_{b,\neq 0}$. Étant donnés un

graphe $G = (V, E)$ et un entier $x$, le problème du coloriage consiste à décider s'il est possible d'affecter une valeur entière comprise entre 1 et $x$ à chaque sommet de $V$ de telle sorte que pour chaque arête $(i, j) \in E$, la valeur affectée à $i$ soit différente de la valeur affectée à $j$. Ce problème est NP-complet [12], et il reste NP-complet même si $x = 2^b - 1$ dès lors que $b \geq 2$ (lorsque $b = 2$, on obtient le problème de coloriage avec 3 couleurs). Pour réduire une instance $(G = (V, E), x)$ d'un problème de coloriage (où $x = 2^b - 1$) en une instance $(X_\delta, C, b, X_{\neq 0})$ de $XOR_{b,\neq 0}$, on définit :

— $X_\delta = \{\delta_i : i \in V\} \cup \{\delta_{ij} : (i, j) \in E\}$
— $C = \{\delta_i \oplus \delta_j \oplus \delta_{ij} = 0 : (i, j) \in E\}$
— $X_{\neq 0} = X_\delta$

Chaque équation $\delta_i \oplus \delta_j \oplus \delta_{ij} = 0$ impose que les variables associées aux sommets $i$ et $j$ (*i.e.*, $\delta_i$ et $\delta_j$) prennent des valeurs différentes, de sorte que toute solution de l'instance de $XOR_{b,\neq 0}$ est un coloriage valide de $G$. De même, tout coloriage valide de $G$ correspond à une solution de l'instance de $XOR_{b,\neq 0}$ : il suffit d'affecter à chaque variable $\delta_i$ associée à un sommet $i$ la valeur affectée à $i$ dans le coloriage et, pour chaque arête $(i, j) \in E$, d'affecter $\delta_{ij}$ à $\delta_i \oplus \delta_j$. □

## 4 Définition de *abstractXOR*

Pour limiter au maximum le nombre de solutions abstraites $b$-incohérentes lors de la première étape du calcul d'un chemin différentiel maximal, il faudrait assurer que, pour chaque solution abstraite, il existe une solution concrète satisfaisant l'ensemble des XOR et telle que, pour chaque variable booléenne $\Delta_i$ affectée à vrai dans la solution abstraite, la variable concrète $\delta_i$ ait une valeur comprise entre 1 et $2^b - 1$. Cela reviendrait à décider si une instance de $XOR_{b,\neq 0}$ admet une solution, et comme ce problème est NP-complet, nous proposons de définir une contrainte globale assurant la relaxation $XOR_{\infty,\neq 0}$ de ce problème : en relâchant la contrainte sur la borne supérieure des valeurs concrètes, on peut propager efficacement la contrainte globale.

**Définition 4** *Etant donnés une instance $(X_\delta, C)$ du problème $XOR_\infty$, et un ensemble $X_\Delta$ de variables booléennes tel que $X_\Delta$ contient une variable $\Delta_i$ pour chaque variable $\delta_i \in X_\delta$, la contrainte globale $abstractXOR_{X_\delta,C}(X_\Delta)$ est satisfaite si et seulement s'il existe une solution à l'instance de $XOR_{\infty,\neq 0}$ définie par $(X_\delta, C, X_{\neq 0})$ avec $X_{\neq 0} = \{\delta_i \in X_\delta : \Delta_i = vrai\}$.*

Notons qu'on ne cherche pas à calculer la solution concrète de $(X_\delta, C, X_{\neq 0})$, mais juste à calculer une solution abstraite (sur les booléens) pour laquelle il existe au moins une solution concrète (sur les entiers).

### 4.1 Vérification de la faisabilité d'*abstractXOR*

On note $n$ et $m$ le nombre de lignes et de colonnes de $M$. Etant donné un numéro de colonne $j \in [1, m]$, on note $\delta_j$ la variable entière associée à la colonne $j$, $\Delta_j$ la variable booléenne correspondante, et $D(\Delta_j)$ son domaine. Pour chaque ligne $i$, on définit :

— $var_i = \{j \in [1, m] : M[i, j] = 1\}$ ;
— $vrai_i = \{j \in var_i : D(\Delta_j) = \{vrai\}\}$.

On note $B \subseteq [1, m]$ l'ensemble des colonnes associées à une variable de base et $HB = [1, m] \setminus B$ l'ensemble des colonnes associées à une variable hors-base. Enfin, rappelons que nous notons $c_i$ le numéro de colonne de la variable de base de la ligne $i$.

Pour vérifier la faisabilité d'*abstractXOR* pendant la recherche, nous maintenons la matrice $M$ sous forme échelonnée réduite, nous supprimons de $M$ les colonnes correspondant à des variables affectées à *faux*, et nous vérifions qu'il y a toujours au moins deux 1 par ligne : la propriété 1 assure qu'il existe toujours une solution dans ce cas. Nous appliquons pour cela les règles R1 à R3 à chaque fois que cela est possible, jusqu'à ce que soit un domaine devienne vide (une incohérence est détectée), soit plus aucune règle ne s'applique.

**R1.** Si $\exists j \in HB, D(\Delta_j) = \{faux\}$, alors supprimer la colonne $j$ de $M$.

**R2.** Si $\exists j_1 \in B, D(\Delta_{j_1}) = \{faux\}$, alors :

— Soit $i_1$ le numéro de ligne tel que $M[i_1, j_1] = 1$
— Si $var_{i_1} = \{j_1\}$, alors supprimer la colonne $j_1$ et la ligne $i_1$ de $M$
— Sinon, il faut choisir une nouvelle base :
  — choisir $j_2 \in var_{i_1} \setminus \{j_1\}$
  — Pour chaque ligne $i_2 \neq i_1$ telle que $M[i_2, j_2] = 1$ et pour chaque colonne $j_3 \in [1, m]$, remplacer $M[i_2, j_3]$ par $M[i_2, j_3] \oplus M[i_1, j_3]$
  — Supprimer la colonne $j_1$

**R3.** Si $\exists i \in [1, n], var_i = \{j\} \wedge vrai \in D(\Delta_j)$ alors supprimer *vrai* de $D(\Delta_j)$.

**Propriété 4** *L'application de R1, R2 et R3 ne change pas l'ensemble des solutions de la contrainte globale.*

**Preuve.** R1 et R2 suppriment une colonne associée à une variable $\delta_j$ qui doit être affectée à 0, et nous avons vu dans la section 3.1 que cela ne change pas l'ensemble des solutions de $M$. R2 remplace également l'équation de la ligne $i_2$ (pour chaque ligne $i_2$ où apparaît la variable qui entre dans la base) par le résultat d'un XOR entre l'équation de la ligne $i_2$ et l'équation de la ligne $i_1$, et nous avons vu dans la section 2 que cela ne change pas l'ensemble des solutions de $M$. R3 supprime

*vrai* de $D(\Delta_j)$ dans le cas où la ligne $i$ correspond à l'équation $\delta_j = 0$. Cette équation ne peut pas être satisfaite si $\Delta_j = vrai$, et donc $\Delta_j$ ne peut pas être affectée à *vrai* dans une solution. □

**Propriété 5** *Si $M$ est sous forme échelonnée réduite avant d'appliquer les règles alors les propriétés suivantes sont vérifiées après avoir appliqué les règles (si aucun domaine n'est vide) : (i) pour chaque colonne $j \in [1, m]$, $vrai \in D(\Delta_j)$, (ii) $M$ est sous forme échelonnée réduite, (iii) pour toute ligne $i$ de $M$, $\#var_i \geq 2$.*

**Preuve.** (i) est assurée par R1 et R2, qui suppriment de $M$ les colonnes associées à des variables affectées à *faux*. (ii) est assurée par R2, qui fait entrer en base une nouvelle variable lorsqu'une colonne supprimée correspond à une variable de base. (iii) est assurée par R2 et R3 : lorsqu'une équation contient une seule variable, alors R3 enlève *vrai* du domaine de cette variable, et R2 supprime la ligne correspondant à l'équation et la colonne correspondant à la variable. □

**Propriété 6** *L'application de R1, R2 et R3 jusqu'à l'obtention d'un point fixe assure la satisfiabilité d'abstractXOR$_{X_\delta,C}(X_\Delta)$ si aucun domaine n'est vide.*

**Preuve.** Les propriétés 5 et 1 assurent que la contrainte est satisfaite par l'affectation $A$ telle que, pour toute variable $\Delta_j \in X_\Delta$, si $vrai \in D(\Delta_j)$ alors $A(\Delta_j) = vrai$ sinon $A(\Delta_j) = faux$. □

### 4.2 Maintien de la cohérence d'arc généralisée

Pour maintenir efficacement la cohérence d'arc généralisée (GAC), nous introduisons deux règles supplémentaires qui sont appliquées en plus de R1, R2 et R3 jusqu'à ce que soit un domaine devienne vide (une incohérence est détectée), soit plus aucune règle ne s'applique (la GAC est atteinte).

**R4.** Si $\exists i \in [1, n], var_i = \{j_1, j_2\} \wedge vrai_i = \{j_1\}$, alors supprimer *faux* de $D(\Delta_{j_2})$.

**R5.** Si $\exists i_1, i_2 \in [1, n], var_{i_1} \setminus \{c_{i_1}\} = var_{i_2} \setminus \{c_{i_2}\} \wedge D(\Delta_{c_{i_1}}) = \{vrai\} \wedge faux \in D(\Delta_{c_{i_2}})$ alors supprimer *faux* de $D(\Delta_{c_{i_2}})$.

**Propriété 7** *L'application de R4 et R5 ne change pas l'ensemble des solutions de la contrainte globale.*

**Preuve.** R4 supprime *faux* de $D(\Delta_{j_2})$ dans le cas où la ligne $i$ correspond à l'équation $\delta_{j_1} = \delta_{j_2}$ et $D(\Delta_{j_1}) = \{vrai\}$. Cette équation ne peut pas être satisfaite si $\Delta_{j_1} = vrai$ et $\Delta_{j_2} = faux$.

R5 supprime *faux* de $D(\Delta_{c_{i_2}})$ dans le cas où $D(\Delta_{c_{i_1}}) = \{vrai\}$ et les lignes $i_1$ et $i_2$ correspondent à

$$\begin{aligned} \delta_{c_{i_1}} &= \delta_{i'_1} \oplus \ldots \oplus \delta_{i'_x} \\ \delta_{c_{i_2}} &= \delta_{i'_1} \oplus \ldots \oplus \delta_{i'_x} \end{aligned}$$

Ces équations impliquent que $\delta_{c_{i_1}} = \delta_{c_{i_2}}$. Par conséquent, si $\Delta_{c_{i_1}} = vrai$, alors $\Delta_{c_{i_2}} \neq faux$. □

**Propriété 8** *L'application des règles R1 à R5 jusqu'à un point fixe assure la GAC d'abstractXOR$_{X_\delta,C}(X_\Delta)$*

**Preuve.** Montrons que pour toute variable $\Delta_i \in X_\Delta$ et toute valeur $v_i \in D(\Delta_i)$, le couple $(\Delta_i, v_i)$ possède un support, *i.e.*, une affectation $A : X_\Delta \rightarrow \{vrai, faux\}$ qui satisfait *abstractXOR*$_{X_\delta,C}(X_\Delta)$ et telle que $A(\Delta_i) = v_i$ et $\forall \Delta_j \in X_\Delta, A(\Delta_j) \in D(\Delta_j)$.

R1, R2 et R3 assurent qu'il existe une affectation $A$ qui est un support du couple $(\Delta_j, vrai)$ pour toute variable $\Delta_j$ telle que $vrai \in D(\Delta_j)$, et du couple $(\Delta_j, faux)$ pour toute variable $\Delta_j$ telle que $D(\Delta_j) = \{faux\}$ (cf preuve de la propriété 6).

Il reste à vérifier que pour toute variable $\Delta_j$ telle que $D(\Delta_j) = \{vrai, faux\}$, il existe un support pour le couple $(\Delta_j, faux)$. Nous distinguons pour cela trois cas, et montrons à chaque fois que si on supprime la colonne $j$ de $M$ et qu'on remet $M$ sous forme échelonnée réduite, la matrice résultante aura au moins deux variables par ligne de sorte qu'il existe un support.

• *Cas 1 : $\delta_j$ n'est pas une variable de base.* Soit $S = \{i \in [1, n] : M[i, j] = 1 \wedge \#var_i = 2\}$ l'ensemble des lignes comportant $\delta_j$ et exactement une autre variable, et soit $X = \{\Delta_{c_i} : i \in S\}$ l'ensemble des variables booléennes associées aux variables de base de ces lignes. On peut affecter chaque variable $\Delta_{c_i} \in X$ à *faux* (car R4 garantit que $faux \in D(\Delta_{c_i}) \Leftrightarrow faux \in D(\Delta_j)$), et supprimer de $M$ chaque ligne $i \in S$ et chaque colonne associée à une variable affectée à *faux*. $M$ est alors sous forme échelonnée réduite et a au moins deux variables par ligne.

• *Cas 2 : $\delta_j$ est une variable de base apparaissant dans une ligne $i$ contenant exactement 2 variables ($\delta_j$ et $\delta_k$).* Dans ce cas, R4 garantit que $faux \in D(\Delta_k) \Leftrightarrow faux \in D(\Delta_j)$ et donc $\Delta_k$ peut être affectée à *faux*. $\Delta_k$ n'est pas une variable de base, et son affectation à *faux* peut impliquer d'affecter à *faux* d'autres variables (cf cas 1) mais cela sera toujours possible grâce à R4 et, une fois supprimées les colonnes des variables affectées à *faux* et les lignes ne comportant que des colonnes supprimées, la matrice résultante sera sous forme échelonnée réduite et aura au moins deux variables par ligne.

• *Cas 3 : $\delta_j$ est une variable de base apparaissant dans une ligne $i$ contenant plus de 2 variables.* Dans ce cas, on supprime la colonne de la variable $\delta_j$, et il faut

**Maximiser** $\sum_{i\in[0,r-1],j,k\in[0,3]} \log_2(p(\delta X_{i,j,k}, \delta SX_{i,j,k}))$

$(C_1)$ $\forall i \in [0, r-2], \forall j, k \in [0, 3], \delta Z_{i,j,k} \oplus \delta K_{j,k} = \delta X_{i+1,j,k}$

$$(C_2)\ \forall i \in [0, r-2], \begin{array}{llll} \delta Y_{i,0,0} = \delta SX_{i,0,0} & \delta Y_{i,1,0} = \delta SX_{i,2,2} & \delta Y_{i,2,0} = \delta SX_{i,1,1} & \delta Y_{i,3,0} = \delta SX_{i,3,3} \\ \delta Y_{i,0,1} = \delta SX_{i,2,3} & \delta Y_{i,1,1} = \delta SX_{i,0,1} & \delta Y_{i,2,1} = \delta SX_{i,3,2} & \delta Y_{i,3,1} = \delta SX_{i,1,0} \\ \delta Y_{i,0,2} = \delta SX_{i,1,2} & \delta Y_{i,1,2} = \delta SX_{i,3,0} & \delta Y_{i,2,2} = \delta SX_{i,0,3} & \delta Y_{i,3,2} = \delta SX_{i,2,1} \\ \delta Y_{i,0,3} = \delta SX_{i,3,1} & \delta Y_{i,1,3} = \delta SX_{i,1,3} & \delta Y_{i,2,3} = \delta SX_{i,2,0} & \delta Y_{i,3,3} = \delta SX_{i,0,2} \end{array}$$

$(C_3)$ $\forall i \in [0, r-2], \forall j, k \in [0, 3], \delta Y_{i,(j+1)\%4,k} \oplus \delta Y_{i,(j+2)\%4,k} \oplus \delta Y_{i,(j+3)\%4,k} = \delta Z_{i,j,k}$

FIGURE 2 – Problème concret du calcul de la probabilité maximale d'un chemin différentiel pour Midori128

faire entrer une variable hors-base de la ligne $i$ dans la base (de façon similaire à ce qui est fait par R2). La matrice résultante sera sous forme échelonnée réduite, mais il reste à prouver qu'elle ne comporte pas de ligne avec une seule variable. R5 garantit cela : le seul cas où une ligne résultant du XOR de la ligne $i$ avec une autre ligne $k$ contenant la variable entrant dans la base peut n'avoir qu'une seule variable est le cas où les variables hors-base de la ligne $i$ sont exactement les mêmes que les variables hors-base de la ligne $k$ (ces variables s'annulent lors du XOR, et il ne reste plus que la variable de base $\delta_{c_k}$ de la ligne $k$). Dans ce cas, la règle 5 nous assure que $faux \in D(\Delta_{c_k})$, et nous pouvons donc affecter $\Delta_{c_k}$ à faux dans le support (et supprimer la ligne $k$ et la colonne $c_k$).

Dans les trois cas, un support de $(\Delta_j, faux)$ est l'affectation qui affecte à *vrai* toutes les variables restant dans $M$ après ces suppressions, et à *faux* toutes les autres variables. □

### 4.3 Implémentation

Les valeurs $var_i$ et $vrai_i$, pour chaque ligne $i$, sont maintenues incrémentalement.

L'implémentation des règles R1 à R5 implique essentiellement de parcourir des lignes ou des colonnes de $M$ : il faut parcourir une colonne pour la supprimer (dans R1 et R2), et parcourir une ligne pour XORer deux lignes (R2) ou décider si deux lignes ont les mêmes variables hors-base (R5). Pour une implémentation efficace de ces opérations, la matrice $M$ (qui est très creuse) est représentée à l'aide de listes chaînées (aussi bien pour les lignes que pour les colonnes), et nous utilisons les *Dancing Links* introduits par Knuth [14] afin de restaurer efficacement l'état de la matrice lors des retour-arrière, en effectuant les opérations inverses des opérations effectuées lors de la création du point de choix. Soit $l$ (resp. $c$) le nombre maximum de 1 dans une ligne (resp. colonne) de $M$. En utilisant les dancing links, la complexité en temps pour supprimer une colonne ou pour la restaurer est $\mathcal{O}(c)$, et la complexité en temps pour XORer deux lignes ou pour restaurer l'état de la ligne avant le XOR est $\mathcal{O}(l)$.

## 5 Application au calcul de chemins différentiels maximaux pour Midori

Midori [1] est un algorithme de chiffrement léger conçu pour consommer moins d'énergie qu'AES. Le chiffrement est itératif et, à chaque itération $i \in [0, r-1]$, les seules opérations qui sont effectuées sont des déplacements d'octets, des XOR entre octets, et une opération appelée *Sbox* qui substitue chaque octet par un autre octet selon une table donnée.

**Description du problème concret.** Le problème concret Midori128 est décrit dans la figure 2, $r$ étant le nombre d'itérations. Les octets de la clé $K$ et du texte $X_i$ à chaque itération $i$ sont regroupés dans des matrices de 4x4 octets. Les variables sont :

- $\delta K_{j,k}$ avec $j, k \in [0, 3]$, qui représente la différence dans l'octet ligne $j$ et colonne $k$ de la clé ;
- $\delta X_{i,j,k}$ avec $i \in [0, r-1], j, k \in [0, 3]$, qui représente la différence en entrée (resp. en sortie de la $i$ème itération et en sortie du chiffrement) dans l'octet ligne $j$ et colonne $k$ du texte quand $i = 0$ (resp. $i \in [1, r-2]$ et $i = r-1$) ;
- $\delta SX_{i,j,k}$ avec $i \in [0, r-1], j, k \in [0, 3]$, qui représente la différence en sortie de la Sbox, quand la différence en entrée est $\delta X_{i,j,k}$ ;
- $\delta Y_{i,j,k}$, avec $i \in [0, r-2]$ et $j, k \in [0, 3]$), qui représente la différence en sortie de l'opération *ShuffleCell* quand la différence en entrée est $\delta SX_{i,j,k}$ ;
- $\delta Z_{i,j,k}$, avec $i \in [0, r-2]$ et $j, k \in [0, 3]$), qui représente la différence en sortie de *MixColumns* quand la différence en entrée est $\delta Y_{i,j,k}$.

Chacune de ces variables est une variable entière pouvant prendre une valeur comprise entre 0 et 255.

L'objectif est de trouver la solution maximisant la probabilité d'observer la différence en sortie $\delta X_{r-1}$ connaissant les différences en entrée $\delta X_0$ et $\delta K$, ce qui revient à maximiser la somme des logarithmes en base 2 de la probabilité d'observer la différence en sortie de chaque Sbox $\delta SX_{i,j,k}$ connaissant la différence en entrée $\delta X_{i,j,k}$. En effet, la Sbox est la seule opération non linéaire pour laquelle la différence

$(C'_0)$ $obj = \sum_{i\in[0,r-1],j,k\in[0,3]} \Delta X_{i,j,k}$

$(C'_1)$ $\forall i \in [0, r-2], \forall j, k \in [0, 3], \Delta Z_{i,j,k} \oplus_{\mathcal{A}} \Delta K_{j,k} \oplus_{\mathcal{A}} \Delta X_{i+1,j,k} = 0$

$$(C'_2)\ \forall i \in [0, r-2], \begin{array}{llll} \Delta Y_{i,0,0} = \Delta X_{i,0,0} & \Delta Y_{i,1,0} = \Delta X_{i,2,2} & \Delta Y_{i,2,0} = \Delta X_{i,1,1} & \Delta Y_{i,3,0} = \Delta X_{i,3,3} \\ \Delta Y_{i,0,1} = \Delta X_{i,2,3} & \Delta Y_{i,1,1} = \Delta X_{i,0,1} & \Delta Y_{i,2,1} = \Delta X_{i,3,2} & \Delta Y_{i,3,1} = \Delta X_{i,1,0} \\ \Delta Y_{i,0,2} = \Delta X_{i,1,2} & \Delta Y_{i,1,2} = \Delta X_{i,3,0} & \Delta Y_{i,2,2} = \Delta X_{i,0,3} & \Delta Y_{i,3,2} = \Delta X_{i,2,1} \\ \Delta Y_{i,0,3} = \Delta X_{i,3,1} & \Delta Y_{i,1,3} = \Delta X_{i,1,3} & \Delta Y_{i,2,3} = \Delta X_{i,2,0} & \Delta Y_{i,3,3} = \Delta X_{i,0,2} \end{array}$$

$(C'_3)$ $\forall i \in [0, r-2], \forall j, k \in [0, 3], \Delta Y_{i,(j+1)\%4,k} \oplus_{\mathcal{A}} \Delta Y_{i,(j+2)\%4,k} \oplus_{\mathcal{A}} \Delta Y_{i,(j+3)\%4,k} \oplus_{\mathcal{A}} \Delta Z_{i,j,k} = 0$

FIGURE 3 – Problème abstrait du calcul d'un chemin différentiel maximal pour Midori128

en sortie n'est pas connue avec certitude quand on connait l'entrée : pour la Sbox, la probabilité d'observer une différence en sortie $\delta_{out}$ étant donnée une différence en entrée $\delta_{in}$ est notée $p(\delta_{in}, \delta_{out})$ et appartient à $\{0, 2^{-2}, 2^{-3}, 2^{-4}, 2^{-5}, 2^{-6}, 1\}$. Le seul cas où $p(\delta_{in}, \delta_{out}) = 1$ est le cas où $\delta_{in} = \delta_{out} = 0$.

La contrainte $(C_1)$ correspond à l'opération de chiffrement *AddKey* qui permet d'obtenir $\delta X_{i+1,j,k}$ en XORant $\delta Z_{i,j,k}$ et $\delta K_{j,k}$. La contrainte $(C_2)$ correspond à l'opération de chiffrement *ShuffleCells*, et lie chaque variable $\delta Y_{i,j,k}$ à une variable différente $\delta SX_{i,j',k'}$. La contrainte $(C_3)$ correspond à l'opération de chiffrement *MixColumns* qui permet d'obtenir $\delta Z_{i,j,k}$ en XORant les octets $\delta Y_{i,j',k}$ tels que $j' \in [0, 3] \setminus \{j\}$.

**Description du problème abstrait.** Lors de la première étape, chaque octet $\delta X_{i,j,k}, \delta Y_{i,j,k}, \delta Z_{i,j,k}$ et $\delta K_{j,k}$ est abstrait par une variable booléenne $\Delta X_{i,j,k}, \Delta Y_{i,j,k}, \Delta Z_{i,j,k}$ et $\Delta K_{j,k}$ indiquant si l'octet est égal à 0 ou non. L'opération Sbox est ignorée lors de cette étape car $(\delta X_{i,j,k} = 0) \Leftrightarrow (\delta SX_{i,j,k} = 0)$. Pour simplifier, nous utilisons uniquement $\Delta X_{i,j,k}$ qui est associé à la fois à $\delta X_{i,j,k}$ et $\delta SX_{i,j,k}$.

Le modèle mathématique du problème abstrait est décrit dans la figure 3. La variable entière *obj* (dont le domaine est $[1, 16 * r]$) représente le nombre de différences dans des octets qui passent par une Sbox (*i.e.*, les octets $\Delta X_{i,j,k}$) : l'objectif de la première étape est d'énumérer toutes les solutions abstraites pour lesquelles *obj* est minimale (car quand $\Delta X_{i,j,k} = faux$, $p(\delta X_{i,j,k}, \delta SX_{i,j,k}) = 1$ alors que quand $\Delta X_{i,j,k} = vrai$, $p(\delta X_{i,j,k}, \delta SX_{i,j,k}) \in \{0, 2^{-2}, 2^{-3}, 2^{-4}, 2^{-5}, 2^{-6}\}$.

La contrainte $(C'_0)$ assure que *obj* est égal au nombre de différences dans les $\Delta X_{i,j,k}$. Les contraintes $(C'_1)$ et $(C'_3)$ sont les abstractions des contraintes $(C_1)$ et $(C_3)$, où le XOR concret $\oplus$ est remplacé par le XOR abstrait $\oplus_{\mathcal{A}}$. La contrainte $(C'_2)$ est identique à $(C_2)$.

**Modèle CP global.** Un premier modèle CP, appelé *Global*, est obtenu à partir de celui de la Figure 3 en remplaçant les contraintes $(C'_1)$ et $(C'_3)$ par la contrainte $abstractXOR_{X_\delta,C}(X_\Delta)$ où $X_\Delta$ (resp. $X_\delta$) contient l'ensemble des variables booléennes (resp. octets) intervenant dans les contraintes $(C'_1)$ et $(C'_3)$ (resp. $(C_1)$ et $(C_3)$), et $C = \{C_1, C_3\}$.

**Modèle CP basic.** Un deuxième modèle, appelé *Basic*, est obtenu en remplaçant les contraintes $(C'_1)$ et $(C'_3)$ par : $\forall i \in [0, r-2], \forall j, k \in [0, 3]$, $\Delta Z_{i,j,k} + \Delta K_{j,k} + \Delta X_{i+1,j,k} \neq 1$ et $\Delta Y_{i,(j+1)\%4,k} + \Delta Y_{i,(j+2)\%4,k} + \Delta Y_{i,(j+3)\%4,k} + \Delta Z_{i,j,k} \neq 1$. Ces contraintes correspondent à la relation définie par la table de vérité de l'opérateur abstrait $\oplus_{\mathcal{A}}$.

**Modèle CP avancé.** Le modèle Basic trouve un grand nombre de solutions abstraites qui sont incohérentes au niveau concret. Afin de diminuer ce nombre de solutions incohérentes, il est possible d'ajouter des contraintes liées à la propriété quasi-MDS (*Maximum Distance Separable*) de l'opération *MixColumns*, comme cela a été proposé dans [9]. Cette propriété est une conséquence mathématique de la définition de *MixColumns* qui a été prouvée par des cryptographes, et se traduit par un grand nombre de contraintes qui sont loin d'être triviales à formuler. Nous ne pouvons décrire ce modèle (que nous appelons *Avancé*) ici, et nous invitons le lecteur à lire [9] pour plus de détails.

**Solveurs considérés.** Les modèles CP Basic, Avancé, et Global ont été implémentés en Choco 4 [16]. Pour Basic et Avancé, nous avons utilisé l'heuristique *WeightedDegree* [5] qui donne de très bons résultats. Pour Global, nous utilisons une heuristique similaire : nous maintenons pour chaque variable de la contrainte globale le nombre d'incohérences où elle a été impliquée, et nous choisissons en premier les variables ayant participé au plus grand nombre d'incohérences.

Le modèle Avancé a également été implémenté en Picat-SAT [18], qui utilise le solveur SAT Lingeling [2].

**Comparaison expérimentale.** Nous comparons ces modèles pour différentes valeurs de $r$ dans la table 1, pour le problème consistant à énumérer toutes les affectations des variables $\Delta X_{i,j,k}$ appartenant à une solution

| | Modèle Basic en Choco | | | Modèle Global en Choco | | | | | Modèle Avancé de [9] | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | Vérif (R1 à R3) | | GAC (R1 à R5) | | | Choco | | SAT |
| $r$ | #s | t | #pc | #s | t | #pc | t | #pc | #s | t | #pc | t |
| 3 | 64 | 0 | 1 838 | 28 | 0 | 4 865 | 0 | 2 229 | 38 | 0 | 1 400 | 20 |
| 4 | 30 | 0 | 11 476 | 16 | 1 | 19 959 | 0 | 7 038 | 16 | 0 | 4 830 | 4 |
| 5 | 26 | 0 | 16 565 | 16 | 3 | 57 406 | 1 | 10 789 | 16 | 1 | 24 137 | 6 |
| 6 | 122 | 2 | 71 933 | 16 | 15 | 211 577 | 3 | 40 058 | 16 | 4 | 62 897 | 10 |
| 7 | 74 | 7 | 334 529 | 16 | 79 | 888 655 | 11 | 119 402 | 16 | 42 | 368 998 | 17 |
| 8 | 32 | 38 | 1 879 589 | 16 | 1 005 | 8 809 858 | 22 | 217 520 | 16 | 332 | 2 571 491 | 23 |
| 9 | 282 | 248 | 5 395 439 | 16 | 3 956 | 34 339 652 | 81 | 637 737 | 16 | 589 | 3 674 192 | 28 |
| 10 | 218 | 772 | 21 795 314 | 16 | 9 401 | 74 481 543 | 467 | 2 940 394 | 16 | 3 889 | 18 460 694 | 37 |
| 11 | 74 | 1 676 | 55 959 203 | 16 | >12H | | 2 498 | 16 165 905 | 16 | 16 272 | 59 210 153 | 50 |
| 12 | - | >12H | | 16 | >12H | | 4 271 | 17 066 102 | 16 | >12H | | 73 |

Table 1 – Comparaison des modèles Basic, Global et Avancé implémentés en Choco 4, ainsi que du modèle Avancé implémenté en Picat-SAT : nombre de solutions (#s), temps (t en secondes) et nombre de points de choix (#pc). Pour Global, on considère 2 propagateurs : Vérif applique les règles R1 à R3, et GAC les règles R1 à R5. La durée maximale d'exécution est de 12H.

(les valeurs des autres variables n'étant pas utiles lors de la seconde étape, nous n'énumérons pas leurs valeurs). Basic trouve beaucoup plus de solutions qu'Avancé et Global car l'abstraction qu'il considère est plus grossière. Global et Avancé trouvent le même nombre de solutions quand $r \geq 4$ mais Global trouve moins de solutions quand $r = 3$ ce qui montre que l'abstraction faite par Avancé est moins bonne. De fait, toutes les solutions trouvées par Global sont cohérentes au niveau concret (alors qu'Avancé trouve 10 solutions qui ne sont pas concrétisables quand $r = 3$).

Nous avons comparé deux propagateurs pour Global : Vérif (qui vérifie la faisabilité en appliquant les règles R1 à R3) et GAC (qui maintient la GAC en appliquant les règles R1 à R5). GAC a tendance à explorer moins de points de choix que Vérif, surtout pour les plus grosses instances. Comme la propagation de GAC est plus coûteuse, GAC n'est plus rapide que Vérif que pour les deux plus grosses instances.

Global (avec Vérif comme avec GAC) est clairement plus rapide que le modèle Avancé implémenté en Choco. En revanche, il n'est pas compétitif avec le modèle Avancé quand on utilise le solveur SAT Lingeling pour le résoudre. En effet, le modèle Avancé est essentiellement composé de variables booléennes, et ses contraintes sont de deux formes uniquement :

— $\sum_{\Delta_i \in S} \Delta_i \neq 1$
— $\sum_{\Delta_i \in S} \Delta_i = v$ où $v$ est une variable entière

où $S$ est un ensemble de variables Booléennes. Ces contraintes sont traduite en formules booléennes en combinant des encodages *at-most* et *at-least* qui sont très étudiés dans la communauté SAT [19]. Cela explique que les solveurs SAT soient particulièrement adaptés pour résoudre le modèle Avancé. De plus, l'utilisation du *Conflict Driven Clause Learning* [15] permet d'apprendre des *nogoods* qui accélèrent probablement la recherche.

Notons cependant que le modèle Global est très simple et est dérivé de façon immédiate de la définition de Midori, tandis que le modèle Avancé a demandé un important travail pour extraire les propriétés de *MixColumns* et les modéliser en termes de contraintes. Notre objectif est de faciliter l'utilisation de la programmation par contraintes par des cryptographes, et *abstractXOR* répond à cet objectif.

## 6 Conclusion

Les premiers résultats expérimentaux sur Midori128 montrent l'intéret de notre contrainte globale : elle simplifie énormément la conception du modèle (qui est dérivé de façon immédiate à partir de la définition du problème concret) tout en réduisant le nombre de solutions abstraites. Le modèle Basic, qui est également dérivé de façon immédiate à partir de la définition du problème concret, admet beaucoup plus de solutions abstraites et la majorité de ces solutions ne sont pas concrétisables ce qui rend la seconde étape du processus de résolution plus difficile. Le modèle Avancé de [9] trouve le même nombre de solutions abstraites que notre contrainte globale (sauf pour $r = 3$ où il trouve plus de solutions), mais il est bien plus difficile à concevoir car il exploite des propriétés dérivées de l'opération *MixColumns* qui sont loin d'être triviales.

En terme de passage à l'échelle, la contrainte *abstractXOR* améliore les performances de Choco, par rapport au modèle Avancé, mais elle n'est pas compétitive avec un solveur SAT. Notre objectif est donc d'améliorer les performances. Une première piste est d'améliorer les heuristiques d'ordre, qui ont un fort im-

pact sur les résultats. Une seconde piste est d'apprendre des *nogoods* (ou explications) lorsque des incohérences sont détectées afin de pouvoir retourner directement au nœud correspondant à la décision à l'origine de l'échec. Les bonnes performances de SAT nous laissent penser que cela devrait fortement améliorer les performances de Choco.

Enfin, nous souhaitons évaluer les performances d'*abstractXOR* pour rechercher des chemins différentiels maximaux pour AES, et nous comparer avec le modèle de [10, 11], mais aussi pour résoudre d'autres problèmes de cryptanalyse différentielle tels que ceux étudiés dans [6] ou [17], par exemple.

**Remerciements :** Le travail décrit dans cet article a été réalisé dans le contexte de l'ANR DeCrypt (ANR-18-CE39-0007). Nous remercions Charles Prud'homme pour ses réponses toujours pertinentes à nos nombreuses questions sur Choco.

## Références

[1] S. Banik, A. Bogdanov, T. Isobe, K. Shibutani, H. Hiwatari, T. Akishita et F. Regazzoni : Midori : A block cipher for low energy. *In ASIACRYPT*, volume 9453 de *LNCS*, pages 411–436. Springer, 2015.

[2] A. Biere : Yet another local search solver and lingeling and friends entering the sat competition 2014. pages 39–40, 01 2014.

[3] E. Biham : New types of cryptoanalytic attacks using related keys (extended abstract). *In EUROCRYPT*, volume 765 de *LNCS*, pages 398–409. Springer, 1993.

[4] E. Biham et A. Shamir : Differential cryptoanalysis of feal and n-hash. *In EUROCRYPT*, volume 547 de *LNCS*, pages 1–16. Springer, 1991.

[5] F. Boussemart, F. Hemery, C. Lecoutre et L. Sais : Boosting systematic search by weighting constraints. *In ECAI*, pages 146–150, 2004.

[6] C. Cid, T. Huang, T. Peyrin, Y. Sasaki et L. Song : Boomerang connectivity table : A new cryptanalysis tool. *In EUROCRYPT*, volume 10821 de *LNCS*, pages 683–714. Springer, 2018.

[7] H. Comon-Lundh et V. Shmatikov : Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. *In 18th Annual IEEE Symposium of Logic in Computer Science, 2003. Proceedings.*, pages 271–280, juin 2003.

[8] FIPS 197 : Advanced Encryption Standard. Federal Information Processing Standards Publication 197, 2001. U.S. Department of Commerce/N.I.S.T.

[9] D. Gérault : *Security Analysis of Contactless Communication Protocols.* Thèse de doctorat, Université Clermont Auvergne, 2018.

[10] D. Gérault, P. Lafourcade, M. Minier et C. Solnon : Revisiting AES related-key differential attacks with constraint programming. *Inf. Process. Lett.*, 139:24–29, 2018.

[11] D. Gérault, M. Minier et C. Solnon : Constraint programming models for chosen key differential cryptanalysis. *In CP*, volume 9892 de *LNCS*, pages 584–601. Springer, 2016.

[12] R. M. Karp : Reducibility among combinatorial problems. *In Symposium on the Complexity of Computer Computations*, pages 85–103, 1972.

[13] L. Knudsen : Truncated and higher order differentials. *In Fast Software Encryption*, pages 196–211. Springer, 1995.

[14] D. E Knuth : Dancing links. *Millenial Perspectives in Computer Science*, 18:4, 2000.

[15] Tero Laitinen, Tommi Junttila et Ilkka Niemelä : Conflict-Driven XOR-Clause Learning (extended version). juillet 2014.

[16] C. Prud'homme, J.-G. Fages et X. Lorca : *Choco Documentation.* TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2016.

[17] Y. Todo, T. Isobe, Y. Hao et W. Meier : Cube attacks on non-blackbox polynomials based on division property. *In CRYPTO*, volume 10403 de *LNCS*, pages 250–279. Springer, 2017.

[18] N.-F. Zhou, H. Kjellerstrand et J. Fruhman : *Constraint Solving and Planning with Picat.* Springer, 2015.

[19] Neng-Fa Zhou et Håkan Kjellerstrand : Optimizing SAT encodings for arithmetic constraints. *In Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017*, volume 10416 de *Lecture Notes in Computer Science*, pages 671–686. Springer, 2017.

# PyCSP$^3$
# Modéliser des problèmes combinatoires sous contraintes en Python

**Christophe Lecoutre**[1] **et Nicolas Szczepanski**[1]

[1] CRIL-CNRS, UMR 8188, Université d'Artois, F-62307 Lens, France
{lecoutre,szczepanski}@cril.fr

**Résumé**

Dans cet article, nous introduisons PyCSP$^3$, une API Python pour construire des modèles de problèmes combinatoires sous contraintes. En traduisant des modèles PyCSP$^3$ au format XCSP$^3$ au moyen du compilateur rendu disponible, il est possible d'utiliser tout solveur de contraintes qui reconnaît XCSP$^3$. Cette API a les mêmes fonctionnalités que celle proposée en Java (JvCSP$^3$). Nous sommes convaincus que cette API peut être utile à un large public scientifique.

**Abstract**

In this paper, we introduce PyCSP$^3$, a Python API for building models of constrained combinatorial problems. By translating PyCSP$^3$ models into the XCSP$^3$ format by means of the available compiler, one can directly use any constraint solver that understands XCSP$^3$. This API has the same functionalities than the one proposed in Java (JvCSP$^3$). We are convinced that this API can be quite useful for a wide scientific audience.

## 1 Introduction

XCSP$^3$ [**?**] est un format qui permet une représentation "intégrée" des problèmes combinatoires sous contraintes. Un point important est que XCSP$^3$ préserve la structure des modèles, en gérant les concepts de tableaux de variables et groupes/blocs de contraintes. De plus, l'écosystème de XCSP$^3$ est bien fourni : il comporte des outils auxiliaires (parseurs et vérificateurs), un site web www.xcsp.org pour notamment sélectionner des instances en fonction de critères précis, et des compétitions de solveurs [1]. Parmi les derniers développements, on trouve une API de modélisation en Java, appelée JvCSP$^3$ [**?**], qui était précédemment nommée MCSP$^3$. Toutefois, parce que Python est un langage adapté aux tâches de modélisation par sa nature mathématique (en particulier, par les possibilités de construire des formes en compréhension), et aussi parce que la communauté Python va bien au-delà de l'informatique, nous pensons qu'une API supplémentaire en Python 3 pour modéliser des problèmes combinatoires sous contraintes est important (un atout) pour promouvoir la PPC (programmation par contraintes) et ses outils. L'API proposée, appelée PyCSP$^3$, s'inspire à la fois de JvCSP$^3$ et Numberjack [**?**], une librairie de modélisation écrite en Python. La sortie officielle de PyCSP$^3$ devrait être effectuée au cours du printemps.

1. Vous pouvez trouver les résultats de la compétition XCSP$^3$ 2018 sur http://www.cril.univ-artois.fr/XCSP18

## 2 Introduction douce à PyCSP$^3$ et XCSP$^3$

Dans cette section, de manière à donner au lecteur une vision générale de PyCSP$^3$ et XCSP$^3$, nous introduisons deux problèmes illustratifs.

### 2.1 All-Interval Series

Comme première illustration, nous considérons un problème académique : All-Interval Series. Il s'agit du problème 007 sur CSPLib [**?**]. Le problème est formulé comme suit : "étant donné un entier $n \in \mathbb{N}$, il faut trouver un vecteur $x = \langle x_1, x_2, \ldots, x_n \rangle$, tel que $x$ soit une permutation de $\{0, 1, \ldots, n-1\}$ et $y = \langle y_1, y_2, \ldots, y_{n-1} \rangle = \langle |x_2 - x_1|, |x_3 - x_2|, \ldots, |x_n - x_{n-1}| \rangle$ soit une permutation de $\{1, 2, \ldots, n-1\}$." Il est clair que les données nécessaires pour définir une instance spécifique sont représentées par un simple entier $n$. Aussi, avec $n$ comme unique "paramètre" de ce

problème, la structure d'un modèle naturel est donnée par :

— $x$, un tableau de $n$ variables entières, chacune avec le domaine $\{0, 1, \ldots, n-1\}$
— $y$, un tableau de $n-1$ variables entières, chacune avec le domaine $\{1, 2, , \ldots, n-1\}$
— les contraintes `allDifferent`$(x)$ et `allDifferent`$(y)$
— un groupe de contraintes liant les variables de $x$ et $y$ : $\forall i \in 1..n-1, y[i] = \texttt{dist}(x[i], x[i+1])$

où `dist` est l'opérateur de distance : $\texttt{dist}(a, b) = |a - b|$. Utiliser PyCSP³ pour écrire un tel modèle est très simple. La sous-figure 1a montre le code pour le modèle (placé dans un fichier Python), après importation des fonctions requises de `modeler.api.functions`.

Notons tout d'abord que les valeurs des paramètres sont obtenues automatiquement sous la forme de champs d'un objet `data`[2], et que les variables et contraintes sont déclarées simplement en séquence. Comme vous pouvez l'observer, nous utilisons certaines fonctions du module `functions` pour poster à la fois les variables et les contraintes : `array()`, `allDifferent()`, et ainsi de suite. La fonction `array()` accepte plusieurs paramètres, le premier est une liste représentant la dimension (`[5]` pour un tableau de taille 5, `[5,5]` pour une matrice $5 \times 5$) et les suivants représentent les domaines des variables (`integer`, `range`, ...). La fonction `equal(...)` est un raccourci pour `intension(eq(...))`. Utiliser la contrainte `intension()` est la manière générale de poster une contrainte définie en intention (i.e., définie par un prédicat), et peut être utilisée pour n'importe quelle expression booléenne via des fonctions de la librairie. De telles expressions peuvent impliquer des variables, valeurs, opérateurs arithmétiques, opérateurs relationnels et logiques. Pour finir, $\texttt{dist}(x[i], x[i+1])$ est équivalent à $\texttt{abs}(\texttt{sub}((x[i+1], x[i]))$, avec `abs` représentant l'opérateur de valeur absolue.

Une fois que le modèle est construit, il est possible de générer des instances (fichiers) XCSP³ en fournissant des données spécifiques. Cela est rendu possible en utilisant des commandes telles que la suivante :

```
python3 compiler.py AllInterval.py -data=5
```

Ici, nous nous intéressons à l'instance de `AllInterval` pour $n = 5$. La sous-figure 1b montre le fichier obtenu après compilation, qui est une instance (fichier) XCSP³ appelée AllInterval-5.xml. Bien sûr, il est aussi possible de choisir le nom du fichier à générer. Comme on peut l'observer, la structure du modèle est toujours présente et visible, car les tableaux de variables et les groupes/-blocs de contraintes peuvent être directement integrés en XCSP³. Pour les deux contraintes `allDifferent`, des formes compactes de listes de variables sont utilisées. On peut construire de telles formes compactes en plaçant soit des intervalles d'indice entre crochets (e.g., $x[1..3]$) ou rien du tout (e.g., $x[]$) quand tous les indices doivent être considérés. Ainsi, sur notre exemple, $x[]$ représente $x[0]$ $x[1]$ $x[2]$ $x[3]$ $x[4]$.

2. Comme nous le verrons plus loin, il est bien sûr possible d'utiliser n'importe quelle forme de données structurées, au moyen d'un fichier JSON.

## 2.2 Balanced Academic Curriculum Problem

Le but du problème BACP (le problème 030 sur CSPLib [**?**]) est de concevoir un programme d'études académique équilibré en affectant des périodes à des cours de manière à ce que la charge académique de chaque période soit équilibrée.

*"An academic curriculum is defined by a set of courses and a set of prerequisite relationships among them. Courses must be assigned within a maximum number of academic periods. Each course has associated a number of credits or units that represent the academic effort required to successfully follow it. The curriculum must obey the following regulations :*

— *Minimum academic load : a minimum number of academic credits per period is required to consider a student as full time.*
— *Maximum academic load : a maximum number of academic credits per period is allowed in order to avoid overload.*
— *Minimum number of courses : a minimum number of courses per period is required to consider a student as full time.*
— *Maximum number of courses : a maximum number of courses per period is allowed in order to avoid overload.*

*The goal is to assign a period to every course in a way that the minimum and maximum academic load for each period, the minimum and maximum number of courses for each period, and the prerequisite relationships are satisfied. An optimal balanced curriculum minimizes the maximum academic load for all periods."*

Lorsqu'on analyse ce problème, on identifie comme paramètres le nombre de périodes (un entier), le nombre minimum et maximum de crédits (deux entiers), le nombre minimum et maximum de cours (deux entiers), les crédits pour chaque cours (un tableau d'entiers à une dimension) et les pré-requis (un tableau d'entiers à deux dimensions). En PyCSP³, tout comme en JvCSP³, les données pour de tels paramètres complexes sont données en JSON. Par exemple, les données pour une instance spécifique très simple sont données par le fichier suivant nommé *easy.json* qui contient :

```
{
  "nPeriods": 4,
  "minCredits": 2,
```

```
from modeler.api.functions import *

# Data
n = data.n

# Variables
x = array([n], dom(range(n)),"x[i] is the ith value of the series")
y = array([n-1], dom(range(1, n)),"y[i] is the distance between x[i] and x[i+1]")

# Constraints
allDifferent(x)
allDifferent(y)
group(
  equal(y[i], dist(x[i], x[i+1])) for i in range(n-1)
)
```

(a) Le fichier `AllInterval.py` en PyCSP³

```
<instance format="XCSP3" type="CSP">
  <variables>
    <array id="x" note="x[i] is the ith value of the series"
      size="[5]"> 0..4 </array>
    <array id="y" note="y[i] is the distance between x[i] and x[i+1]"
      size="[4]"> 1..4 </array>
  </variables>
  <constraints>
    <allDifferent> x[] </allDifferent>
    <allDifferent> y[] </allDifferent>
    <group>
      <intension> eq(%0,dist(%1,%2)) </intension>
      <args> y[0] x[0] x[1] </args>
      <args> y[1] x[1] x[2] </args>
      <args> y[2] x[2] x[3] </args>
      <args> y[3] x[3] x[4] </args>
    </group>
  </constraints>
</instance>
```

(b) Le fichier `AllInterval-5.xml` en XCSP³ obtenu pour $n = 5$

FIGURE 1 – Le modèle PyCSP³ du problème 'All-Interval Series' et un résultat de compilation en XCSP³

```
  "maxCredits": 5,
  "minCourses": 2,
  "maxCourses": 3,
  "credits": [2,3,1,3,2,3,3,2,1],
  "prequisites":
      [[2,0],[4,1],[5,2],[6,4]]
}
```

Un modèle possible pour BACP est donné par la figure 2. Les champs de l'objet `data` correspondent exactement à ceux de l'objet principal du fichier JSON. Dans ce modèle, seules des contraintes `intension` et `sum` sont postées. Il faut noter que `equivalence()` et `lessThan(...)` sont respectivement des raccourcis pour `intension(iff(...))` et `intension(lt(...))`.

Lorsqu'on construit des groupes, on peut bénéficier des listes en compréhension de Python. Cela nous permet d'itérer facilement en combinant potentiellement plusieurs boucles. Pour poster une contrainte `sum`, on a toujours besoin d'une liste principale de variables dont il faut faire la somme (le premier argument positionnel), un second argument nommé pour représenter éventuellement des coefficients, et un dernier argument obligatoire pour représenter une condition. En Python, les arguments nommés peuvent être spécifiés dans n'importe quel ordre et peuvent être optionnels. Dans notre modèle, la condition est simplement représentée par un argument nommé permettant un raccourci : `equalTo=X` est équivalent à l'argument nommé `condition=(EQ, X)`. Une condition est toujours un couple de la forme (opérateur, opérande) où le premier élément est un

```
from modeler.api.functions import *

# Data
nPeriods = data.nPeriods
minCredits, maxCredits = data.minCredits, data.maxCredits
minCourses, maxCourses = data.minCourses, data.maxCourses
credits, prerequisites = data.credits, data.prerequisites
nCourses, nPrerequisites  = len(credits), len(prerequisites)

# Variables
s = array([nCourses], dom(range(nPeriods)),
  "s[c] is the period (schedule) for course c")
co = array([nPeriods], dom(range(minCourses, maxCourses + 1)),
  "co[p] is the number of courses at period p")
cr = array([nPeriods], dom(range(minCredits, ubCredits + 1)),
  "cr[p] is the number of credits at period p")
pc = array([nPeriods, nCourses], dom(0, 1),
  "pc[p][c] is 1 iff the course c is at period p")

# Constraints
group(
  equivalence(pc[p][c], eq(s[c], p)) for p in range(nPeriods) for c in range(nCourses)
).tag(CHANNELING)
group(
  sum(columnOf(pc, c), equalTo=1) for c in range(nCourses)
).note("ensuring that each course is assigned to a period")
group(
  sum(pc[p], equalTo=co[p]) for p in range(nPeriods)
).note("counting the number of courses in each period")
group(
  sum(pc[p], coeffs=credits, equalTo=cr[p]) for p in range(nPeriods)
).note("counting the number of credits in each period")
group(
  lessThan(s[prerequisites[i][0]], s[prerequisites[i][1]]) for i in range(nPrerequisites)
).note("handling prerequisites")

# Objectives and Annotations
minimize(MAXIMUM, cr)
  .note("minimizing the maximum number of credits in periods")

annotations(decisionVariables=s)
```

FIGURE 2 – Le fichier `Bacp.py` en PyCSP³

opérateur relationnel (ici, c'est EQ pour l'égalité) et le second élément est un entier (or une variable) définissant une limite.

BACP est un problème d'optimisation : l'objectif est de minimiser la valeur maximale prise par les variables du tableau *cr*. Cet objectif est déclaré par l'avant-dernière instruction. La dernière instruction indique que le tableau *s* représente un ensemble de variables de décision. Cela signifie que lorsque toutes les variables de *s* sont affectées, le domaine de toute autre variable devient singleton (pourvu qu'un niveau de filtrage de type Forward Checking [**?**] soit garanti par le solveur).

Pour compiler, on exécute simplement :

```
python3 compiler.py Bacp.py -data=easy.json
```

Après compilation, on obtient un fichier XCSP³ appelé *Bacp-easy.xml* dont le contenu est donné par la figure 3. Par souci de simplicité, nous avons écarté les commentaires des variables, et certaines lignes (ceci étant indiqué par l'ellipse "... `// ellipsis`"). En XCSP³, lorsqu'une contrainte de somme ou de comptage est représentée, telle que `sum` ou `count`, on a toujours un élément `<condition>` qui contient un opérateur relationnel et une limite. Si l'on considère

```
<instance format="XCSP3" type="COP">
  <variables>
    <array id="s" size="[9]"> 0..3 </array>
    <array id="co" size="[4]"> 2 3 </array>
    <array id="cr" size="[4]"> 2..5 </array>
    <array id="pc" size="[4][9]"> 0 1 </array>
  </variables>
  <constraints>
    <group class="channeling">
      <intension> iff(%0,eq(%1,%2)) </intension>
      <args> pc[0][0] s[0] 0 </args>
      <args> pc[0][1] s[1] 0 </args>
      ... // ellipsis
    </group>
    <group note="ensuring that each course is assigned to a period">
      <sum>
        <list> %... </list>
        <condition> (eq,1) </condition>
      </sum>
      <args> pc[][0] </args>
      <args> pc[][1] </args>
      ... // ellipsis
      </group>
    <block note="counting the number of courses in each period">
      <sum>
        <list> pc[0][] </list>
        <condition> (eq,co[0]) </condition>
      </sum>
      ... // ellipsis
    </block>
    <block note="counting the number of credits in each period">
      <sum>
        <list> pc[0][] </list>
        <coeffs> 2 3 1 3 2 3 3 2 1 </coeffs>
        <condition> (eq,cr[0]) </condition>
      </sum>
      ... // ellipsis
    </block>
    <group note="handling prerequisites">
      <intension> lt(%0,%1) </intension>
      <args> s[2] s[0] </args>
      <args> s[4] s[1] </args>
      ... // ellipsis
    </group>
  </constraints>
  <objectives note="minimizing the maximum number of credits in periods">
    <minimize type="maximum"> cr[] </minimize>
  </objectives>
  <annotations>
    <decision> s[] </decision>
  </annotations>
  </instance>
```

FIGURE 3 – Le fichier `Bacp-easy.xml` en XCSP³ obtenu pour `easy.json` (fichier de données)

la première contrainte du bloc ("counting the number of periods"), elle garantie que la somme des variables de la liste spécifiée respecte la condition donnée. Ici, nous avons $\sum_{j=0}^{8} pc[0][j] = co[0]$. Pour la première contrainte du bloc ("counting the number of credits"), des coefficients sont impliqués, ce qui donne $\sum_{j=0}^{8} pc[0][j] \times credit[j] = cr[0]$.

## 3 Chaînes de compilation connexes

Dans cette section, nous montrons comment la chaîne de compilation XCSP$^3$ (i.e., le format XCSP$^3$ et son écosystème, incluant les API JvCSP$^3$ et PyCSP$^3$ avec leurs compilateurs) est connectée aux chaînes de compilation développées par la communauté PPC. Nous comparons tout d'abord XCSP$^3$ à FlatZinc, et discutons ensuite de MiniZinc [**?**] et Conjure [**?**].

### 3.1 FlatZinc

Avant de discuter des similarités et différences entre les chaînes XCSP$^3$ et MiniZinc, focalisons nous tout d'abord sur FlatZinc en considérant à nouveau le problème 007 de CSPLib [**?**]. Un modèle (mzn), proposé par H. Kjellerstrand, est décrit par la figure 4. Ce modèle est équivalent à celui décrit en section 2.1.

Lorsqu'il est compilé (libminizinc-2.2.3), avec $n = 4$, on obtient le fichier (fzn) FlatZinc donné par la figure 5.

Il est assez évident que FlatZinc n'est pas conçu pour être manipulé directement (ni même lu) par un humain, mais plutôt traité par des outils et solveurs. On peut également remarquer qu'une partie de la structure est décomposée car des variables auxiliaires sont introduites et une analyse est nécessaire (avec éventuellement des annotations) pour reconstruire une expression telle que $\texttt{diffs}[k] = \texttt{abs}(x[k+1] - x[k])$ pour tout $k$. Par ailleurs, la compilation en MiniZinc est toujours ajustée à un solveur cible (ceci est discuté plus loin). Cela signifie qu'il n'est pas prévu qu'un fichier FlatZinc généré soit lu par des solveurs différents. Pour faire une comparaison entre deux solveurs, la plupart du temps, il est nécessaire de partir de deux fichiers : le fichier mzn et le fichier dzn. En comparaison, avec XCSP$^3$, un seul fichier est nécessaire, et même si cela peut paraître assez mineur au premier regard, c'est assez pratique (et moins sujet à une erreur de manipulation). Pour finir, les fichiers XCSP$^3$ sont généralement assez compacts. Par exemple, les fichiers XCSP$^3$ et FlatZinc générés pour All-Interval avec $n = 1000$ ont comme taille respective $36Kb$ et $422Kb$.

```
include "globals.mzn";

int: n;

array[1..n] of var 1..n: x;
array[1..n-1] of var 1..n-1: diffs;

constraint
all_different(diffs)
/\
all_different(x)
/\
forall(k in 1..n-1) (
  diffs[k] = abs(x[k+1] - x[k])
)
/\ % symmetry breaking
x[1] < x[n-1]
/\
diffs[1] < diffs[2]
;
```

FIGURE 4 – Le fichier MiniZincBacp.mzn

### 3.2 MiniZinc et Conjure

MiniZinc et Conjure [**?**] sont deux chaînes de compilation populaires, visant à ajuster les modèles aux exigences des solveurs cibles. En MiniZinc, lorsqu'on interface un solveur spécifique à FlatZinc, il est possible de spécifier comment les contraintes (globales) doivent être considérées (reformulées), de manière à générer un fichier FlatZinc adéquat au solveur. Il est également possible d'utiliser un outil (appelé globalizer) pour reformuler automatiquement certaines parties des modèles. Un autre atout de MiniZinc est l'intégration possible de solveurs MIP (Mixed Integer Programming).

Conjure est un outil PPC qui peut automatiquement lire des spécifications de problème abstraites données dans le langage Essence [**?**] et générer des modèles concrets dans le langage de modélisation Essence' [**?**]. Ensuite, l'assistant de modélisation Savile Row [**?**] est capable de traduire les modèles Essence' sous des formes appropriées aux solveurs cibles (y compris, SAT). Savile Row applique automatiquement un certain nombre de techniques sophistiquées, telles que la 'common sub-expression elimination' (CSE), qui remplace par des variables les expressions similaires trouvées dans les expressions.

Tandis que MiniZinc et Conjure se focalisent tous les deux à l'ajustement de compilation, le principal objectif de XCSP$^3$ est d'être capable de représenter des instances de problèmes sous une forme compacte, lisible et structurée, les rendant aussi proches que possible des modèles originaux. Lorsque la compilation est lancée depuis PyCSP$^3$, aucune reformulation ou

```
predicate all_different_int(array [int] of var int: x);
array [1..2] of int: X_INTRODUCED_7_ = [1,-1];
var 1..4: X_INTRODUCED_0_;
var 1..4: X_INTRODUCED_1_;
var 1..4: X_INTRODUCED_2_;
var 1..4: X_INTRODUCED_3_;
var -3..3: X_INTRODUCED_8_ ::var_is_introduced :: is_defined_var;
var 1..3: X_INTRODUCED_9_ ::var_is_introduced :: is_defined_var;
var -3..3: X_INTRODUCED_10_ ::var_is_introduced :: is_defined_var;
var 1..3: X_INTRODUCED_11_ ::var_is_introduced :: is_defined_var;
var -3..3: X_INTRODUCED_12_ ::var_is_introduced :: is_defined_var;
var 1..3: X_INTRODUCED_13_ ::var_is_introduced :: is_defined_var;
array [1..4] of var int: x:: output_array([1..4]) =
  [X_INTRODUCED_0_,X_INTRODUCED_1_,X_INTRODUCED_2_,X_INTRODUCED_3_];
array [1..3] of var int: diffs:: output_array([1..3]) = [X_INTRODUCED_9_,
    X_INTRODUCED_11_,X_INTRODUCED_13_];
constraint all_different_int(diffs);
constraint all_different_int(x);
constraint int_abs(X_INTRODUCED_8_,X_INTRODUCED_9_):: defines_var(
    X_INTRODUCED_9_);
constraint int_abs(X_INTRODUCED_10_,X_INTRODUCED_11_):: defines_var(
    X_INTRODUCED_11_);
constraint int_abs(X_INTRODUCED_12_,X_INTRODUCED_13_):: defines_var(
    X_INTRODUCED_13_);
constraint int_lin_le(X_INTRODUCED_7_,[X_INTRODUCED_0_,X_INTRODUCED_2_],-1);
constraint int_lin_le(X_INTRODUCED_7_,[X_INTRODUCED_9_,X_INTRODUCED_11_],-1);
constraint int_lin_eq([1,-1,-1],[X_INTRODUCED_1_,X_INTRODUCED_0_,X_INTRODUCED_8_
    ],0)
  :: defines_var(X_INTRODUCED_8_);
constraint int_lin_eq([1,-1,-1],[X_INTRODUCED_2_,X_INTRODUCED_1_,
    X_INTRODUCED_10_],0)
  :: defines_var(X_INTRODUCED_10_);
constraint int_lin_eq([1,-1,-1],[X_INTRODUCED_3_,X_INTRODUCED_2_,
    X_INTRODUCED_12_],0)
  :: defines_var(X_INTRODUCED_12_);
solve  satisfy;
```

FIGURE 5 – Le fichier FlatZincBacp.fzn

transformation n'est appliquées. Il s'agit d'une forme de compilation 'haute fidélité'. D'un côté, le principal avantage de la compilation ajustée est d'ajuster les modèles aux solveurs, tout en prêtant garde à l'efficacité. De l'autre, le principal avantage de la compilation 'haute fidélité' est de pouvoir tester la robustesse des solveurs (une sorte de défi IA), tout en gardant les choses simples du point de vue de l'utilisateur. Il est à noter qu'on pourrait envisager pour des tâches spécifiques de construire des outils indépendants permettant de reformuler des instances XCSP$^3$, avant ou au moment du parsing, mais ceci n'est pas l'une de nos priorités.

Pour conclure cette partie, nous tenons à souligner qu'il est facile d'éditer les fichiers XCSP$^3$ pour tester par exemple des modifications mineures. Par exemple, on peut facilement tester (lorsqu'on cherche à corriger un bug) si ajouter ou supprimer un tuple dans la liste des supports (ou conflits) d'une contrainte table modifie le comportement du solveur. Également, on peut facilement éditer l'automate définissant une contrainte `regular` car son expression en XCSP$^3$ est tout à fait naturelle et lisible. Une alternative est de modifier le modèle et de relancer la compilation : cela est vrai pour les approches ajustées et haute-fidélité. Cependant, l'utilisateur doit être très rigoureux, en s'assurant de récupérer le bon modèle original, le bon fichier de données, et avec les approches ajustées, éventuellement les bons fichiers de configuration (pour les contraintes globales) et les bonnes options de compilation. De plus, pour MiniZinc et Conjure, il est nécessaire de relancer la compilation pour chaque solveur devant être testé sur l'instance modifiée.

## 4 Conclusion

À l'heure d'écrire cet article, environ 80 problèmes ont été modélisés avec PyCSP³. Les modèles sont en règle générale (légèrement) plus compacts que ceux, équivalents, écrits en JvCSP³. Dans un futur proche, nous prévoyons quelques extensions à ce travail, comme par exemple, offrir à l'utilisateur la possibilité d'écrire des contraintes en intention sous forme naturelle (infixe) comme « $x = y + z$ ».

## Remerciements

Ce travail a obtenu le soutien du projet CPER DATA de la région "Hauts-de-France".

## Références

[1] O. Akgun, A. Frisch, I. Gent, B. Hussain, C. Jefferson, L. Kotthoff, I. Miguel et P. Nightingale : Automated symmetry breaking and model selection in Conjure. *In Proceedings of CP'13*, pages 107–116, 2013.

[2] C. Bessiere, P. Meseguer, E.C. Freuder et J. Larrosa : On Forward Checking for non-binary constraint satisfaction. *Artificial Intelligence*, 141: 205–224, 2002.

[3] F. Boussemart, C. Lecoutre et G. Audemard C. Piette : XCSP3 : an integrated format for benchmarking combinatorial constrained problems. *CoRR*, abs/1611.03398, 2016.

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

[5] E. Hebrard, E. O'Mahony et B. O'Sullivan : Constraint programming and combinatorial optimisation in Numberjack. *In Proceedings of CPAIOR'10*, pages 181–185, 2010.

[6] B. Hnich, Z. Kiziltan et T. Walsh : CSPLib Problem 030 : Balanced academic curriculum problem. http://www.csplib.org/Problems/prob030.

[7] H. Hoos : CSPLib Problem 007 : All-interval series. http://www.csplib.org/Problems/prob007.

[8] C. Jefferson, I. Miguel, B. Hnich, T. Walsh et I. Gent : CSPLib : A problem library for constraints. http://www.csplib.org, 1999.

[9] C. Lecoutre : MCSP3 : Easy modeling for everybody. Rapport technique, CRIL, 2018. https://github.com/xcsp3team/XCSP3-Java-Tools/tree/master/doc.

[10] N. Nethercote, P. Stuckey, R. Becket, S. Brand, G. Duck et G. Tack : MiniZinc : Towards a standard CP modelling language. *In Proceedings of CP'07*, pages 529–543, 2007.

[11] P. Nightingale, O. Akgün, I. Gent, C. Jefferson, I. Miguel et P. Spracklen : Automatically improving constraint models in Savile Row. *Artificial Intelligence*, 251:35–61, 2017.

[12] P. Nightingale et A. Rendl : Essence' description. Rapport technique arXiv :1601.02865, CoRR, 2016.

# Transformation de modèles et programmation par contraintes avec ATL$^{C}$

**Théo Le Calvar**[1] **Fabien Chhel**[2] **Frédéric Jouault**[2] **Frédéric Saubion**[1]
[1] LERIA, Université d'Angers, France
[2] Équipe ERIS, Groupe ESEO, Angers, France
prénom.nom@{univ-angers.fr,eseo.fr}

**Résumé**

La transformation de modèles est l'un des piliers de l'ingénierie dirigée par les modèles. De nombreux outils de transformation de modèles existent et ont permis de démontrer la pertinence de cette approche. Cependant, peu d'entre eux permettent d'exprimer des préférences entre différentes solutions possibles d'une transformation. Plus généralement, ces outils ne permettent pas d'explorer l'ensemble des solutions possibles afin de sélectionner une solution optimale.

Dans cet article, nous présentons ATL$^{C}$un nouvel outil permettant la spécification de transformations contenant des contraintes définissant ainsi un espace de modèles possibles en intention. Il est alors possible d'utiliser un (ou plusieurs) solveur(s) de contraintes pour explorer l'espace de modèles et sélectionner les meilleures solutions. De telles transformations ne sont actuellement pas prises en compte par les outils de transformation classiques. Nous développons un exemple combinant les solveurs Choco et Cassowary qui permet de générer la visualisation interactive d'un planning.

**Abstract**

Model transformation is a pillar of model driven engineering. Many model transformation tools exist and have shown the approach to be relevant.

However, few of them enable the user to specify a preference among all possible solutions of a transformation. More generally, these tools dot not let users explore model sets to find an optimal model.

In this paper we present ATL$^{C}$, a tool to write transformations containing constraints. These constraints define a model set that can be explored. With one (or multiple) solver(s) it is possible to explore and select an optimal model in the generated model set. It is thus possible to define model transformations that otherwise would not have been possible with classical tools. To illustrate our tool, we present a transformation that generates an interactive task schedule using two solvers.

## 1 Introduction

La transformation de modèles s'est imposée comme une méthode centrale de l'ingénierie dirigée par les modèles (IDM). En IDM, le modèle est considéré comme l'artéfact principal. De nombreuses techniques de transformation existent et offrent différentes capacités. Certaines sont incrémentales [5, 13], d'autres bidirectionnelles [3, 7].

Cependant, peu d'entre elles permettent d'intégrer des problématiques de recherche de solution au sein de la transformation. Dans [10] nous avons présenté une approche utilisant des contraintes pour générer des ensembles de modèles explorables. L'utilisateur peut modifier le modèle généré en respectant certaines contraintes vérifiées par un ou plusieurs solveurs, qui permettent de réparer interactivement ce modèle le cas échéant. Ces contraintes définissent un ensemble de modèles défini en intention, que l'utilisateur explore de manière guidée afin de trouver son modèle optimal.

Dans cet article nous présentons ATL$^{C}$(**ATL** with **C**onstraints) qui vient remplacer le prototype présenté dans [9, 10]. Ce langage est une extension d'ATL [6] (**A**TL **T**ransformation **L**anguage), un langage déclaratif de transformation de modèles basé sur le *pattern matching* de règles, auquel avons ajouté la possibilité de définir des contraintes qui s'appliquent aux éléments générés. Dans un premier temps, nous présenterons brièvement la notion d'exploration d'ensembles de modèles ainsi que le langage ATL dans la section 2. Puis, dans la section 3 nous présenterons les extensions d'ATL permettant l'ajout des contraintes. Nous discuterons ensuite de la gestion des contraintes dans la section 4. Un exemple de transformation sera présenté dans la section 5. Enfin, nous conclurons dans la section 6.

## 2 Contexte

### 2.1 Ensemble de modèles explorables

La notion d'ensemble de modèles explorables est basée sur celle d'espace de modèles [4]. Un espace de modèles est un graphe orienté $M = (M_\bullet, M_\Delta)$ avec $M_\bullet$ un ensemble de noeuds appelés modèles et $M_\Delta \subseteq M_\bullet \times M_\bullet$ un ensemble d'arcs appelés modifications ou deltas. Ce graphe est symétrique : pour tout $\delta \in M_\Delta$ (e.g., $(a, b)$) il existe un $\delta^{-1} \in M_\Delta$ (e.g., $(b, a)$). Un *ensemble de modèles explorable* [10] est un sous-ensemble $M_V \subseteq M_\bullet$. Ce sous ensemble regroupe les modèles dit *valides*. Il peut être défini par un ensemble de contraintes.

Lorsqu'un modèle $m \in M_V$ est modifié par l'application d'un changement $\delta = (m, m') \in M_\Delta$, le modèle $m' \in M_\bullet$ obtenu a deux états possibles. Ce nouveau modèle peut être soit valide ($m' \in M_V$), soit invalide ($m' \notin M_V$). S'il est valide, alors aucune intervention n'est nécessaire. S'il ne l'est pas, alors il faut appliquer une nouvelle modification $\delta' \in M_\Delta$ à $m'$ afin de retrouver un modèle valide. Il peut exister plusieurs modifications qui, appliquées à $m'$, permettent d'obtenir à nouveau un modèle valide. Par exemple, la plus simple consiste à annuler la modification $\delta$ en appliquant sa modification inverse $\delta^{-1}$.

### 2.2 Le langage ATL

ATL [6] est un langage de transformation de modèles. Il comporte une partie déclarative s'articulant autour de la notion de règle, ainsi qu'une partie impérative que nous ignorerons ici. Chaque règle définit une relation entre un ou plusieurs éléments sources et un ou plusieurs éléments cibles. Ces règles comportent un ensemble de *bindings*. Un *binding* est une correspondance entre une propriété d'un élément cible et une expression impliquant des propriétés de la source. Ici, nos éléments sources/cibles sont des objets, ainsi les propriétés correspondent à des attributs de ces classes. Par exemple, $tDuree \leftarrow s.fin - s.debut$ définit un bindings calculant la propritété `tDuree` d'un élément de sortie à partir des propriétés `fin` et `debut` d'un élément d'entrée $s$.

Lors de l'exécution de la transformation, les règles sont appliquées aux éléments sources afin de créer les différents éléments cibles. Les propriétés des éléments cibles sont calculées grâce aux *bindings* définis dans les règles. Pour assurer la bonne exécution des transformations les expressions utilisées dans les *bindings* ne peuvent pas accéder aux propriétés des éléments cibles dans leurs parties droites. De même, il n'est pas possible de modifier les éléments sources, qui sont accessibles en lecture seule.

Il existe une exception à ces règles : lors de transformation en mode *refining*, le même modèle peut être utilisé en source et en cible. L'exécution doit alors se faire de manière à ce que toutes les expressions soient évaluées sur le modèle original. Ceci peut être réalisé en utilisant comme cible une copie de la source. Alternativement, tous les changements à effectuer peuvent être mémorisés sans modifier la source dans une première phase, puis être appliqués sur le modèle source en place dans une deuxième phase.

## 3 Un extension d'ATL

Nous avons défini une approche [10, 9] permettant de générer des ensembles de modèles explorables fonctionnant grâce à un ensemble de contraintes s'appliquant à un modèle. Un modèle peut ainsi être exploré et réparé à l'aide d'un solveur de contraintes. Cependant, le langage de transformation présenté était embarqué dans un langage généraliste (Xtend). Il n'était donc pas satisfaisant car il ne masquait pas certains détails d'implémentation. C'est pourquoi nous avons décidé de le remplacer pour une version étendue d'ATL, connu dans le domaine de la transformation de modèles.

L'objectif est alors d'étendre ATL afin qu'il permette de définir le modèle de contraintes qui doit être généré en plus du modèle cible, ainsi que les structures de synchronisation nécessaires. Nous appelons ATL$^{\mathrm{C}}$ cette version étendue d'ATL. La Figure 1 présente la structure générale des éléments principaux d'une transformation ATL$^{\mathrm{C}}$.

Ainsi, il est possible d'écrire des contraintes dans un style proche de ce qui peut être fait avec des langages tels que MiniZinc [11] ou MCPS3 [2]. Cependant, par soucis de facilité d'intégration, nous avons décidé d'utiliser une syntaxe compatible avec celle d'ATL.

Lors de la compilation d'une transformation ATL par notre compilateur, celui-ci délègue la compilation des *bindings* impliquant des contraintes à une extension spécialisée qui se charge de générer le problème de satisfaction de contraintes (CSP) équivalent.

## 4 Gestion des contraintes

Une fois le CSP généré par la transformation il est nécessaire de lui appliquer une série de transformations pour les adapter aux solveurs ciblés. Ceci se fait d'abord par la répartition des différentes contraintes vers le solveur ciblé (*constraint dispatcher* dans la Figure 1). Une fois les contraintes spécifiques à un solveur isolées, elles sont envoyées à une couche d'abstraction se chargeant de la transformation vers des contraintes spécifiques au solveur (*solver wrapper* dans la Figure 1). Le tri des

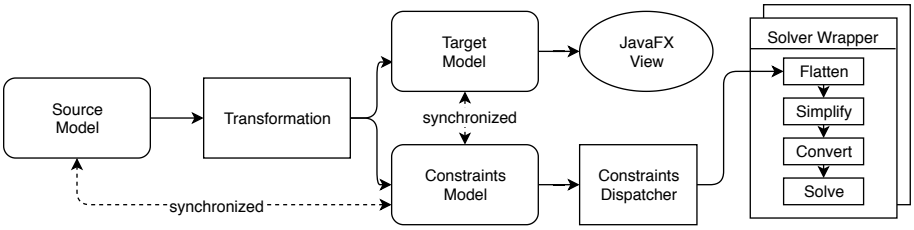



FIGURE 1 – Exécution d'une transformation ATL$^{C}$

contraintes se fait directement dans la transformation grace à une propriété `solver`.

Les *wrappers* autour des solveurs traduisent les contraintes de notre représentation vers celles utilisées par les solveurs. De plus, ils appliquent plusieurs transformations permettant d'adapter les contraintes au solveur. Par exemple, dans notre représentation il est possible d'écrire des contraintes dans lesquelles apparaissent des listes de variables. Tous les solveurs ne supportent pas ce type de contraintes. Il faut alors réécrire les contraintes contenant ces listes en plusieurs contraintes ne contenant plus que des variables classiques.

Il est aussi possible de pallier les limitations de certains solveurs grâce à cette étape de transformation. Par exemple, il est possible d'ajouter la notion de relation utilisée dans les modèles. Pour cela il faut encoder la relation par de nouvelles contraintes adaptées au solveur.

Actuellement, nous intégrons Choco [12], Cassowary [1], MiniCP [8] et XCSP3 [2]. Le niveau de support de chaque sortie varie grandement, par exemple, MiniCP ne supporte qu'une partie de ce que le *wrapper* Choco supporte.

## 5 Exemple de transformation

L'exemple de transformation présenté ici permet la visualisation sous forme de diagramme d'un modèle représentant un `Project` composé de `Tasks` qui doivent être assignées à des `Periods`. Le listing 1 présente une règle transformant une `Task` en un `Rectangle`, un `Text` et un `ConstraintGroup` contenant des contraintes. Nous ne présentons ici qu'une petite partie de la transformation. Une version complète est disponible sur GitHub `https://github.com/TheoLeCalvar/scheduling-example`. La version complète comporte une seconde transformation en mode refining définissant un problème d'affectation de `Tasks` à des `Periods`.

La première partie de la règle (l.5-11) est de l'ATL classique. La déclaration des contraintes se fait ensuite. Avec notre approche il est possible d'utiliser différents solveurs selon le type de contraintes utilisées. Ici nous utilisons Cassowary [1] (l.13).

On peut remarquer l'utilisation de différentes opérations particulières. Premièrement, comme en ATL, il est possible de naviguer le modèle source par l'utilisation de la notation pointée (`.`). Ensuite (l.15 et 16), `toConstant()`, permet de spécifier qu'une propriété doit être considérée comme une constante par le solveur. La contrainte `stay` (l.17) permet de minimiser les changements de valeur d'une variable entre deux résolutions successives du problème. On remarque aussi l'opération `collectTo()` qui permet de faire référence aux éléments qui seront générés par l'application d'une règle de transformation à un élément source. Ainsi, il est possible de placer le `Rectangle` correspondant à une `Task` à l'intérieur du `Rectangle` correspondant à une `Period` (l.20-25).

Lors de l'exécution de cette transformation, la première partie de la règle génère deux éléments de sortie, $r$ et $t \in M_{\bullet}$, dont les positions ne sont pas spécifiées. Sans contraintes sur les positions, il est très probable que $r$ et $t \notin M_V$. Les contraintes sont utilisées par un solveur pour réparer $r$ et $t$ en $r'$ et $t'$ de façon à ce que $r'$ et $t' \in M_V$.

## 6 Discussion & conclusion

Dans cet article, nous avons présenté une extension d'un langage de transformation de modèle permettant de définir relativement facilement des ensembles de modèles explorables. Ces derniers permettent d'introduire des techniques de recherche à la transformation de modèles.

L'utilisation d'un modèle intermédiaire de contraintes nous a permis de pouvoir considérer plusieurs solveurs simultanément. Chaque solveur est responsable de la résolution d'une partie des contraintes appliquées au modèle. Cependant, l'interaction entre les solveurs est pour l'instant limitée à des cas relativement simples. Les propriétés peuvent être lues par tous les solveurs, cependant, l'écriture est

```
1  unique lazy rule Task {
2  from
3   c: Scheduling!Task
4  to
5   r : JFX!Rectangle (
6    movable <- true
7   ),
8   t : JFX!Text (
9    text <- c.code, textOrigin <- #TOP,
10   mouseTransparent <- true
11  ),
12  constraints: Constraints!ConstraintGroup (
13   solver <- 'cassowary',
14   constraints <- Sequence {
15    r.width=1.2*t.width.toConstant()
16   ,r.height=1.2*t.height.toConstant()
17   ,r.x.stay('MEDIUM'),r.y.stay('MEDIUM')
18   ,t.x = r.x + 5,t.y = r.y + 2
19   ,r.x >= 0
20   ,r.y>=c.period->collectTo('Period').l.startY
21  ,r.x>=c.period->collectTo('Period').r.x--strong
22  ,r.x+r.width<=c.period->collectTo('Period').r.x
23   +c.period->collectTo('Period').r.width--strong
24 ,r.y+r.height<=c.period->collectTo('Period').r.y
25   +c.period->collectTo('Period').r.height
26 })}
```

Listing 1: Exemple de règle de transformation en ATL$^C$

limité à un seul solveur. De plus, c'est à l'utilisateur de s'assurer qu'il n'y ait pas de cycles de dépendances entre les variables.

Notre approche nécessite de pouvoir soumettre des modifications à une solution calculée par un solveur pour permettre l'exploration de l'ensemble de modèles. Ce genre de problème est particulièrement bien géré par Cassowary, mais ce n'est pas le cas de tous les solveurs que nous avons utilisés. Par exemple, Choco intègre les contraintes sur les entiers mais n'est pas particulièrement adapté à la résolution incrémentale de problèmes.

En outre, ATL$^C$ permet de définir des transformations nécessitant l'exploration d'un espace de recherche dans un langage (ATL) et un écosystème (Eclipse Modeling Framework) existants et couramment utilisés.

## Remerciements

Ces travaux ont été partiellement financés par Angers Loire Métropole et le RFI Atlanstic 2020. Nous remercions Dániel Varró pour ses idées de cas d'études.

## Références

[1] Greg J Badros, Alan Borning et Peter J Stuckey : The Cassowary linear arithmetic constraint solving algorithm. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 8(4):267–306, 2001.

[2] Frédéric Boussemart, Christophe Lecoutre et Cédric Piette : XCSP3 : an integrated format for benchmarking combinatorial constrained problems. *CoRR*, abs/1611.03398, 2016.

[3] Antonio Cicchetti, Davide Di Ruscio, Romina Eramo et Alfonso Pierantonio : JTL : A bidirectional and change propagating transformation language. *In* Brian Malloy, Steffen Staab et Mark van den Brand, éditeurs : *Software Language Engineering*, pages 183–202, 2011.

[4] Zinovy Diskin, Arif Wider, Hamid Gholizadeh et Krzysztof Czarnecki : Towards a rational taxonomy for increasingly symmetric model synchronization. *In ICMT 2014*, pages 57–73, 2014.

[5] Frédéric Jouault et Olivier Beaudoux : Efficient OCL-based Incremental Transformations. *In 16th International Workshop in OCL and Textual Modeling*, pages 121–136, 2016.

[6] Frédéric Jouault et Ivan Kurtev : Transforming models with ATL. *In* Jean-Michel Bruel, éditeur : *Satellite Events at the MoDELS 2005 Conference*, pages 128–138, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[7] Mathias Kleiner, Marcos Didonet Del Fabro et Patrick Albert : Model search : Formalizing and automating constraint solving in MDE platforms. *In ECMFA 2010*, pages 173–188, 2010.

[8] Laurent Michel, Pierre Schaus, Pascal Van Hentenryck : MiniCP : A lightweight solver for constraint programming, 2018. Available from `https://minicp.bitbucket.io`.

[9] Théo Le Calvar, Fabien Chhel, Frédéric Jouault et Frédéric Saubion : Transformation de Modèles et Contraintes pour l'Ingénierie Dirigée par les Modèles. *In JFPC2018*, Amiens, France, juin 2018.

[10] Théo Le Calvar, Fabien Chhel, Frédéric Jouault et Frédéric Saubion : Toward a Declarative Language to Generate Explorable Sets of Models. *In 34th ACM/SIGAPP Symposium on Applied Computing (SAC '19)*, pages 1837–1844, Limassol, Cyprus, avril 2019.

[11] Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck et Guido Tack : MiniZinc : Towards a standard CP modelling language. *In CP 2007*, pages 529–543, 2007.

[12] Charles Prud'homme *et al.* : *Choco Documentation.* TASC - LS2N CNRS UMR 6241, COSLING S.A.S., 2017.

[13] Dániel Varró *et al.* : Road to a reactive and incremental model transformation platform : three generations of the VIATRA framework. *Software & Systems Modeling*, 15(3):609–629, Jul 2016.

# Compact-Diagram
# Propagateur efficace pour la contrainte (s)MDD

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

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

**Résumé**

Les diagrammes de décision multi-valués (MDD) représentent une structure utile pour la modélisation de problèmes combinatoires sous contraintes. Dans notre article [3] intitulé "Compact-MDD : Efficiently Filtering (s)MDD Constraints with Reversible Sparse Bit-Set" et publié à IJCAI-18, nous proposons une structure graphique proche, appelée sMDD, où la couche centrale peut s'avérer non-déterministe. Nous montrons tout d'abord qu'il est simple et efficace de transformer toute table (ensemble de tuples) en sMDD. Nous introduisons ensuite un nouvel algorithme de filtrage, appelé Compact-Diagram (CD), exploitant des vecteurs de bits et pouvant être appliqué aux contraintes sMDD. Nos expérimentations montrent l'intérêt pratique de notre approche.

## 1 Introduction

Une contrainte table, également nommée contrainte en extension, exprime explicitement soit les combinaisons de valeurs acceptées (*supports*), soit les combinaisons de valeurs rejetées (*conflits*) par les variables impliquées dans la contrainte. En théorie, toute contrainte peut se (re-)formuler sous forme de contrainte table. C'est l'une des raisons pour lesquelles ce type de contrainte est particulièrement prisé en programmation par contraintes.

Une contrainte MDD repose sur une structure différente qui est décrite ci-dessous. Un MDD (Multi-Valued Decision Diagram) est un graphe orienté acyclique. Lorsqu'une telle structure est associée à une contrainte, elle est organisée en $r+1$ couches de noeuds et $r$ couches d'arêtes, où $r$ représente le nombre de variables de la contrainte. Chaque couche d'arêtes est associée à une variable, chacune d'entre elles étant étiquetée par une valeur pouvant potentiellement être affectée à la variable. Chaque arête lie un noeud d'une couche $i$ à un noeud de la couche $i+1$. Au niveau des noeuds, la première couche ne contient que la *racine* et la dernière couche ne contient que le noeud *terminal*. Chaque chemin allant de la racine au noeud terminal correspond donc à une instanciation possible des variables, ce qui est équivalent à un tuple dans une table.

Beaucoup d'efforts ont été consentis pour développer des algorithmes de filtrage dédiés aux contraintes tables et aux contraintes MDD. Actuellement, les algorithmes les plus efficaces sont respectivement Compact-Table (CT) [1] et MDD4R [2]. Dans cet article, nous proposons un nouveau propagateur pour les contraintes MDD, appelé Compact-Diagram (CD) et empruntant certains mécanismes à CT. Nous introduisons également une structure légèrement plus générale que MDD : le semi-MDD (sMDD), structure sur laquelle CD peut être appliqué.

## 2 Les semi-MDD (sMDD)

Un semi-MDD (ou sMDD) est, tout comme un MDD, un graphe orienté acyclique organisé en couches. Contrairement aux MDD, un semi-MDD n'est pas uniquement composé de noeuds déterministes (dits de décision). Un noeud est dit déterministe en entrée (respectivement, déterministe en sortie) s'il y a au niveau du noeud au plus une arête entrante (respectivement, sortante) étiquetée par une même valeur. S'il y a au moins deux arêtes entrantes (respectivement, sortantes) avec la même étiquette (valeur), alors le noeud est dit non-déterministe en entrée (respectivement, non-déterministe en sortie). Les noeuds d'un MDD sont par définition tous déterministes en sortie.

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

Pour un semi-MDD d'arité $r$ (et donc pourvu de $r+1$ couches de noeuds), les $\lfloor \frac{r}{2} \rfloor$ premiers niveaux sont déterministes en sortie, les deux niveaux suivants sont non-déterministes en entrée et sortie et pour finir, les $\lfloor \frac{r-1}{2} \rfloor$ derniers niveaux sont déterministes en entrée. Nous proposons une adaptation de *pReduce*, l'algorithme pour générer un MDD à partir d'une table. Cet algorithme que nous appelons *sReduce* permet de transformer toute table en sMDD.

## 3 Le propagateur Compact-Diagram

CD (Compact-Diagram) utilise, tout comme CT (Compact-Table), des vecteurs de bits pour améliorer les performances. Un vecteur de bits (basé sur un sparse set) réversible est associé à chaque couche du sMDD. Chacun des bits est associé à l'une des arêtes de la couche et permet de maintenir l'état de l'arête (à savoir si elle est encore valide et/ou accessible). Des vecteurs de bits pré-calculés supplémentaires regroupent les arêtes étiquetées de manière identique, ou encore les arêtes avec une même source ou une même destination.

Le filtrage se déroule en deux étapes. La première étape, de mise à jour, permet d'éliminer les arêtes qui ne sont plus accessibles. Sont éliminées tout d'abord les arêtes qui ont comme étiquette les valeurs supprimées des domaines depuis le dernier appel au propagateur. Cette première opération s'effectue de manière similaire à l'étape de mise à jour de CT. Ensuite, une propagation du retrait de ces arêtes doit être faite afin de conserver l'intégrité du sMDD (toute arête restante doit appartenir à un chemin allant de la racine au noeud terminal). Pour cette seconde opération, tout comme dans MDD4R, un double parcours (de haut en bas puis de bas en haut) de la structure est nécessaire.

La deuxième étape, le filtrage à proprement parler, se déroule comme pour CT, en effectuant l'intersection du vecteur de bits identifiant les arêtes encore valides et le vecteur de bits `supports` de la valeur testée. Si cette intersection est vide, la valeur peut être éliminée du domaine.

## 4 Résultats

Pour commencer, nous avons comparé les MDD et les sMDD générés par *sReduce* (une option permet de choisir si un 'simple' MDD doit être généré). Nous nous intéressons au nombre de noeuds et au nombre d'arètes. Nous avons pu observer que pour 70% des instances, les MDD générés comportent au moins 7 fois plus de noeuds. De plus, il n'existe aucune instance dans notre benchmark où le nombre de noeuds s'est avéré plus petit dans les MDD générés. Concernant les arêtes, généralement les MDD générés en comportent moins que les sMDD. Toutefois, pour 85% des sMDD le nombre d'arètes est au maximum 1,5 fois celui des MDD. Pour les 15% restants, le nombre d'arètes est au plus 4 fois plus important.

Ensuite, nous avons comparé CD à MDD4R. MDD4R appliqué aux contraintes MDD est comparé à CD appliqué aux contraintes MDD et sMDD. De manière générale, CD est plus performant que MDD4R. De plus, le choix de la structure sMDD (plutôt que MDD) est typiquement un avantage. En partie, cela est dû au nombre réduit de noeuds.

CD a également été comparé à CT. CT reste globalement plus performant, malgré une réduction de l'écart de performance entre les deux types de représentations (tables versus diagrammes).

## 5 Conclusion

Nous avons proposé une nouvelle structure graphique, appelée sMDD, combinant deux demi-MDD ainsi qu'un algorithme permettant de générer de telles structures à partir de tables. Nos résultats montrent qu'un MDD a généralement plus de noeuds et moins d'arêtes qu'un sMDD correspondant. Nous avons également proposé un nouveau propagateur générique pour les contraintes sMDD. Nos résultats montrent l'obtention d'un gain de performance par rapport à MDD4R. Pour plus de détails concernant les structures algorithmes et résultats, n'hésitez pas à vous reporter à l'article original.

## Remerciements

Le second auteur est financé par le projet CPER Data des Hauts-de-France.

## Références

[1] Jordan Demeulenaere, Renaud Hartert, Christophe Lecoutre, Guillaume Perez, Laurent Perron, Jean-Charles Régin et Pierre Schaus : Compact-Table : efficiently filtering table constraints with reversible sparse bit-sets. *In Proceedings of CP'16*, pages 207–223, 2016.

[2] Guillaume Perez et Jean-Charles Régin : Improving GAC-4 for Table and MDD constraints. *In Proceedings of CP'14*, pages 606–621, 2014.

[3] Hélène Verhaeghe, Christophe Lecoutre et Pierre Schaus : Compact-MDD : Efficiently filtering (s)MDD constraints with reversible sparse bit-sets. *In IJCAI18*, pages 1383–1389, 2018.

# Extension de Compact-Diagram aux smart MVD

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

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

**Résumé**

Les diagrammes de décision multi-valués (MDD), et plus généralement les diagrammes variables multi-valués (MVD), sont des structures qui peuvent s'avérer utiles pour la modélisation de problèmes combinatoires sous contraintes. Ces dernières années, de nombreux algorithmes de filtrage ont été proposés tels que mddc, MDD4R et Compact-Diagram. Par ailleurs, diverses formes compressées de tables ont également été proposées au fil du temps menant notamment à une hybridation 'smart' entre les représentations des contraintes dite en extension et en intension. Cela peut se résumer à intégrer des contraintes arithmétiques simples dans les tables définissant la sémantique des contraintes. L'algorithme état de l'art Compact-Table a été récemment étendu pour gérer ces tables 'smart simple', i.e. des tables contenant des éléments de la forme '$*$', '$\neq v$', '$\leq v$', '$\geq v$' et '$\in S$'. Dans notre article, intitulé "Extending Compact-Diagram to Basic Smart Multi-Valued Variable Diagrams"[2] et publié à CPAIOR-19, nous introduisons le concept de MVD 'smart simple' (*bs*-MVD) correspondant à la possibilité d'utiliser ces formes de contraintes arithmétiques au niveau des étiquettes des diagrammes. Nous montrons comment étendre l'algorithme Compact-Diagram pour gérer ce type de diagrammes.

## 1 Introduction

La représentation efficace de contraintes sous forme de tables ou diagrammes de décision est l'un des sujets de recherche importants de la dernière décennie dans le contexte de la programmation par contraintes. Ces améliorations peuvent être classées en deux grandes catégories : i) l'amélioration des algorithmes de filtrage (introduction de la technique de réduction tabulaire, exploitation des opérations bit à bit, ...) et ii) l'amélioration de la représentation (tables compressées, formes compactes de graphes, ...).

Compact-Diagram (appelé précédemment Compact-MDD) est un algorithme de filtrage introduit récemment [1]. Il exploite notamment les opérations bit à bit adaptées à toute contrainte définie par un diagramme variable multi-valué (MVD), qui est une généralisation du diagramme de décision multi-valué autorisant le non-déterminisme. Dans cet article, nous proposons une extension de l'algorithme Compact-Diagram (CD) permettant de gérer des diagrammes 'smart' (simples), i.e. des diagrammes tels que les arêtes peuvent avoir des étiquettes de la forme '$= v$', '$*$', '$\neq v$', '$\leq v$', '$\geq v$' et '$\in S$'. Nous décrivons également comment il est possible de générer ces diagrammes à partir de tables.

Au delà de la compression potentielle obtenue avec ces nouveaux diagrammes, une application directe est la représentation de la contrainte regular de manière compacte.

## 2 Transformation de Tables en *bs*-MVD

La transformation peut être effectuée de deux manières. La première possibilité est de transformer une table en diagramme en utilisant l'un des algorithmes déjà connu tel que pReduce (pour générer un MDD) ou sReduce (pour générer un semi-MDD). Ensuite, le diagramme est réduit en regroupant certaines arêtes. Cette opération consiste à examiner tous les groupes d'arêtes partageant le même nœud source et le même nœud destination. Les étiquettes présentes dans le groupe sont alors examinées et comparées au domaine de la variable associée. Des expressions unaires peuvent alors être dérivées et les arêtes concernées sont remplacées par de nouvelles arêtes qui sont étiquetées avec ces expressions. Par exemple, si toutes les valeurs du domaine sont représentées dans le groupe, cela correspond à '$*$', si toutes les valeurs inférieures à $v$ sont représentées, cela correspond à '$\leq v$',....

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

La deuxième possibilité est de commencer par transformer la table en une table smart (simple). Ensuite, celle-ci est transformée en diagramme par l'adaptation des algorithmes connus pour gérer les nouvelles catégories d'étiquettes. Pour pReduce et sReduce, cette adaptation ne requiert que la définition d'un ordre total sur toutes les étiquettes possibles.

## 3 CD$^{bs}$ : CD appliqué aux *bs*-MVD

CD et CT sont relativement similaires en terme de conception. Les deux algorithmes utilisent des vecteurs de bits pré-calculés appelés `supports` pour identifier les arêtes et tuples qui doivent être éliminés durant la phase de mise à jour (suite à la suppression de valeurs dans les domaines des variables associées). Pour gérer les contraintes unaires, CD peut être modifié selon la logique suivie par CT. Tout comme pour CT$^{bs}$, trois vecteurs de bits pré-calculés sont introduits : `supports`$^*$, `supportsMin` et `supportsMax`. Des modifications s'opèrent également comme pour CT pendant la phase de mise à jour.

Contrairement aux tables pour lesquelles un bit est associé à un tuple et donc à plusieurs variables simultanément, ici, chaque bit est associé à une arête et donc à une seule variable. L'ordre des bits d'un niveau peut donc être facilement changé sans impacter les autres niveaux du diagramme. Une partition des arêtes est donc possible suivant leur type d'étiquette. Les ensembles de bits, implantés par des tableaux de "mots" (i.e. groupements de 64 bits quand on utilise en Java un entier de type Long), peuvent donc être construits selon cette partition (tout les bits d'un même mot sont associés à des arêtes d'une même catégorie). Cela permet à CD$^{bs}$ d'appliquer un nombre restreint d'opérations pour chacun des mots, suivant la catégorie de celui-ci contrairement à CT$^{bs}$ qui ne pouvait pas faire d'hypothèses sur le contenu des mots.

## 4 Résultats

Les résultats portant sur la transformation de tables en *bs*-MVD nous montre tout d'abord que la méthode transitant par les diagrammes génère moins de nœuds et d'arcs que la méthode transitant par les *bs*-tables. Nous expliquons cela par une probabilité plus faible d'obtenir un regroupement de nœuds lors du passage par la *bs*-table dû à l'augmentation du nombre d'étiquettes possibles amenée par l'introduction des étiquettes 's-mart' présentes dans la *bs*-table.

Les résultats concernant l'algorithme de filtrage montrent une amélioration en temps lorsque le diagramme compressé est plus petit en taille que le diagramme non compressé correspondant. Cela est le cas pour les instances générées par transformation de tables en *bs*-MDD en transitant préalablement par la transformation en diagrammes.

En comparaison avec CT$^{bs}$, CD$^{bs}$ reste toutefois moins performant. Néanmoins l'écart est de plus en plus réduit entre les deux approches (tables versus diagrammes) et un plus grand nombre d'instances est résolu (de manière plus rapide) par CD$^{bs}$ que CD. Ces instances sont sans grande surprise celles où la compression en diagrammes est la plus importante. Il est également connu que certains types de problèmes (tels que ceux impliquant la contrainte globale AllDifferent) ne sont pas du tout appropriés à une représentation sous forme de diagrammes dû une possibilité de compression très limitée (lors du processus de regroupement de nœuds). Il est de ce fait, et selon notre opinion, utile d'avoir les deux formes de contraintes, et leurs propagateurs CD et CT, pour pouvoir sélectionner le plus pertinent en fonction de la capacité de compression en diagrammes.

## 5 Conclusion

Dans cet article, nous avons proposé une extension des diagrammes classiques. Nous avons montré comment générer ces *bs*-MVD à partir de tables, de tables 'smart simple' et de diagrammes. Nous avons présenté un algorithme de filtrage pour cette nouvelle forme de diagramme. Cet algorithme est efficace et permet de réduire encore un peu plus l'écart entre les méthodes basées sur les tables et les méthodes basées sur les diagrammes. Pour plus de détails concernant les structures, algorithmes et résultats, veuillez consulter l'article original [2].

## Remerciements

Le second auteur est financé par le projet CPER Data des Hauts-de-France.

## Références

[1] Hélène Verhaeghe, Christophe Lecoutre et Pierre Schaus : Compact-MDD : Efficiently filtering (s)MDD constraints with reversible sparse bit-sets. *In IJCAI18*, pages 1383–1389, 2018.

[2] Hélène Verhaeghe, Christophe Lecoutre et Pierre Schaus : Extending compact-diagram to basic smart multi-valued variable diagrams. *In Proceedings of CPAIOR'17*, 2019.

# Gestion robuste des opérations sur Mars

**Michael Saint-Guillain***

Université catholique de Louvain, Belgique
Institut National des Sciences Appliquées de Lyon, France
michael.saint@uclouvain.be

**Résumé**

Résumé d'un article accepté à la conférence ICAPS'19 : *Robust Operations Management on Mars*. Nous comparons les approches déterministes et robustes au problème d'ordonnancer un ensemble de tâches scientifiques, dont les durées sont incertaines. En prenant en compte les contraintes temporelles du problème, nous développons les formules fermées permettant d'évaluer, de manière exacte, la probabilité qu'a une solution de rester valide. Nos expériences, qui prennent en compte l'incertitude sur les connaissances stochastiques du problème elles-mêmes, sont effectuées sur des instances réelles provenant des projets scientifiques, contraintes et objectifs poursuivis lors d'une mission de simulation de séjour habité sur Mars. Les résultats montrent que, même lorsque les distributions de probabilités fournies sont de très mauvaise qualité, les solutions obtenues grâce à notre modèle stochastique surpassent largement celles obtenues avec un modèle déterministe classique, tout en préservant la quasi totalité de la qualité des projets scientifiques.

## 1 Introduction

Contrairement à la plupart des problèmes classiques d'ordonnancement de tâches, les opérations d'une mission spatiale doivent être planifiées plusieurs jours à l'avance. La complexité des chaînes de décisions ainsi que les délais de communication ne permettent pas de replanification de dernière minute. Même dans sa version déterministe, l'ordonnancement et la distribution des tâches, dans un environnement de travail aussi contraint que la *Mars Research Desert Station* (MDRS, Fig. 1), n'est pas un problème trivial.

À la MDRS, calculer un planning optimal devient nettement moins intéressant alors que les données du problème, comme la durée de manipulation des tâches, se révèlent différentes des valeurs attendues. Dans un environnement contraint où les ressources sont partagées, de telles déviations se propagent aux tâches suivantes, débouchant à une infaisabilité globale.

Figure 1 – La Mars Desert Research Station (Mars Society) dans le désert de l'Utah, est un complexe de simulation de missions habitées sur Mars.

L'objectif de notre étude est de déterminer l'impacte, sur la fiabilité de la planification d'une mission, d'une modélisation robuste stochastique du problème au lieu de celle classique, déterministe. Les résultats expérimentaux sont obtenus grâce à l'étude de cas réels d'une simulation de mission habitée sur Mars.

## 2 Insuffisance des modèles déterministes

Considérons le problème, très simple, suivant. Quatre tâches $\{A, B, C, D\}$, assignées à un seul opérateur, doivent être ordonnancées. Chacune peut avoir une ou plusieurs fenêtres temporelles, des contraintes de précédence ainsi que des temps minimum de transitions, ces dernières indiquant un délai minimum entre la fin d'une tâche et le début d'une autre. Les tâches ne pouvent être étalées sur plusieurs fenêtres. L'horizon opérationnel est constitué de deux journées de cinq heures, allant de 9 :00 à 14 :00. La Fig. 2 complète l'explication de notre problème.

*Papier doctorant : Michael Saint-Guillain est auteur principal.

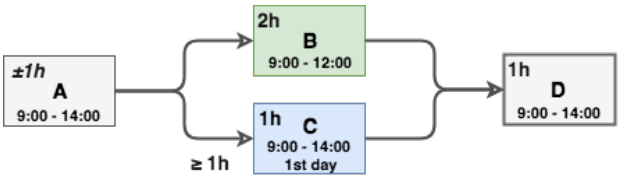



FIGURE 2 – Chaque tâche a une durée de 1 ou 2h, et peut être effectuée soit entre 9h à 14h, soit uniquement le matin (9h à 12h). Un délai d'une heure doit être respecté entre le début de C, laquelle doit être terminée durant le premier jour, et la fin de A.

Un objectif courant est de minimiser la durée totale du projet, ce que fait la solution $\langle A, B, C, D\rangle$. Il s'agit de l'unique solution optimale, permettant de boucler le projet en une journée : A commence à 9h, B à 10h, C à 12h et D à 13h. Supposons maintenant qu'au lieu de durer exactement une heure, notre tâche A nécessite un peu plus de temps. Dans ce cas, B ne peut être terminée pour 12h, et doit être reprogrammée au jour suivant. La solution n'est alors plus faisable, car la tâche C n'est pas effectuée au jour 1. Même si la tâche A requiert en moyenne une heure, la probabilité que celle-ci dépasse l'heure (et donc que le projet se solde par un échec !) n'est absolument pas négligeable, et ce quel que soit son degré d'incertitude (l'écart-type de sa distribution). Une meilleure planification, beaucoup plus fiable, est de suivre la solution $\langle A, C, B, D\rangle$.

## 3 Étude de cas et résultats

Durant notre séjour à la MDRS, nous avons mené 10 projets scientifiques (voir `ucltomars.org/#!/crews/190/projects`). Chaque projet, composé d'un ensemble de tâches sous contraintes, est associée à un membre de l'équipage, constitué de 8 chercheurs. Un exemple de projet est donné en Fig. 3.

**Modèle déterministe.** Durant la mission, le problème rencontré sur place a été modélisé en tant que *job-shop scheduling problem* déterministe, en considérant connues avec certitude les durées de manipulation des tâches. L'horizon du problème comprend initialement 13 journées de 9 heures. À la fin de chaque journée, le problème est mis à jour en fonction des opérations menées le jour même. Un nouveau problème est donc résolu, produisant un planning pour les jours restants.

Étant donné que l'ensemble des ressources (matériel, salles, opérateurs, ...) de la MDRS sont partagées entre l'ensemble des projets de recherche, ces derniers ne sont clairement pas indépendants, et forment un problème global. En tout, ce problème est initialement

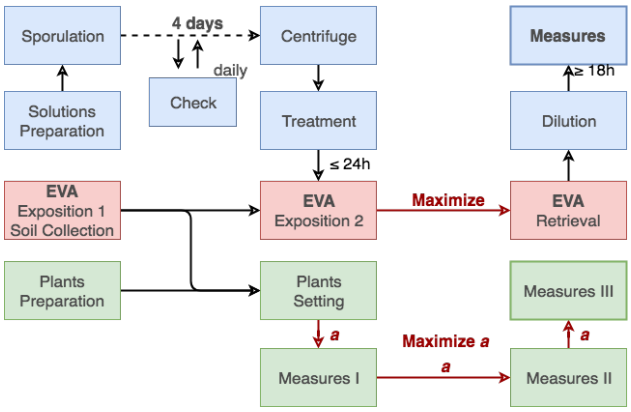



FIGURE 3 – Modélisation d'un projet de recherche, combinant botanique (vert) et biologie (bleu), mené par Frédérique Peyrusson (UCLouvain, Belgique).

composé de 237 tâches, sur un horizon de 13 jours de 144 unités de temps, chacune de 10 minutes.

**Modèle robuste stochastique.** Grâce aux données du problème recueillies durant la mission, nous avons effectué une étude a posteriori de l'impacte de l'incertitude, selon que celle-ci soit prise en compte ou non, au moment d'optimiser les plannings d'opération.

Dans notre modèle robuste, l'objectif à optimiser est remplacé par l'heuristique

$$f(s) = f^{\text{mdrs}}(s) \ \times \ r^{\text{mdrs}}(s)$$

où $f^{\text{mdrs}}(s)$ désigne la qualité d'une solution $s$ telle que calculée dans un contexte déterministe (la Fig. 3 montre par ailleurs des exemples d'objectifs à optimiser), et $r^{\text{mdrs}}(s)$ désigne la probabilité qu'a la solution $s$ de rester valide, c'est-à-dire sa *robustesse*.

Le calcul de $r^{\text{mdrs}}(s)$, pour $n$ tâches avec un horizon discrétisé en $h$ unités de temps, a une complexité temporelle pseudo-polynomiale en $\mathcal{O}(nh^2 + n^2h^3)$. Dans notre étude, nous identifions également certaines limites théoriques de cette complexité.

La probabilité $r^{\text{mdrs}}(s)$ dépend des distributions de probabilités sur les durées des tâches, ne pouvant être que des estimations. Au moment de comparer les solutions obtenues, sur base du modèle déterministe ou stochastique, nos simulations sont effectuées en considérant les vraies distributions comme étant cachées.

**Résultats.** Les résultats obtenus sur base de notre étude de cas sont très encourageants. Même dans un contexte où les durées des tâches sont globalement toutes surestimées, la fiabilité des solutions peut être multipliée en moyenne par trois. De plus, cette robustesse accrue ne vient qu'à un prix dérisoire d'environ 7%, en termes de différence de qualité $f^{\text{mdrs}}(s)$.

# Testé comme jamais...

**Xavier Gillard*** **Pierre Schaus**
Université Catholique de Louvain, Louvain-la-Neuve, Belgium
{xavier.gillard, pierre.schaus}@uclouvain.be

**Résumé**

Cet article présente SolverCheck, une bibliothèque libre de 'test par propriétés' (property-based testing, PBT) qui a été spécifiquement conçue pour tester les solveurs CP. En particulier, SolverCheck offre un langage déclaratif qui permet de décrire le comportement attendu d'un propagateur et de le tester automatiquement. Ces tests permettent non seulement de s'assurer que le propagateur ne supprime aucune des solutions d'un CSP mais aussi de s'assurer du niveau de consistance imposé par celui-ci (en ce compris, des consistances hybrides, personnalisables). Les expériences réalisées sur AbsCon [24], Choco [29], JaCoP [22] et MiniCP [23] ont permis la mise en évidence de bugs non triviaux malgré le grand soin apporté à la confection des suites de tests de ces solveurs.

**Abstract**

This paper introduces SolverCheck, an open source property-based testing library specifically designed to assess the quality of CP solvers. In particular, SolverCheck provides a declarative language to express a propagator's expected behavior and test it automatically. These tests not only allow checking that the propagator never wrongly removes any solution from a CSP but also to check the consistency level enforced by the propagator (including custom hybrid consistencies). No matter how carefully the test suites of AbsCon [24], Choco [29], JaCoP [22] and MiniCP [23] have been engineered, experiments with SolverCheck revealed non-trivial bugs.

## 1 Introduction

Le succès de la programmation par contraintes doit beaucoup à l'aspect déclaratif de ses modèles et à l'expressivité de son large catalogue de contraintes [6] pour lesquelles il existe des propagateurs efficaces. Ce modèle est néanmoins fragile, car les résultats fournis par un solveur CP pourraient être invalidés par une implémentation incorrecte d'un seul de ces propagateurs. Or il se fait que les algorithmes et structures de données mis en œuvre par ceux-ci sont assez avancés et font notamment intervenir des mécanismes de restauration d'état. C'est pourquoi il est essentiel de s'assurer de l'exactitude de leur implémentation. Cependant, une validation qui se bornerait uniquement à vérifier qu'aucune solution n'est supprimée par erreur ne pourrait s'avérer suffisante. En effet, afin de pouvoir traiter des problèmes issus du monde réel, il est essentiel qu'un solveur soit à la fois correct et efficace. En pratique cette efficacité est obtenue grâce à un compromis entre la consistance imposée par un propagateur et la complexité de l'algorithme qu'il implémente. De ce fait, il est nécessaire d'être capable de tester le niveau de consistance imposé par un propagateur, car si celui-ci s'avérait être plus faible qu'annoncé, certaines instances pourraient devenir intraitables de manière insidieuse.

Dans ce contexte, nous proposons SolverCheck : une bibliothèque de tests par propriétés (property-based testing, PBT) libre inspirée par QuickCheck [13]. Celle-ci a été spécifiquement pensée pour améliorer la qualité et l'efficacité des tests utilisés pour valider les solveurs CP. Concrètement, SolverCheck permet aisément d'à la fois tester la correction des propagateurs implémentés dans un solveur mais aussi le niveau de consistance imposé par ceux-ci. En outre, SolverCheck se veut être une bibliothèque extensible. Elle fournit donc à un utilisateur les interfaces nécessaires pour décrire la relation imposée par une nouvelle contrainte via un checker. De la même façon, le niveau de consistance n'est pas nécessairement cantonné aux consistances classiques (GAC, Bound(D), Bound(Z), Range, FC) [7], car SolverCheck permet de définir des consistances hybrides (personnalisées) qui correspondent au comportement attendu en pratique par un propagateur.

La suite de cet article est structurée comme suit : dans un premier temps, la section 2 vous présentera le paradigme de test dans lequel s'inscrit SolverCheck et la problématique à laquelle notre outil essaie de ré-

*Papier doctorant : Xavier Gillard est auteur principal.

pondre. La section 3 vous introduira brièvement les travaux connexes aux nôtres et explicitera en quoi l'approche de SolverCheck diffère. Ensuite, la section 4 vous présentera les différentes caractéristiques de notre outil au travers d'un exemple simple mais qui permet de brosser un aperçu des fonctionnalités offertes par la bibliothèque. Après cela, la section 5 couvrira certains résultats expérimentaux qui valident la pertinence de notre approche.

## 2 Test par propriétés

Le paradigme de test dans lequel s'inscrit SolverCheck est celui du *test par propriété*. Celui-ci répond aux faiblesses de l'approche classique (*test par l'exemple*) qui est utilisée pour valider l'implémentation de la plupart des solveurs et notamment AbsCon [24] Choco ă[29], JaCoP [22], MiniCP[23], OScaR[28]. En effet, l'approche classique impose au testeur de définir manuellement des cas de tests jugés pertinents, ce qui n'est pas une tâche aisée requérant à la fois intuition et expertise. Il en résulte que le code de test est souvent de moindre qualité que le code testé et comprend de nombreux passages dupliqués. Par ailleurs, ces tests avec des valeurs codées en dur adoptent souvent un style très impératif qui permet difficilement de comprendre et mettre en évidence la propriété testée par un cas particulier.

En réponse à cela, le test par propriété propose de résoudre ces faiblesses en combinant *une spécification formelle* de la propriété à tester avec une *génération aléatoire* de cas de tests. Dans ce contexte, le testeur exprime les propriétés générales qui doivent être satisfaites par une implémentation et laisse le soin de générer des cas de test concrets à la librairie utilisée. Ces propriétés sont exprimées dans un langage déclaratif de haut niveau qui abstrait les détails du test et permet au testeur de se concentrer sur *ce qui* doit être vérifié plutôt que sur *la manière* dont cette vérification doit avoir lieu.

## 3 Travaux connexes

L'objet de nos travaux diffère de ceux qui ont été entrepris dans le courant des années '90-2000 [16, 18, 15, 25]. En effet, ces travaux visent principalement à vérifier que le *programme CP* est correct alors que SolverCheck teste l'*implémentation du solver* utilisé pour résoudre celui-ci.

Bien que les propriétés à tester soient exprimées de manière formelle, SolverCheck est une bibliothèque de test et non pas un outil de vérification formelle, ce qui le rend plus simple à utiliser. En effet, ceux-ci ne supportent généralement pas toutes les constructions du langage utilisé, requièrent une guidance humaine [1, 4] ou ne sont pas applicables pour des programmes aussi larges et complexes que les solveurs CP actuels [20, 19, 21]. De même, le développement de solveurs CP certifiés formellement [17, 12] ne permet pas – jusqu'à présent – d'égaler les solveurs CP actuels ; tant au niveau de l'efficacité (code suboptimal généré par une extraction de code OCaml depuis Coq [14]) qu'au niveau des fonctionnalités (ni contraintes ternaires, ni contraintes n-aires).

Récemment, la communauté SAT/SMT/ASP/QBF a entrepris de réaliser des travaux connexes aux nôtres [11, 10, 3, 27]. Ceux-ci proposent aussi d'utiliser du fuzzing (génération d'entrées aléatoires) pour s'assurer de la qualité de leurs outils. Cependant, contrairement à SolverCheck, ces travaux ne s'intéressent pas au cas d'un solveur CP. En particulier, ils ignorent les problématiques liées à la consistance (hybride ou non) des propagateurs. Or, comme cela a été dit précédemment, il s'agit d'un aspect essentiel du raisonnement et du développement d'un solveur CP.

Par ailleurs, Akgün et al. ont décrit lors de la dernière conférence CP une approche visant à valider l'implémentation d'un solveur CP via des tests métamorphiques [2]. L'objectif et l'intuition qui sous tendent leurs travaux sont les mêmes que pour SolverCheck. Dans les deux cas, il s'agit de valider l'implémentation des contraintes d'un solveur en comparant le résultat de deux implémentations distinctes d'un même propagateur. Les différences essentielles entre les deux approches sont que : d'une part, leur méthode requiert que le testeur fournisse une table avec toutes les solutions de la contrainte alors que SolverCheck dérive automatiquement une implémentation naïve du propagateur. D'autre part, SolverCheck mets l'accent sur les consistances hybrides, ce qui n'est pas le cas de [2].

Enfin, il nous parait important de mentionner que Gecode [31] adopte une stratégie de test originale ; laquelle lui permet de valider la correction et le niveau de consistance (AC, BC(D) ou BC(Z)) [7] imposé par ses propagateurs. Pour ce faire, il exploite habilement la propriété de monotonicité-faible [30] des propagateurs et le fait que toutes les valeurs d'un domaine filtré ont un (bound) support. Cette approche, bien qu'élégante et efficace, ne permet pas de tester la consistance d'un propagateur si celui-ci impose une consistance hybride (comme c'est le cas de la contrainte *element* [32]).

## 4 SolverCheck

Dans cet article, nous aborderons le langage fourni par SolverCheck au travers d'un exemple issu de la suite de tests que nous avons créé pour JaCoP.

**Listing 1 – Exemple : LexOrder(≤) doit être GAC.**

```
1 @Test
2 public void statelessLexLE() {
3  assertThat(
4   forAll(listOf("x", jDom())).assertThat(x ->
5   forAll(listOf("y", jDom())).assertThat(y ->
```

```
6    a(statelessJacopLexOrder(false))
7    .isEquivalentTo(
8     arcConsistent(lexLE(x.size(), y.size())))
9    .forAnyPartialAssignment().with(x).then(y))));
10 }
11
12 // Generer un domaine acceptable par JaCoP
13 public GenDomainBuilder jDom() {
14  return domain().withValuesBetween(
15    IntDomain.MinInt,
16    IntDomain.MaxInt);
17 }
18 // Discriminer les solutions
19 public Checker lexLE(int x_sz, int y_sz) {
20  return assignment -> {
21    var xs = assignment.subList(0, x_sz);
22    var ys = assignment.subList(x_sz, x_sz+y_sz);
23    for(int i=0; i < min(x_sz, y_sz); i++) {
24     if (xs.get(i) < ys.get(i)) return true;
25     if (xs.get(i) > ys.get(i)) return false;
26    }
27    return x_sz <= y_sz;
28  };
29 }
```

### 4.1 Test déclaratif

Le code de test donné dans le listing-1 illustre l'aspect déclaratif de SolverCheck. La propriété testée (lignes 3-9) stipule qu'étant donné deux listes $x$ et $y$ de variables dont le domaine n'est contraint que par la plage des valeurs acceptées par JaCoP (lignes 14-16) ; le filtrage des domaines obtenu par l'implémentation concrète est égal à ce qui serait attendu en théorie pour une contrainte et un niveau de consistance donné.

### 4.2 Extensible

Malgré son apparente simplicité, l'exemple reproduit dans le listing-1 illustre bien la flexibilité offerte par SolverCheck. En effet, les lignes 19-27 montrent comment étendre les capacités de SolverCheck pour qu'il puisse tester de façon automatique des propriétés portant sur des contraintes qui lui sont a priori inconnues [1]. Il suffit pour ce faire de définir un nouveau `Checker`, c'est à dire, un prédicat portant sur les assignations complètes ; lequel est vrai ssi l'assignation complète satisfait la contrainte voulue.

### 4.3 Consistance

De la même façon, il est possible de paramétrer le niveau de consistance utilisé pour tester le propagateur. Pour ce faire, afin de modifier la propriété exprimée dans l'exemple du listing-1 de sorte qu'elle spécifie que le niveau de filtrage du propagateur testé soit BC(Z) plutôt qu'AC, il suffirait de remplacer la ligne 8 par `boundZConsistent(lexLE(x.size(), y.size()))`. De plus, étant donné que le niveau de filtrage implémenté dans les contraintes offertes pas les solveurs ne correspondent pas toujours à la définition d'une des consistances "pures" ; SolverCheck permet de spécifier qu'un filtre peut être plus fort (`isStrongerThan()`), plus faible (`isWeakerThan()`) ou équivalent (`isEquivalentTo()`) à niveau de consistance donné. Ceci est illustré par la ligne 7 de notre exemple. Cependant, on peut considérer qu'il n'est pas suffisant de pouvoir dire qu'un filtre est plus fort ou plus faible qu'une consistance donnée. C'est pourquoi SolverCheck permet en outre de définir de nouvelles consistances hybrides. L'exemple du listing-2 illustre la façon de spécifier la consistance hybride exacte d'un propagateur. Cet exemple nous montre que la contrainte $element(A, x, y) \equiv A[x] = y$ de MiniCP dans laquelle $A$ est un tableau d'entier et $x, y$ deux variables, impose que chaque valeur du domaine de $x$ doit avoir un *bound-support* en $y$ alors que seules les bornes sup. et inf. du domaine de $y$ doivent avoir un support en $x$. De plus, cet exemple montre aussi (ligne 3) comment donner une limite de temps à SolverCheck pour l'exécution de ses tests.

**Listing 2 – A[x] = y a une consistance hybride**

```
1  @Test
2  public void elementIsHybridConsistent() {
3   given(TIMELIMIT, TimeUnit.SECONDS).assertThat(
4    forAll(listOf("A", integer())).assertThat(A ->
5    forAll(domain("x")).assertThat(x ->
6    forAll(domain("y")).assertThat(y ->
7      a(minicpElement1D(A))
8      .isEquivalentTo(hybrid(element(A),
9        rangeDomain(),
10       bcDDomain()))
11    .forAnyPartialAssignment().with(x).then(y)
12  ))));
13 }
```

### 4.4 Test dynamique

Étant donné qu'un nombre non-négligeable de contraintes implémentent une propagation incrémentale en utilisant des structures à restauration d'état, il est nécessaire de s'assurer du bon fonctionnement du propagateur à tout moment de la recherche. En particulier, il faut s'assurer que les propriétés qui sont énoncées soient valides même après plusieurs affectations et restaurations d'état. C'est pourquoi SolverCheck propose un mode de test *dynamique* en plus des tests *statiques* qui ont été présentés jusqu'à maintenant.

Comme dans le cas d'un test statique, lorsque SolverCheck teste dynamiquement une propriété, il commence par générer un input aléatoire. Cependant, contrairement au cas statique, il considère cet input comme la racine d'un arbre de recherche et réalise des plongées pour explorer une fraction de celui-ci. A chaque noeud de l'arbre, le système vérifie que la propriété testée est bien maintenue ; sans quoi une trace menant à un état dans lequel la propriété est violée est rapportée à l'utilisateur. Lorsqu'une feuille de l'arbre est rencontrée, le système décide de remonter jusqu'à un noeud (choisi arbitrairement sur la dernière branche explorée) avant d'entamer une nouvelle plongée.

1. La bibliothèque fournit un certain nombre de checkers pour les contraintes les plus usuelles (alldiff, element, gcc, etc...).

En pratique, pour activer ce mode de test, il suffit à un utilisateur de fournir un `StatefulFilter` (dont l'interface est reproduite par le listing-3) plutôt qu'un `Filter` et d'utiliser le mot clé *stateful* pour encadrer la déclaration du filtrage attendu. Concrètement, pour adapter l'exemple du listing-1, il suffirait de modifier la ligne 8 pour qu'elle ait la forme suivante : `stateful(arcConsistent(...))))`.

**Listing 3 – Tests dynamiques : Interface StatefulFilter**

```
1 public interface StatefulFilter {
2  void setup(PartialAssignment initialDomains);
3  void pushState();
4  void popState();
5  void branchOn(int var, Operator op, int val);
6  PartialAssignment currentState();
7 }
```

## 5 Évaluation

Afin d'évaluer la capacité de SolverCheck à identifier des problèmes dans l'implémentation de contraintes par des solveurs réels, nous avons choisi d'utiliser notre outil pour tester les contraintes de quatre solveurs basés sur la machine virtuelle Java ; à savoir : AbsCon[24], Choco[29], JaCoP[22] et MiniCP[23]. Ceux-ci ont été choisis d'une part, parce qu'ils tournent sur la JVM qui est la plate-forme cible de SolverCheck et d'autre part parce qu'ils ont été développés avec soin par des experts du domaine [2]. Parmi le large éventail de contraintes possibles, nous en avons sélectionné six qui nous semblaient être représentatives du type de contraintes généralement disponibles dans un solveur CP.

La table-1 résume les résultats observés lors de cette recherche de bugs. Comme on peut le voir, SolverCheck a été capable d'identifier des problèmes dans chacun des solveurs testés et cela pour pratiquement chacune des contraintes considérées. Et cela en dépit du fait que ces contraintes avaient déjà été testées par les auteurs de ces solveurs. La nature des problèmes identifiés est, elle aussi, très variable : la stratégie de génération d'inputs pseudo-aléatoires biaisée pour générer des valeurs extrêmes a naturellement permis la mise en évidence d'un certain nombre de problèmes d'over- et under-flows. Ceux-ci sont souvent contre-intuitifs pour un être humain et peuvent avoir un impact désastreux sur le solveur considéré : crash, boucle infinie et décision incorrecte de la contrainte sont quelques exemples qui ont été observés. Les problèmes de dépassement sur les entiers finis ne sont toutefois pas les seuls problèmes à avoir été identifiés par notre outil. Parmi ceux-ci, on compte évidemment les problèmes de consistance plus faible qu'annoncée et des erreurs de programmation (off-by-one, erreurs de cast, etc..). Mais on observe aussi d'autres types de problèmes tels que des erreurs logiques, des fuites mémoires et des désaccords entre le comportement réel et documenté d'un propagateur.

2. D'autres choix auraient bien entendu été possibles. Par exemple, il aurait tout à fait été envisageable d'intégrer OScaR [28] à la liste des solveurs testés malgré le fait qu'il soit écrit en `Scala` et non pas en `Java`.

Table 1 – Problèmes trouvés lors de l'évaluation

| Solver | Alldiff | Element | Table | Sum | GCC | Lex |
|---|---|---|---|---|---|---|
| AbsCon | oui | oui | oui | oui | oui | oui |
| Choco | oui | oui | oui | oui | non | oui |
| JaCoP | non | oui | non | non | non | oui |
| MiniCP | oui | non | oui | oui | N/A | N/A |

## 6 Conclusions

Dans cet article, nous avons présenté SolverCheck, une bibliothèque de test par propriété spécifiquement conçue pour évaluer la qualité des solveurs CP. Au travers d'un exemple simple, nous avons mis en évidence le caractère déclaratif des propriétés énoncées via le DSL de SolverCheck. Toujours sur base du même exemple, nous avons montré que la bibliothèque que nous proposons est extensible et peut être facilement complétée de sorte à traiter des contraintes qui n'ont pas été initialement été prévues. Nous avons aussi montré que le langage de SolverCheck était suffisamment flexible que pour permettre de spécifier de manière assez fine la consistance attendue d'un propagateur ; ce qui facilite le raisonnement à propos de ce dernier. Enfin, nous avons montré que SolverCheck permet non seulement de réaliser des tests statiques de contraintes, mais aussi des tests dynamiques de celles-ci. Cela, afin de rendre compte de l'aspect incrémentiel inhérent à certains algorithmes efficaces. Enfin, l'évaluation de notre outil sur des contraintes et solveurs réels a permis de montrer la redoutable efficacité de celui-ci pour identifier et corriger des problèmes non triviaux dans les solveurs. De ce fait, nous pensons qu'une adoption plus large de SolverCheck serait bénéfique à la communauté dans son ensemble, car cela permettrait d'augmenter la qualité du code des solveurs tout en libérant du temps pour les développeurs qui devraient passer moins de temps à imaginer et copier-coller des cas de test. A fortiori puisque SolverCheck s'utilise facilement au travers de tests JUnit [5]/TestNG [8] et peut donc sans problème s'intégrer à un système d'intégration continue.

Nous envisageons plusieurs extensions à ces travaux. Entre autre, étendre SolverCheck de sorte qu'il génère et interprète des fragments de code XCSP3 [9] ou MiniZinc [26] afin de s'affranchir du cadre de la JVM et ainsi tester des solveurs qui ne tournent pas sur cette plateforme. D'autres options incluent le microbenchmarking et test de contraintes de scheduling.

## Références

[1] Wolfgang Ahrendt, Bernhard Beckert, Richard Bubel, Reiner Hähnle, Peter H Schmitt et Mattias Ulbrich : *Deductive Software Verification–The KeY Book : From Theory to Practice*, volume 10001. Springer, 2016.

[2] Özgür Akgün, Ian P Gent, Christopher Jefferson, Ian Miguel et Peter Nightingale : Metamorphic testing of constraint solvers. *In International Conference on Principles and Practice of Constraint Programming*, pages 727–736. Springer, 2018.

[3] Cyrille Artho, Armin Biere et Martina Seidl : Model-based testing for verification back-ends. *In International Conference on Tests and Proofs*, pages 39–55. Springer, 2013.

[4] John Barnes : *SPARK : The Proven Approach to High Integrity Software*. Altran Praxis, 2012.

[5] Kent Beck et Erich Gamma : Test infected : Programmers love writing tests. *Java Report*, 3(7) :37–50, 1998.

[6] Nicolas Beldiceanu, Mats Carlsson et Jean-Xavier Rampon : Global constraint catalog, 2010.

[7] Christian Bessiere : Constraint propagation. *In Foundations of Artificial Intelligence*, volume 2, pages 29–83. Elsevier, 2006.

[8] Cedric Beust et Alexandru Popescu : Testng : Testing, the next generation, 2007.

[9] Frédéric Boussemart, Christophe Lecoutre et Cédric Piette : Xcsp3 : An integrated format for benchmarking combinatorial constrained problems. *arXiv preprint arXiv :1611.03398*, 2016.

[10] Robert Brummayer et Matti Järvisalo : Testing and debugging techniques for answer set solver development. *Theory and Practice of Logic Programming*, 10(4-6) :741–758, 2010.

[11] Robert Brummayer, Florian Lonsing et Armin Biere : Automated testing and debugging of sat and qbf solvers. *In International Conference on Theory and Applications of Satisfiability Testing*, pages 44–57. Springer, 2010.

[12] Matthieu Carlier, Catherine Dubois et Arnaud Gotlieb : A certified constraint solver over finite domains. *In International Symposium on Formal Methods*, pages 116–131. Springer, 2012.

[13] Koen Claessen et John Hughes : Quickcheck : a lightweight tool for random testing of haskell programs. *Acm sigplan notices*, 46(4) :53–64, 2011.

[14] The coq development team : The coq proof assistant, En ligne : 10 mai 2019 (`https://coq.inria.fr/`).

[15] Michael Dahmen : A debugger for constraints in prolog. 1991.

[16] Romuald Debruyune, Jean-Daniel Fekete, Narendra Jussien et Mohammad Ghoniem : Proposition de format concret pour des traces générées par des solveurs de contraintes réalisation rntl oadymppac 2.2. 2.1. 2001.

[17] Catherine Dubois et Arnaud Gotlieb : Solveurs cp (fd) vérifiés formellement. *In Neuvi ? mes Journ ? es Francophones de Programmation par Contraintes (JFPC 2013)*, pages 115–118, 2013.

[18] Mireille Ducassé : Opium+, a meta-debugger for prolog. *In European Conference on Artificail Intelligence*, 1988.

[19] Jean-Christophe Filliâtre et Claude Marché : The why/krakatoa/caduceus platform for deductive program verification. *In International Conference on Computer Aided Verification*, pages 173–177. Springer, 2007.

[20] Jean-Christophe Filliâtre et Andrei Paskevich : Why3 – Where Programs Meet Provers. *In ESOP'13 22nd European Symposium on Programming*, volume 7792 de *LNCS*, Rome, Italy, mars 2013. Springer.

[21] Florent Kirchner, Nikolai Kosmatov, Virgile Prevosto, Julien Signoles et Boris Yakobowski : Frama-c : A software analysis perspective. *Formal Aspects of Computing*, 27(3) :573–609, 2015.

[22] Krzysztof Kuchcinski et Radoslaw Szymanek : Jacop-java constraint programming solver. *In CP Solvers : Modeling, Applications, Integration, and Standardization, co-located with the 19th International Conference on Principles and Practice of Constraint Programming*, 2013.

[23] Laurent Michel, Pierre Schaus, Pascal Van Hentenryck : MiniCP : A lightweight solver for constraint programming, 2018. Available from `https://minicp.bitbucket.io`.

[24] Christophe Lecoutre et Sebastien Tabary : Abscon 112 : towards more robustness. *In 3rd International Constraint Solver Competition (CSC'08)*, pages 41–48, 2008.

[25] Micha Meier : Debugging constraint programs. *In International Conference on Principles and Practice of Constraint Programming*, pages 204–221. Springer, 1995.

[26] Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck et Guido Tack : Minizinc : Towards a standard cp modelling language. *In International Conference on Principles and Practice of Constraint Programming*, pages 529–543. Springer, 2007.

[27] Aina Niemetz, Mathias Preiner et Armin Biere : Model-based api testing for smt solvers. *In Proceedings of the 15th International*

*Workshop on Satisfiability Modulo Theories, SMT*, page 10, 2017.

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

[29] Charles Prud'homme, Jean-Guillaume Fages et Xavier Lorca : *Choco Documentation*. TASC - LS2N CNRS UMR 6241, COSLING S.A.S., 2017.

[30] Christian Schulte et Guido Tack : Weakly monotonic propagators. *In* Ian P. Gent, éditeur : *Principles and Practice of Constraint Programming - CP 2009*, volume 5732 de *Lecture Notes in Computer Science*, pages 723–730, Berlin, Heidelberg, septembre 2009. Springer.

[31] Christian Schulte, Guido Tack et Mikael Z. Lagerkvist : Modeling. *In* Christian Schulte, Guido Tack et Mikael Z. Lagerkvist, éditeurs : *Modeling and Programming with Gecode*. 2018. Corresponds to Gecode 6.1.0.

[32] Pascal Van Hentenryck et Jean-Philippe Carillon : Generality versus specificity : An experience with ai and or techniques.

# Estimer le nombre de solutions des contraintes de cardinalité grâce à leur décomposition `range` et `roots`

**Giovanni Lo Bianco**[1*] **Xavier Lorca**[2] **Charlotte Truchet**[3]

[1] IMT Atlantique, Nantes
[2] IMT Mines Albi-Carmaux, Albi
[3] Université de Nantes
giovanni.lo-bianco@imt-atlantique.fr

**Résumé**

En programmation par contraintes, le choix d'une heuristique de recherche plutôt qu'une autre dépend souvent du problème. Cependant il existe des heuristiques génériques utilisant plutôt des indicateurs sur la structure combinatoire du problème. Les heuristiques "Counting-Based", introduites par Pesant et al., font des choix basés sur une estimation du nombre de solutions restantes dans tel ou tel sous-arbre de l'arbre de recherche. Un inconvénient de ces heuristiques est qu'elles nécessitent des algorithmes de dénombrement spécifiques à chaque contrainte. Cette étude s'intéresse aux contraintes de cardinalité, dont alldifferent, atmost, nvalue, etc... Nous proposons une méthode de comptage de solutions pour les contraintes range et roots, introduites par Bessiere et al. Grâce à la décomposition des contraintes de cardinalité en contraintes range et roots, nous dérivons une méthode systématique de dénombrement de solutions pour la plupart de ces contraintes.

**Abstract**

This paper proposes a systematic approach for solution counting on cardinality constraints. A main limitation of the classical counting-based search approaches is due to the fact that counting solutions is at least as hard as the constraints' development itself. A typical example is the process of counting solutions for the `alldifferent` and `gcc` constraints in the context of counting-based search [6]. This paper shows that it is possible to obtain a systematic and efficient method by counting solutions on the global constraint decomposition. For this purpose, we use the `range` and `roots` constraint decomposition [3] to count solutions for a large family of cardinality constraints such as `alldifferent`, `disjoint`, `nvalue`, `atmost`, `atleast`, etc.

## 1 Introduction

L'un des sujets de recherche actifs en Programmation par contraintes est la conception d'heuristiques indépendantes des problèmes traités, mais tout de même adaptées à leur structure combinatoire. Dans [6], les auteurs présentent les heuristiques appelées counting-based search. Ces heuristiques sélectionnent une paire variable/valeur en se basant sur le nombre de solutions restant après l'instanciation. Par exemple, l'heuristique `maxSD`,(pour maximal solutions density), explore d'abord la branche de l'arbre de recherche qui promet le plus de solutions. En pratique, elle évalue, pour chaque contrainte du problème individuellement, la densité de solutions pour chaque paire variable/valeur impliquée dans cette contrainte. De telles heuristiques requièrent des algorithmes de dénombrement de solutions dédiés pour chaque contrainte. Ces algorithmes de dénombrement sont donnés pour `alldifferent`, `gcc`, `knapsack` et `regular` dans [6]. L'une des limites de ces heuristiques est que la conception d'un algorithme de comptage efficace pour une contrainte spécifique est aussi difficile que le développement de la contrainte elle-même.

Dans cet article, nous portons nos efforts sur plusieurs contraintes de cardinalité : `alldifferent`, `disjoint`, `nvalue`, `atleast_nvalue`, `atmost_nvalue`, `atleast`, `atmost`. Ces contraintes limitent le nombre d'occurrences de certaines valeurs ou le nombre de va-

*Papier doctorant : Giovanni Lo Bianco[1] est auteur principal.

leurs différentes dans une solution. Elles peuvent être modélisées grâce à des graphes bipartis et [6] montre, pour `alldifferent` et `gcc` que le problème consistant à dénombrer les solutions sur ces contraintes peut être ramené à un problème de dénombrement de couplages dans ces graphes. La résolution de tels problèmes est très difficile : ils appartiennent souvent à la classe de complexité #P-complet. C'est pourquoi les counting-based search ne reposent pas sur un comptage exact, mais sur des estimations du nombre réel de solutions. Dans [6], les auteurs donnent une estimation du nombre de solutions sur `alldifferent` et `gcc` avec une borne supérieure. Dans cet article, nous proposons une approche probabiliste pour calculer de telles estimations.

Dans [3], les auteurs présentent deux nouvelles contraintes globales `range` et `roots`, qui spécifient un grand nombre de contraintes de cardinalité. En d'autres termes, pour presque toutes les contraintes de cardinalité, il existe un modèle équivalent utilisant uniquement les contraintes plus primitives `range` et `roots` (ainsi que certaines contraintes arithmétiques). Ce modèle équivalent est appelé la décomposition de la contrainte initiale. Dans [3], les auteurs développent un algorithme de propagation pour `range` et pour `roots` et montrent que l'étape de propagation peut être plus efficace en utilisant la décomposition plutôt que l'algorithme de propagation de la contrainte initial.

Dans cet article, nous utilisons la décomposition en contraintes `range` et `roots` pour compter les solutions. Plus précisément, nous développons une approche probabiliste pour estimer le nombre de solutions sur les contraintes `range` et `roots` et nous développons une méthode systématique pour estimer le nombre de solutions sur de nombreuses contraintes de cardinalité. Par rapport à [6], nous obtenons une estimation au lieu d'une borne supérieure et nous proposons une méthode qui peut être généralisée à un grand nombre de contraintes de cardinalité sans redévelopper un modèle mathématique dédié.

**Plan :** L'article est organisé de la manière suivante. La section 2 sert d'introduction aux contraintes `range` et `roots` et donne quelques éléments pour comprendre le modèle de graphe biparti associé. Dans la section 3, nous détaillons comment compter exactement le nombre de solutions sur `range` et `roots` puis nous utilisons un modèle probabiliste sur ces contraintes pour estimer le nombre réel de solutions. Dans la section 4, nous donnons la décomposition en contraintes `range` et `roots` et un estimateur du nombre de solutions pour plusieurs contraintes de cardinalité.

## 2 Résultats préliminaires : Introduction à `range` et `roots`

Dans toute cette étude, nous utiliserons les notations suivantes. Soit $X = \{x_1, \ldots, x_n\}$, l'ensemble des variables. Pour chaque variable $x_i \in X$, on note $D_i$, son domaine, $Y = \bigcup_{i=1}^{n} D_i = \{y_1, \ldots, y_m\}$, l'union des domaines et $\mathcal{D} = D_1 \times \ldots \times D_n$,, le produit cartésien des domaines. On note $d_i = |D_i|$, la taille du domaine de $x_i$. Soit $C$, une contrainte portant sur l'ensemble $X$, on note $\mathcal{S}_{C(X)}$, l'ensemble des tuples autorisés par $C$ sur $X$ et on note $\#C(X)$, le nombre de tuples autorisés.

Les contraintes de cardinalités portent sur le nombre d'occurrences dans la solution d'une valeur spécifique, ou sur le nombre de valeurs ou variables répondant à certaines critères. Parmi elles, nous pouvons lister `alldifferent`, `disjoint`, `nvalue`, `atleast_nvalue`, `atmost_nvalue`, `atleast`, `atmost`. Nous reviendrons sur ces contraintes et les définirons convenablement une à une dans la section 4. Ces contraintes peuvent, la plupart du temps, être modélisées à l'aide de graphes bipartis, dans lesquelles nous recherchons certaines structures, comme, par exemple, des couplages.

**Définition 1** (Graphe des domaines)**.** *Soit $G_{X,Y} = G(X \cup Y, E)$, le graphe sur les noeuds $X \cup Y$, et les arêtes $E = \{(x_i, y_j) \mid y_j \in D_i\}$. $G_{X,Y}$ est un graphe biparti représentant le domaine des variables. Il y a une arête entre $x_i$ et $y_j$ ssi $y_j \in D_i$.*

**Exemple 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 $D_5 = \{2, 4, 5\}$. Nous obtenons le graphe des domaines $G_{X,Y}$ tel que sur la Figure 1a.*

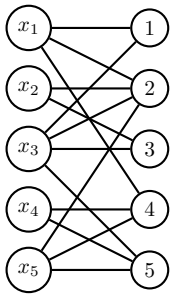
Nous définissons aussi le sous-graphe des domaines induit par les sous-ensembles $X' \subseteq X$ and $Y' \subseteq Y$, comme le graphe des domaines restreints aux sous ensembles de noeuds considérés

**Définition 2** (Sous-graphe des domaines)**.** *Soit $G_{X',Y'} = G(X' \cup Y', E)$, le graphe des domaines de $X'$ avec $E = \{(x_i, y_j) \mid y_j \in D'_i = D_i \cap Y'\}$. $G_{X',Y'}$ est un graphe biparti représentant les sous-domaines induits par $Y'$ de chaque variable. Il y a une arête entre $x_i$ et $y_j$ ssi $y_j \in D'_i$*

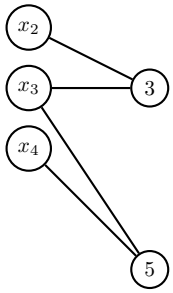
On note $d_i(Y') = |D'_i|$, la taille du sous-domaine de $x_i$ restreint aux valeurs de $Y'$

L'exemple 2 illustre un sous-graphe des domaines possible au graphe des domaines présenté dans l'exemple 1

**Exemple 2.** *Soit $X' = \{x_2, x_3, x_4\} \subseteq X$ et $Y' = \{3, 5\} \subseteq Y$. Le sous-graphe des domaines induit par $X'$ et $Y'$ est représenté dans la Figure 1b*




(a) Value Graph $G_{X,Y}$




(b) Sub-Value Graph $G_{X',Y'}$

FIGURE 1 – Value graph and Sub-Value Graph of Example 1 and Example 2.

Les contraintes `range` et `roots` [3] sont deux contraintes auxiliaires qui servent à modéliser la plupart des contraintes de cardinalité. Dans cette étude, nous nous servons de la décomposition en contraintes `range` et `roots` pour dénombrer les solutions des contraintes de cardinalité. La contrainte `range` encapsule la notion d' "image" d'une fonction et la contrainte `roots` encapsule la notion de "domaine". Nous utiliserons ici, une formalisation différente pour ces contraintes de la manière dont elles sont présentées dans [3].

**Définition 3** (`range`). *Soit $X' \subseteq X$ et $Y' \subseteq Y$. La contrainte* `range`$(X, X', Y')$ *est satisfaite si les valeurs affectées aux variables de $X'$ couvrent* ***exactement*** *$Y'$ et pas plus. Formellement :*

$$\mathcal{S}_{range(X,X',Y')} = \{(v_1, \ldots, v_n) \in \mathcal{D} \mid \{v_i | x_i \in X'\} = Y'\} \tag{1}$$

**Définition 4** (`roots`). *Soit $X' \subseteq X$ et $Y' \subseteq Y$. La contrainte* `roots`$(X, X', Y')$ *est satisfaite si les variables qui sont instanciées aux valeurs de $Y'$ couvrent* ***exactement*** *$X'$ et pas plus. Formellement :*

$$\mathcal{S}_{roots(X,X',Y')} = \{(v_1, \ldots, v_n) \in \mathcal{D} \mid \{x_i | v_i \in Y'\} = X'\} \tag{2}$$

Dans l'exemple 3, nous illustrons comment ces contraintes fonctionnent et pourquoi ces contraintes ne sont pas l'inverse l'une de l'autre.

**Exemple 3.** *Reprenons le graphe des domaines de l'example 1a.*

- *Le tuple $(2, 2, 3, 4, 5)$ est autorisé par la contrainte* `range`$(X, \{x_1, x_2, x_3\}, \{2, 3\})$.
- *Le tuple $(2, 2, 5, 4, 5)$ est autorisé par la contrainte* `range`$(X, \{x_1, x_2, x_3\}, \{2, 3\})$ *mais pas par* `range`$(X, \{x_1, x_2, x_3\}, \{2, 5\})$.
- *Le tuple $(2, 2, 3, 4, 5)$ est autorisé par la contrainte* `roots`$(X, \{x_1, x_2, x_3\}, \{2, 3\})$.
- *Le tuple $(2, 2, 3, 4, 2)$ n'est pas autorisé par la contrainte* `roots`$(X, \{x_1, x_2, x_3\}, \{2, 3\})$, *mais l'est par* `roots`$(X, \{x_1, x_2, x_3, x_5\}, \{2, 3\})$.
- *Le tuple $(2, 2, 2, 4, 5)$ est autorisé par la contrainte* `range`$(X, \{x_1, x_2, x_3\}, \{2\})$ *mais pas par* `range`$(X, \{x_1, x_2, x_3\}, \{2, 3\})$, *puisque 3 n'est ni assigné à $x_1, x_2$ ou $x_3$. En revanche, le tuple $(2, 2, 2, 4, 5)$ est autorisé par* `roots`$(X, \{x_1, x_2, x_3\}, \{2, 3\})$. *En effet, 3 n'est assigné à aucune variable, donc $X'$ n'est défini que par les variables instanciées à 2.*

La raison pour laquelle `range` et `roots` ne sont pas l'inverse l'une de l'autre est que toutes les variables doivent être instanciées à une valeur mais une valeur n'est pas nécessairement affectée à une variable.

## 3 Dénombrement de solutions sur les contraintes `range` et `roots`

Le dénombrement de solutions sur les contraintes de cardinalité nécessite des algorithmes dédiés à chaque contrainte. Dans cette section, nous nous intéressons au dénombrement de solutions sur les contraintes `range` et `roots`. L'idée est alors d'utiliser la décomposition des contraintes de cardinalité en ces contraintes plus primitives et de réutiliser la méthode de comptage sur `range` et `roots` pour dénombrer les solutions sur les contraintes de cardinalité.

### 3.1 Dénombrement exact sur `range` and `roots`

Dans cette sous-section, nous sommes intéressés par le dénombrement exact des tuples autorisés par les contraintes `range` et `roots`.

**Proposition 1.** *Soit $X' \subseteq X$ et $Y' \subseteq Y$. On note $\overline{X'}$, le complémentaire de $X'$ dans $X$, tel que $\overline{X'} \cup X' = X$ et $\overline{X'} \cap X' = \emptyset$. Le nombre de tuples autorisés par* `range`$(X, X', Y')$ *est*

$$\#\texttt{range}(X, X', Y') = \#\texttt{range}(X', X', Y') \cdot \prod_{x_i \in \overline{X'}} d_i \tag{3}$$

*Démonstration.* D'un côté, nous devons considérer toutes les instanciations possibles des variables de $\overline{X'}$, qui ne sont pas contraintes : $\prod_{x_i \in \overline{X'}} d_i$

Et de l'autre, nous devons énumérer toutes les instanciations possibles pour les variables de $X'$ qui sont contraintes : $\#\texttt{range}(X', X', Y')$ □

Proposition 1 réduit le problème du comptage d'instanciations autorisées pour les variables de $X$ au comptage d'instanciations autorisées pour les variables contraintes de $X'$ seulement. Nous cherchons maintenant à dénombrer les tuples autorisés lorsque que toutes les variables de $X$ sont contraintes.

**Proposition 2.**

$$\#\mathit{range}(X, X, Y) = \prod_{x_i \in X} d_i - \sum_{Y' \subsetneq Y} \#\mathit{range}(X, X, Y') \tag{4}$$

*Démonstration.* Dans $G_{X,Y}$, nous devons compter toutes les affectations possibles des variables de $X$ de sorte que toutes les valeurs de $Y$ soient couvertes. Pour ce faire, nous comptons d'abord le nombre d'affectations possibles des variables de $X$ dans $G_{X,Y}$ (sans tenir compte de la contrainte `range`) :

$$\prod_{x_i \in X} d_i$$

Et ensuite, nous retirons, une à une, les affectations de $X$ qui ne couvrent pas entièrement $Y$ , c'est-à-dire, pour chaque sous-ensemble $Y' \subsetneq Y$, les solutions de $\texttt{range}(X, X, Y')$ :

$$\sum_{Y' \subsetneq Y} \#\texttt{range}(X, X, Y')$$

En effet, pour deux sous-ensembles $Y'_1 \neq Y'_2 \subsetneq Y$, l'ensemble des tuples autorisés $\mathcal{S}_{\texttt{range}(X,X,Y'_1)}$ et $\mathcal{S}_{\texttt{range}(X,X,Y'_2)}$ sont nécessairement disjoints : il existe une valeur $y_j \in Y$ tel que $y_j \in Y'_1$ et $y_j \notin Y'_2$ (ou bien $y_j \in Y'_2$ et $y_j \notin Y'_1$), alors $y_j$ doit être affectée à une variable de $X$ pour satisfaire $\texttt{range}(X, X, Y'_1)$ mais aucune des variables de $X$ doit être instanciée à $y_j$ pour satisfaire $\texttt{range}(X, X, Y'_2)$ (ou inversement). Une solution de $\texttt{range}(X, X, Y'_1)$ ne peut être solution de $\texttt{range}(X, X, Y'_2)$ et inversement.

Aucune solution n'est comptée deux fois dans $\sum_{Y' \subset Y} \#\texttt{range}(X, X, Y')$. On a alors :

$$\#\texttt{range}(X, X, Y) = \prod_{x_i \in X} d_i - \sum_{Y' \subsetneq Y} \#\texttt{range}(X, X, Y')$$

□

**Remarque 1.** *Proposition 2 peut être utilisée dans Proposition 1 :*

$$\#\mathit{range}(X, X', Y') = \prod_{x_i \in \overline{X'}} d_i \cdot \left( \prod_{x_i \in X'} d_i(Y') - \sum_{Y'' \subsetneq Y'} \#\mathit{range}(X', X', Y'') \right)$$

Cette formule est récursive et n'est pas utile en pratique. Dans la sous-section suivante, nous verrons qu'il s'agit d'un résultat préliminaire pour le calcul d'une estimation du nombre de solutions pour la contraintes `range`. Nous nous intéressons maintenant à la contrainte `roots`.

**Proposition 3.** *Soit $X' \subseteq X$ et $Y' \subseteq Y$. On note $\overline{X'}$, le complémentaire de $X'$ dans $X$ et $\overline{Y'}$ le complémentaire de $Y'$ dans $Y$. Le nombre de tuples autorisés par* $\mathit{roots}(X, X', Y')$ *est*

$$\#\mathit{roots}(X, X', Y') = \prod_{x_i \in X'} d_i(Y') \cdot \prod_{x_i \in \overline{X'}} d_i(\overline{Y'}) \tag{5}$$

*Pour rappel, $d_i(Y') = |D_i \cap Y'|$ et $d_i(\overline{Y'}) = |D_i \cap \overline{Y'}|$*

*Démonstration.* Pour satisfaire la contrainte $\texttt{roots}(X, X', Y')$, toutes les variables de $X'$ doivent prendre des valeurs dans $Y'$ et aucune valeur de $Y'$ doit être affectée aux variables de $\overline{X'}$, c'est-à-dire toutes les variables de $\overline{X'}$ doivent être affectées à des valeurs de $\overline{Y'}$ :

- $\prod_{x_i \in X'} d_i(Y')$ donne le nombre d'affectations possibles pour $X'$
- $\prod_{x_i \in \overline{X'}} d_i(\overline{Y'})$ donne le nombre d'affectations possibles pour $\overline{X'}$

□

### 3.2 Modèle probabiliste appliqué à `range` et `roots`

Dans cette sous-section, nous présentons un modèle probabiliste pour les contraintes de cardinalité en se basant sur les travaux de Erdős et Renyi [4]. L'idée est de probabiliser le domaine des variables. Ensuite, nous utiliserons ce modèle pour obtenir une estimation du nombre de solutions sur `range` et `roots`.

#### 3.2.1 Le modèle Erdős-Renyi appliqué aux CSP

Erdős et Renyi [4] ont mené des recherches sur les graphes aléatoires et ont étudié l'existence et le nombre de couplages parfaits dans ces structures aléatoires. Appliquée directement à notre problème, l'idée est de probabiliser le domaine de chaque variable de sorte que : pour tout $x_i \in X$ et pour tout $y_j \in Y$, l'évènement

$\{y_j \in D_i\}$ se produit avec une probabilité $p \in [0,1]$ et tous ces évènements sont **indépendants** :

$$\mathbb{P}(\{y_j \in D_i\}) = p \in [0,1] \tag{6}$$

Dans la suite, $\mathbb{E}(.)$ désigne l'espérance sous les hypothèses du modèle Erdős-Renyi

### 3.2.2 Le modèle Erdős-Renyi appliqué à la contrainte `range`

Nous étudions maintenant l'espérance du nombre de solutions d'une contrainte `range`. Dans le cas où chaque variable de $X$ et chaque valeur de $Y$ sont contraintes, l'espérance de $\#\texttt{range}(X,X,Y)$ est une fonction de $n, m$ et $p$ (pour rappel, $|X| = n$ et $|Y| = m$). Proposition 4 précise la forme de cette fonction.

**Proposition 4.** *Dans le cas où toutes les variables de $X$ et toutes les valeurs de $Y$ sont contraintes, il existe $a_{n,m} \in \mathbb{N}$ tel que :*

$$\mathbb{E}(\#\texttt{range}(X,X,Y)) = a_{n,m} \cdot p^n \tag{7}$$

*Démonstration.* Pour montrer ce résultat, nous raisonnons simplement par récurrence sur $|Y| = m$. Soit $|X| = n \in \mathbb{N}$,

**Initialisation.** Soit $Y = \{y\}$ un singleton. Dans ce cas particulier, une instance $\texttt{range}(X,X,Y)$ a un seul tuple autorisé si $y$ est dans tous les domaines $D_i$, et n'a aucun tuple autorisé sinon. Donc,

$$\begin{aligned}
&\mathbb{E}(\#\texttt{range}(X,X,\{y\})) \\
&= 0 * \mathbb{P}(\{\texttt{range}(X,X,\{y\}) \text{ n'a pas de solution}\}) \\
&\quad + 1 * \mathbb{P}(\{\texttt{range}(X,X,\{y\}) \text{ a une seule solution}\}) \\
&= \mathbb{P}(\{\texttt{range}(X,X,\{y\}) \text{ a une seule solution}\}) \\
&= \mathbb{P}(\{\forall x_i \in X, y \in D_i\}) \\
&= \prod_{i=1}^{n} \mathbb{P}(\{y \in D_i\}), \text{par hypothèse d'indépendance} \\
&= p^n
\end{aligned}$$

On a alors $a_{n,1} = 1$.

**Récurrence.** On suppose que la proposition est vraie pour tout $Y$, tel que $1 \leq |Y| = k \leq m-1, \exists a_{n,k} \in \mathbb{N}$,

$$\mathbb{E}(\#\texttt{range}(X,X,Y)) = a_{n,k} \cdot p^n$$

On veut prouver, sous ces hypothèses, pour un ensemble $Y$ avec $|Y| = m$, qu'il existe $a_{n,m}$, tel que $\mathbb{E}(\#\texttt{range}(X,X,Y)) = a_{n,m} \cdot p^n$

On a :

$$\begin{aligned}
&\mathbb{E}(\#\texttt{range}(X,X,Y)) \\
&= \mathbb{E}\left(\prod_{x_i \in X} d_i\right) - \sum_{Y' \subsetneq Y} \mathbb{E}(\#\texttt{range}(X,X,Y')) \\
&= \mathbb{E}\left(\prod_{x_i \in X} d_i\right) - \sum_{k=1}^{m-1} \binom{m}{k} a_{n,k} \cdot p^n \\
&= \prod_{x_i \in X} \mathbb{E}(d_i) - \sum_{k=1}^{m-1} \binom{m}{k} a_{n,k} \cdot p^n \\
&= (mp)^n - \sum_{k=1}^{m-1} \binom{m}{k} a_{n,k} \cdot p^n \\
&= \left(m^n - \sum_{k=1}^{m-1} \binom{m}{k} a_{n,k}\right) \cdot p^n
\end{aligned}$$

La première égalité vient de Proposition 2 et de la linéarité de l'opérateur $\mathbb{E}(.)$. La seconde vient de l'hypothèse de récurrence. Le troisième vient de l'hypothèse d'indépendance. Et le quatrième est dû au fait que $\forall x_i \in X, \mathbb{E}(d_i) = \sum_{j=1}^{m} \mathbb{P}(\{y_j \in D_i\}) = mp$.

Nous avons identifié le $a_{n,m}$ :

$$a_{n,m} = m^n - \sum_{k=1}^{m-1} \binom{m}{k} a_{n,k} \tag{8}$$

□

En remarquant que $\binom{m}{m} = 1$, on peut réécrire 8 comme suit :

$$m^n = \sum_{k=1}^{m} \binom{m}{k} a_{n,k} \tag{9}$$

Aussi, $\forall n \in \mathbb{N}^+, a_{n,1} = 1$. Ces coefficients sont référencés comme les "triangles of numbers" dans OEIS [1]. Les coefficients $a_{n,m}$ sont en fait le nombre de surjections possibles d'un ensemble de taille $n$ dans un autre ensemble de taille $m$ [2]. Il existe une formule non récursive pour calculer ces coefficients. Le résultat suivant est admis ici. L'intuition de la preuve est qu'il s'agit d'une application du principe inclusion-exclusion (voir section 1.9. The Twelvefold Way de [8]).

**Proposition 5.** *Pour $0 < m \leq n$,*

$$a_{n,m} = \sum_{k=0}^{m} (-1)^k \binom{m}{k} (m-k)^n \tag{10}$$

1. https://oeis.org/A019538
2. $a_{n,m}$ est en fait égale à $m! \cdot S_2(n,m)$, où $S_2(n,m)$ est le nombre de Stirling de deuxième espèce. Plus d'informations dans la section 1.9 de [8]

Proposition 6 est une propriété des "triangles of numbers" and sera utilisée pour faire des simplifications lors de prochains calculs.

**Proposition 6.**

$$a_{n,n} = n! \tag{11}$$

*Démonstration.* $a_{n,n}$ est le nombre de surjections possibles d'un ensemble de cardinalité n dans un ensemble de cardinalité n, soit le nombre de bijections dans ce cas particulier. □

Nous pouvons maintenant étendre Proposition 4 au cas où la contrainte `range` concerne uniquement des sous-ensembles $X' \subseteq X$ et $Y' \subseteq Y$ :

**Proposition 7.** *Soit $X' \subseteq X$ et $Y' \subseteq Y$. On note $|X'| = n'$ et $|Y'| = m'$.*

$$\mathbb{E}\left(\#\mathit{range}(X, X', Y')\right) = a_{n',m'} \cdot m^{n-n'} \cdot p^n \tag{12}$$

*Démonstration.* On déduit des propositions 1 et 4 et par hypothèse d'indépendance :

$$\begin{aligned}
&\mathbb{E}\left(\#\texttt{range}(X, X', Y')\right) \\
&= \mathbb{E}\left(\#\texttt{range}(X', X', Y')\right) \cdot \mathbb{E}\left(\prod_{x_i \in \overline{X'}} d_i\right) \\
&= a_{n',m'} \cdot p^{n'} \cdot \prod_{x_i \in \overline{X'}} \mathbb{E}\left(d_i\right) \\
&= a_{n',m'} \cdot p^{n'} \cdot (mp)^{n-n'} \\
&= a_{n',m'} \cdot m^{n-n'} \cdot p^n
\end{aligned}$$

□

### 3.2.3 Le modèle Erdős-Renyi appliqué à la contrainte `roots`

On étudie maintenant l'espérance du nombre de solutions sur une contrainte `roots`.

**Proposition 8.** *Soit $X' \subseteq X$ et $Y' \subseteq Y$. On note $|X'| = n'$ et $|Y'| = m'$.*

$$\mathbb{E}\left(\#\mathit{roots}(X, X', Y')\right) = m'^{n'} \cdot (m-m')^{n-n'} \cdot p^n \tag{13}$$

*Démonstration.* D'après Proposition 3 et par hypothèse d'indépendance :

$$\begin{aligned}
&\mathbb{E}\left(\#\texttt{roots}(X, X', Y')\right) \\
&= \mathbb{E}\left(\prod_{x_i \in X'} d_i(Y')\right) \cdot \mathbb{E}\left(\prod_{x_i \in \overline{X'}} d_i(\overline{Y'})\right) \\
&= \prod_{x_i \in X'} \mathbb{E}\left(d_i(Y')\right) \cdot \prod_{x_i \in \overline{X'}} \mathbb{E}\left(d_i(\overline{Y'})\right) \\
&= (m'p)^{n'} \cdot ((m-m')p)^{n-n'} \\
&= m'^{n'} \cdot (m-m')^{n-n'} \cdot p^n
\end{aligned}$$

□

**Remarque 2.** *Le paramètre p correspond à la densité d'arêtes dans le graphe des domaines. Pour utiliser l'estimateur en pratique, il nous faut donc évaluer p en divisant la somme de la taille des domaines par le nombre total d'arêtes possible : $n \cdot m$.*

## 4 Généralisation aux contraintes de cardinalité

Cette section présente une méthode systématique pour dénombrer les solutions pour la plupart des contraintes de cardinalité grâce à leur décomposition `range` et `roots`. Chaque sous-section rappelle d'abord les définitions des contraintes puis leur décomposition.

### 4.1 `alldifferent` [7]

**Définition 5.** *La contrainte $\mathit{alldifferent}(X)$ est satisfaite ssi chaque variable $x_i \in X$ est instanciée à une valeur de son domaine $D_i$ et chaque valeur de $y_j \in Y$ est affectée au plus une fois. Formellement :*

$$\mathcal{S}_{\mathit{alldifferent}(X)} = \{(v_1, \ldots, v_n) \in \mathcal{D} \mid \forall i, j \in \{1, ..., n\}, i \neq j \Leftrightarrow v_i \neq v_j\}$$

La décomposition de `alldifferent` en contrainte `range` est la suivante [3] :

$$\texttt{alldifferent}(X) \Leftrightarrow \texttt{range}(X, X, Y') \ \&\ |Y'| = n$$

À partir de cette décomposition, on en déduit une formule pour estimer le nombre de solutions sur `alldifferent`, d'après le modèle Erdős-Renyi.

**Proposition 9.**

$$\mathbb{E}\left(\#\mathit{alldifferent}(X)\right) = \frac{m!}{(m-n)!} \cdot p^n \tag{14}$$

*Démonstration.* D'après la décomposition de `alldifferent`

$$\#\texttt{alldifferent}(X) = \sum_{Y' \subseteq Y,\, |Y'|=n|} \#\texttt{range}(X, X, Y')$$

Alors,

$$\begin{aligned}
&\mathbb{E}\left(\#\texttt{alldifferent}(X)\right) \\
&= \sum_{Y' \subseteq Y,\, |Y'|=n} \mathbb{E}\left(\#\texttt{range}(X, X, Y')\right) \\
&= \binom{m}{n} \cdot a_{n,n} \cdot p^n \\
&= \frac{m!}{(m-n)!} \cdot p^n
\end{aligned}$$

□

**Remarque 3.** *Compter le nombre de solutions sur la contrainte* `alldifferent` *équivaut à compter le nombre de couplages couvrant $X$ dans le graphe des domaines [9]. L'application du modèle Erdős-Renyi sur ce problème et le calcul de l'espérance du nombre de couplages parfaits ont été étudiés dans [4] dans le cas de graphes bipartis équilibrés. La proposition 9 est en quelque sorte une extension de ce résultat pour les graphes bipartis non-équilibrés.*

### 4.2 `nvalue` [5]

**Définition 6.** *Soit $N \in \mathbb{N}$. La contrainte* `nvalue`$(X, N)$ *est satisfaite si il y a exactement $N$ valeurs de $Y$ assignées aux variables de $X$. Formellement :*

$$\mathcal{S}_{\texttt{nvalue}(X,N)} = \{(v_1, \ldots, v_n) \in \mathcal{D} |\ N = |\{y_j \in Y | \exists x_i \in X, v_i = y_j\}|\}$$

La décomposition de `nvalue` en contrainte `range` est la suivante [3] :

$$\texttt{nvalue}(X, N) \Leftrightarrow \texttt{range}(X, X, Y') \ \&\ |Y'| = N$$

À partir de cette décomposition, on en déduit une formule pour estimer le nombre de solutions sur `nvalue`, d'après le modèle Erdős-Renyi.

**Proposition 10.**

$$\mathbb{E}\left(\#\texttt{nvalue}(X, N)\right) = \binom{m}{N} \cdot a_{n,N} \cdot p^n \tag{15}$$

*Démonstration.* La preuve est la même que pour Proposition 9 □

### 4.3 `atmost_nvalue` et `atleast_nvalue`

À partir de Proposition 10, on peut déduire l'espérance du nombre de solutions pour `atmost_nvalue`et `atleast_nvalue`.

**Définition 7** (`atmost_nvalue` [2])**.** *Soit $N \in \mathbb{N}$. La contrainte* `atmost_nvalue`$(X, N)$ *est satisfaite si au plus $N$ valeurs de $Y$ sont affectées aux variables de $X$.*

**Définition 8** ( `atleast_nvalue` [2])**.** *Soit $N \in \mathbb{N}$. La contrainte* `atleast_nvalue`$(X, N)$ *est satisfaite si au moins $N$ valeurs de $Y$ sont affectées aux variables de $X$.*

On peut décomposer `atleast_nvalue` and `atmost_nvalue` directement avec la contrainte `nvalue` :

$$\texttt{atmost_nvalue}(X, N) \Leftrightarrow \texttt{nvalue}(X, k) \ \&\ k \leq N$$

$$\texttt{atleast_nvalue}(X, N) \Leftrightarrow \texttt{nvalue}(X, k) \ \&\ k \geq N$$

**Proposition 11.**

$$\mathbb{E}\left(\#\texttt{atmost_nvalue}(X, N)\right) = \sum_{k=1}^{N} \binom{m}{k} a_{n,k} \cdot p^n \tag{16}$$

$$\mathbb{E}\left(\#\texttt{atleast_nvalue}(X, N)\right) = \sum_{k=N}^{n} \binom{m}{k} a_{n,k} \cdot p^n \tag{17}$$

### 4.4 `disjoint` [1]

**Définition 9.** *Soit $X_1 \subset X$ et $X_2 \subset X$, tels que $X_1 \cap X_2 = \emptyset$,* `disjoint`$(X, X_1, X_2)$ *assure qu'aucune des variables de $X_1$ n'est affectée à la même valeur que l'une des variables de $X_2$. Formellement :*

$$\mathcal{S}_{\texttt{disjoint}(X,X_1,X_2)} = \{(v_1, \ldots, v_n) \in \mathcal{D} |\ \{v_i | x_i \in X_1\} \cap \{v_i | x_i \in X_2\} = \emptyset\}$$

Une décomposition de `disjoint` avec deux contraintes `range` est donnée par l'équivalence suivante [3] :

$$\begin{aligned}
&\texttt{disjoint}(X, X_1, X_2) \Leftrightarrow \\
&\qquad \texttt{range}(X, X_1, Y_1) \ \&\ \texttt{range}(X, X_2, Y_2) \\
&\qquad \&\ Y_1 \subset Y \ \&\ Y_2 \subset Y \ \&\ Y_1 \cap Y_2 = \emptyset
\end{aligned}$$

On en déduit une formule pour estimer le nombre de solutions sur `disjoint`, sous les hypothèses du modèle Erdős-Renyi.

**Proposition 12.** *Soit $X_1 \in X$ et $X_2 \in X$ avec $|X_1| = n_1$ and $|X_2| = n_2$ et $X_1 \cap X_2 = \emptyset$*

$$\mathbb{E}\left(\#\texttt{disjoint}(X, X_1, X_2)\right) = p^n \cdot m^{n-n_1-n_2} \cdot \sum_{k=1}^{min(n_1,m)} \sum_{l=1}^{min(n_2,m-k)} \binom{m}{k}\binom{m-k}{l} a_{n_1,k} a_{n_2,l}$$

*Démonstration.* D'après la décomposition de `disjoint`, on a :

$$\#\texttt{disjoint}(X, X_1, X_2) = \left(\prod_{x_i \in X \setminus \{X_1 \cup X_2\}} d_i\right) \cdot \left(\sum_{Y_1 \cap Y_2 = \emptyset} \#\texttt{range}(X_1, X_1, Y_1) \cdot \#\texttt{range}(X_2, X_2, Y_2)\right)$$

Le premier facteur du produit est le nombre total de combinaisons pour les variables non contraintes. Une fois que ces variables sont instanciées, on doit choisir deux sous-ensembles $Y_1$ et $Y_2$ tels que $Y_1 \cap Y_2 = \emptyset$ et calculer le nombre de solutions de $\texttt{range}(X_1, X_1, Y_1)$ et $\texttt{range}(X_2, X_2, Y_2)$. Pour deux paires différentes de sous-ensembles $(Y_1^A, Y_2^A)$ et $(Y_1^B, Y_2^B)$, l'ensemble des solutions de $\{\texttt{range}(X_1, X_1, Y_1^A) \& \texttt{range}(X_2, X_2, Y_2^A)\}$ et $\{\texttt{range}(X_1, X_1, Y_1^B) \& \texttt{range}(X_2, X_2, Y_2^B)\}$ sont disjoints. Nous ne comptons alors pas plusieurs fois une même solution.

On a aussi :

$$\mathbb{E}\left(\prod_{x_i \in X \setminus \{X_1 \cup X_2\}} d_i\right) = \prod_{x_i \in X \setminus \{X_1 \cup X_2\}} \mathbb{E}\left(d_i\right) = (mp)^{n-n_1-n_2}$$

Et,

$$\mathbb{E}\left(\sum_{Y_1 \cap Y_2 = \emptyset} \#\texttt{range}(X_1, X_1, Y_1) \cdot \#\texttt{range}(X_2, X_2, Y_2)\right) = \sum_{Y_1 \cap Y_2 = \emptyset} a_{n_1,|Y_1|} p^{n_1} \cdot a_{n_2,|Y_2|} p^{n_2},$$

par la Proposition 4 et hypothèse d'indépendance

Chaque paire $(Y_1, Y_2)$ avec une taille fixée à $|Y_1| = k$ et $|Y_2| = l$ aboutisse à une même formule d'estimation, alors nous choisissons d'abord un sous-ensemble de valeur $Y_1$ de taille $k \in \{1, \ldots, min(n_1, m)\}$ et ensuite un sous-ensemble de valeur $Y_2$ de taille $l \in \{1, \ldots, min(n_2, m-k)\}$ :

$$\sum_{Y_1 \cap Y_2 = \emptyset} a_{n_1,|Y_1|} \cdot a_{n_2,|Y_2|} = \sum_{k=1}^{min(n_1,m)} \sum_{l=1}^{min(n_2,m-k)} \binom{m}{k}\binom{m-k}{l} a_{n_1,k} a_{n_2,l}$$

En multipliant tous ces facteurs, nous déduisons la Proposition 12 □

### 4.5 `atmost` et `atleast`

**Définition 10** (`atmost`)**.** *Soit $y \in Y$ et $N \in \mathbb{N}$, la contrainte $\texttt{atmost}(X, y, N)$ est satisfaite si au plus $N$ variables sont instanciées à la valeur $y$*

$$\mathcal{S}_{\texttt{atmost}(X,y,N)} = \{(d_1, \ldots, d_n) | N \geq |\{x_i | d_i = y\}|\}$$

**Définition 11** (`atleast`)**.** *Soit $y \in Y$ et $N \in \mathbb{N}$, la contrainte $\texttt{atleast}(X, y, N)$ est satisfaite si au moins $N$ variables sont instanciées à la valeur $y$.*

$$\mathcal{S}_{\texttt{atleast}(X,y,N)} = \{(d_1, \ldots, d_n) | N \leq |\{x_i | d_i = y\}|\}$$

Les décompositions des contraintes `atmost` and `atleast` en contraintes `range` et `roots`sont données par les équivalences suivantes [3] :

$$\texttt{atmost}(X, y, N) \Leftrightarrow \texttt{roots}(X, X', \{y\}) \ \& \ |X'| \leq N$$

$$\texttt{atleast}(X, y, N) \Leftrightarrow \texttt{roots}(X, X', \{y\}) \ \& \ |X'| \geq N$$

**Proposition 13.**

$$\mathbb{E}\left(\#\texttt{atmost}(X, y, N)\right) = \sum_{k=1}^{N} \binom{n}{k} (m-1)^{n-k} \cdot p^n \quad (18)$$

$$\mathbb{E}\left(\#\texttt{atleast}(X, y, N)\right) = \sum_{k=N}^{n} \binom{n}{k} (m-1)^{n-k} \cdot p^n \quad (19)$$

*Démonstration.* Nous faisons la preuve seulement pour la contrainte `atmost`. On a :

$$\#\texttt{atmost}(X, y, N) = \sum_{X' \subseteq X, |X'| \leq N} \#\texttt{roots}(X, X', \{y\})$$

En effet, pour deux sous-ensembles différents $X_1' \neq X_2' \subseteq X$, l'ensemble des solutions de $\texttt{roots}(X, X_1', \{y\})$ et $\texttt{roots}(X, X_2', \{y\})$ ont une intersection vide. Aucune solution n'est comptée plusieurs fois. Et :

$$\begin{aligned}
&\mathbb{E}\left(\#\texttt{atmost}(X, y, N)\right) \\
&= \sum_{X' \subseteq X, |X'| \leq N} \mathbb{E}\left(\#\texttt{roots}(X, X', \{y\})\right) \\
&= \sum_{X' \subseteq X, |X'| \leq N} (m-1)^{n-|X'|} \cdot p^n, \text{ par la Proposition 8} \\
&= \sum_{k=1}^{N} \binom{n}{k} (m-1)^{n-k} \cdot p^n
\end{aligned}$$

□

**Remarque 4.** *Nous remarquons que beaucoup de ces formules nécessitent le calcul de coefficients binomiaux, de factorielles et des "triangles of numbers" $a_{n,m}$. Lors de l'utilisation d'heuristiques counting-based search, ces estimateurs doivent être calculés plusieurs fois au cours de la recherche. Nous proposons de pré-calculer tous ces coefficients au début de la résolution pour diminuer le temps de calcul. Le calcul des estimateurs pour* `alldifferent` *et* `nvalue` *est donc en temps constant, le calcul des estimateurs pour* `atmost_nvalue`*,* `atleast_nvalue`*,* `atmost`*,* `atleast` *est linéaire et le calcul de l'estimateur pour* `disjoint` *a une complexité quadratique.*

## 5 Conclusion

Dans cet article, nous avons proposé une méthode pour estimer le nombre de solutions des contraintes `range` et `roots` de manière probabiliste avec le modèle Erdős-Renyi. Nous avons montré que nous pouvons estimer le nombre de solutions pour de nombreuses contraintes de cardinalité en utilisant leur décomposition `range` et `roots`. Nous avons expliqué notre méthode pour les contraintes `alldifferent`, `disjoint`, `nvalue`, `atleast_nvalue`, `atmost_nvalue`, `atleast`, `atmost`.

Dans la poursuite de ce travail, nous pensons étendre cette méthode de dénombrement probabiliste à d'autres contraintes de cardinalité. Nous souhaitons également expérimenter l'utilisation de ces estimateurs au sein d'heuristiques counting-based search et de les comparer à d'autres stratégies, telles que `dom/wdeg`, `abs` ou bien `ibs`.

Les contraintes pour lesquelles la décomposition est une conjonction de plusieurs contraintes `range` et/ou `roots` non indépendantes (`gcc` par exemple) posent encore problème. En d'autres termes, si au moins deux contraintes `range` et/ou `roots` concernent des ensembles de variables qui ne sont pas disjoints, notre méthode doit être réadaptée.

## Références

[1] Nicolas Beldiceanu : Global constraints as graph properties on a structured network of elementary constraints of the same type. *In* Rina Dechter, éditeur : *Principles and Practice of Constraint Programming - CP 2000, 6th International Conference, Singapore, September 18-21, 2000, Proceedings*, volume 1894 de *Lecture Notes in Computer Science*, pages 52–66. Springer, 2000.

[2] Christian Bessière, Emmanuel Hebrard, Brahim Hnich, Zeynep Kiziltan et Toby Walsh : Filtering algorithms for the nvalue constraint. *In* Roman Barták et Michela Milano, éditeurs : *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Second International Conference, CPAIOR 2005, Prague, Czech Republic, May 30 - June 1, 2005, Proceedings*, volume 3524 de *Lecture Notes in Computer Science*, pages 79–93. Springer, 2005.

[3] Christian Bessiere, Emmanuel Hebrard, Brahim Hnich, Zeynep Kiziltan et Toby Walsh : Range and roots : Two common patterns for specifying and propagating counting and occurrence constraints. *Artif. Intell.*, 173(11):1054–1078, 2009.

[4] P. Erdos et A. Renyi : On random matrices. *Publication of the Mathematical Institute of the Hungarian Academy of Science*, 1963.

[5] François Pachet et Pierre Roy : Automatic generation of music programs. *In* Joxan Jaffar, éditeur : *Principles and Practice of Constraint Programming - CP'99, 5th International Conference, Alexandria, Virginia, USA, October 11-14, 1999, Proceedings*, volume 1713 de *Lecture Notes in Computer Science*, pages 331–345. Springer, 1999.

[6] Gilles Pesant, Claude-Guy Quimper et Alessandro Zanarini : Counting-based search : Branching heuristics for constraint satisfaction problems. *J. Artif. Intell. Res.*, 43:173–210, 2012.

[7] Jean-Charles Régin : A filtering algorithm for constraints of difference in csps. pages 362–367, 1994.

[8] Richard P. Stanley : *Enumerative Combinatorics : Volume 1.* Cambridge University Press, New York, NY, USA, 2nd édition, 2011.

[9] Willem Jan van Hoeve : The alldifferent constraint : A survey. *CoRR*, cs.PL/0105015, 2001.

# Qualité et diversité garanties dans les réseaux de fonctions de coût

**M. Ruffini[1] J. Vucinic[1] S. de Givry[1] G. Katsirelos[2] S. Barbe[3] T. Schiex[1]**

[1] INRA MIA Toulouse, UR 875, 31320 Auzeville-Tolosane, France
[2] INRA MIA-Paris, UMR 518 AgroParisTech, 75231 Paris, France
[3] INSA LISBP, UMR 792, 31400 Toulouse, France

**Résumé**

Il est en général impossible de modéliser toutes les informations pertinentes dans un modèle graphique. Dans ce contexte, nous nous intéressons au problème de production d'un ensemble divers de solutions de bonne qualité, avec des garanties à la fois sur la qualité, et sur la diversité des solutions. Nous nous appuyons sur l'utilisation d'automates pondérés décomposés en fonctions d'arité bornée, et exploitons les propriétés des contraintes de dissimilarité pour améliorer leur représentation. Nous montrons que cette approche est capable de produire des ensembles de solutions diverses sur des réseaux bayésiens de grande taille, qu'elle permet d'énumérer des modes locaux, et qu'elle fournit des designs améliorés dans le cadre du Design Computationnel de Protéines.

**Abstract**

Because it's usually not possible to capture all relevant information in a graphical model. In this context, motivated by a Computational Protein Design application, we consider the general problem of producing a diverse set of high quality solutions, with guarantees both on solution quality and diversity. We use weighted automata decomposed in functions of bounded arity and exploit the semantics of distance constraint to improve the representation. We show that this approach is able to solve this problem on large Bayesian networks, that it has the capacity to enumerate local modes and to provide improved designs in the context of Computational Protein Design.

## 1 Introduction

Les modèles graphiques (GMs) permettent la représentation concise de distributions multivariées de grande dimension, via une factorisation de la distribution en facteurs de petite arité. Dans le cas de variables discrètes, ces modèles englobent un ensemble de représentations de distributions (ou fonctions) jointes discrètes, qui peuvent être des fonctions booléennes, comme dans SAT ou les problèmes de satisfaction de contraintes (CSP) ; des fonctions de coût comme dans MaxSAT ou les réseaux de fonctions de coût (CFN, *cost function networks*) ; ou des distributions de probabilités dans les champs de Markov (MRF, *Markov Random Fields*) ou les réseaux bayésiens (BN, *Bayesian Networks*). L'optimisation est une des requêtes les plus usuelles sur ces modèles et est NP-complète (satisfaction dans le cas SAT/CSP). Elle correspond à la recherche d'une affectation des variables qui optimise la fonction jointe : elle définit un modèle dans SAT, une solution dans les CSP, une solution optimale dans les CFN, un *maximum a posteriori* (MAP/MRF) ou une explication de probabilité maximale (Maximum probability explanation MPE/BN).

En général, la fonction jointe n'est pas une représentation parfaite du problème réel, et les requêtes d'optimisation ne permettent pas d'identifier une unique affectation satisfaisante. Pour SAT et CSP, toutes les solutions sont équivalentes mais il peut exister des critères supplémentaires non représentés dans le modèle. Même quand un critère existe, il est fréquent que les solutions optimales ne correspondent pas aux meilleures solutions en pratique. Ce cas de figure peut par exemple être rencontré quand le modèle graphique a été appris à partir d'un jeu de données : à cause de l'échantillonnage fini, le modèle appris attribue souvent une probabilité sous-optimale aux solutions extraites de l'ensemble apprentissage/test ; ou lorsque le modèle graphique est une estimation d'un modèle difficile à représenter. Alors, produire un ensemble de solutions diverses de coût faible paraît être une bonne idée.

Dans cet article, nous nous intéressons particulièrement au problème de design computationnel de protéines (CPD, *Computational Protein Design* [33]). Une protéine est une chaîne de molécules simples appelées acides aminés. Les acides aminés sont composés d'un noyau commun fixe formé d'atomes de carbone, d'azote et d'oxygène, et d'une chaîne latérale variable et flexible, aux propriétés chimiques variées. La nature de la chaîne latérale définit le type d'acide aminé. Dans la nature, il existe 20 chaînes latérales qui définissent 20 acides aminés. Dans une protéine, les parties invariables des acides aminés sont liées entre elles en une chaîne pour former le *squelette* de la protéine. La protéine se *replie* en une structure 3D qui est déterminée par la séquence d'acides aminés. Une chaîne latérale peut posséder jusqu'à 4 axes de rotations par rapport au squelette. La structure 3D définie par le squelette et l'ensemble des positions des chaînes latérales est appelée *conformation* ; elle détermine l'activité chimique et la fonction de la protéine. Le problème de design computationnel de protéine à squelette fixe consiste, à partir d'un squelette 3D rigide, à déterminer la séquence d'acides aminés qui se repliera sur ce squelette. Pour résoudre ce problème, l'approche usuelle consiste à utiliser une fonction d'énergie qui estime l'énergie de la protéine en fonction de la nature et de l'orientation de toutes ses chaînes latérales. On peut alors identifier la nature et la conformation des acides aminés qui correspondent à l'énergie minimale (ou stabilité maximale). La fonction d'énergie étant supposée décomposable en termes binaires, et le squelette étant supposé rigide, l'énergie de la protéine peut être décrite au moyen d'un modèle graphique binaire avec une variable par position dans la chaîne protéique, et un domaine qui décrit l'ensemble des types de chaînes latérales possibles, avec chacun un ensemble discret d'orientations déterminées a priori (appelée bibliothèque de rotamères). Une affectation d'énergie minimale définit une séquence d'acides aminés qui peut être synthétisée et testée. Les protéines régissent l'essentiel du fonctionnement des cellules chez les humains, animaux, plantes et microbes, et les protéines nouvellement conçues ont un grand potentiel d'applications en médecine, chimie verte...

Pour résoudre les réseaux de fonctions booléennes, la programmation par contrainte et les solveurs SAT combinent recherche arborescente, propagation de contraintes (ou filtrage par cohérence locale) avec des heuristiques de choix de variables et autres techniques pour identifier des solutions ou prouver qu'il n'en existe pas. Cette approche a été étendue à la recherche de solutions dans les réseaux de fonctions de coût et à la preuve de leur optimalité. Il a été montré récemment qu'un algorithme qui maintient des cohérences locales souples (MSLC, *Maintaining Soft Local Consistencies*) au cours de la recherche arborescente est souvent plus efficace que d'autres approches exactes pour trouver les affectations optimales dans les modèles graphiques [18].

Sur le problème de design à squelette rigide, cette approche est plus efficace que les solveurs de programmations linéaire et quadratique en nombres entiers, les solveurs MaxSAT, les solveurs MAP/MRF, ainsi que les algorithmes exacts dédiés [1]. Cette approche a également mis en lumière les limitations de l'implémentation du recuit simulé dédié au design de protéines, et a récemment permis de produire une protéine hyperstable auto-assemblée [24].

En design de protéines, les fonctions d'énergie décomposables ne permettent que d'estimer un modèle de mécanique quantique qui est hors de portée des supercalculateurs les plus puissants. Cela rend le processus de conception de protéines peu fiable : une approche classique consiste à produire plusieurs protéines qui devront ensuite être testées expérimentalement. Idéalement, cette bibliothèque est constituée de séquences diverses de faible énergie, l'idée étant que la diversité augmente les chances de trouver une protéine fonctionnelle. La diversité peut être définie par la distance de Hamming, ou une diversité "chimique" estimée au moyen de matrices de dissimilarité existantes pour les protéines. Comme leurs applications sont importantes, les séquences protéiques peuvent faire l'objet de brevets : une nouvelle séquence devra satisfaire une contrainte de distance de Hamming par rapport aux séquences brevetées.

Dans cet article, nous nous intéressons à la production d'un ensemble de bonnes solutions satisfaisant des contraintes de distance entre elles. Parce que la diversité est une exigence usuelle, de nombreuses approches de ce problème existent mais aucune ne correspond à nos besoins. Après une présentation rapide des approches existantes, nous observons qu'il est difficile d'appliquer une cohérence locale souple sur de telles exigences de distance, même pour les cohérences locales les plus faibles. Nous représentons les contraintes de distance à l'aide de fonctions globales basées sur des automates et optimisons la représentation en exploitant la sémantique de la fonction de distance. En raison de la variété et de la complexité des cohérences d'arc souples, nous avons testé mais abandonné l'optimisation monolithique usuelle des fonctions globales basées sur des automates pour une une décomposition en un ensemble non homogène de fonctions de coût d'arité bornée et exploré l'utilisation des représentations duale/cachée des CSP sur des modèles graphiques. Nous évaluons notre approche sur un ensemble de problèmes comprenant des réseaux bayésiens usuels et des instances de CPD et la comparons, lorsque c'est possible, à un algorithme récent et proche, calculant des $M$-modes [11].

## 2 Préalables

Nous utilisons des capitales pour les variables, des minuscules pour les valeurs et des caractères gras pour les ensembles, séquences ou $n$-uplets.

**Definition 1.** *Un réseau de fonctions de coût (CFN) est une paire* $(\mathbf{X}, \mathbf{W})$ *où* $\mathbf{X} = \{X_1, \ldots, X_n\}$ *est un ensemble de* $n$ *variables et* $\mathbf{W}$ *un ensemble de fonctions. Chaque variable* $X_i \in \mathbf{X}$ *a un domaine fini* $\mathbf{D}_i$ *de valeurs possibles. Pour un ensemble de variables* $\mathbf{S} \subseteq \mathbf{X}$, $\mathbf{D_S}$ *désigne le produit Cartésien des domaines des variables de* $\mathbf{S}$. *Pour un* $n$*-uplet de valeurs* $\mathbf{t}$, $\mathbf{t}[\mathbf{S}]$ *désigne la projection de* $\mathbf{t}$ *sur* $\mathbf{S}$. *Une fonction* $w_\mathbf{S} \in \mathbf{W}$, *avec une portée* $\mathbf{S} \subseteq \mathbf{X}$, *est une function* $w_\mathbf{S}$ *qui associe un coût entier inférieur ou égal à* $k \in \mathbb{Z} \cup \{\infty\}$ *à chaque* $n$*-uplet de* $\mathbf{D_S}$. $k$ *est associé aux* $n$*-uplets interdits. Un CFN est en Forme Normale si tout* $w_\mathbf{S} \in \mathbf{W}, \mathbf{S} \neq \varnothing$ *est non négative et* $\mathbf{W}$ *contient une fonction constante (de portée vide)* $w_\varnothing$ *de valeur arbitraire dans* $\mathbb{Z}$.

Un CFN $N = (\mathbf{X}, \mathbf{W})$ définit une fonction de coût entière jointe $J_N(\mathbf{X}) = \bigoplus_{w_\mathbf{S} \in \mathbf{W}} w_\mathbf{S}$, où $a \oplus b = \min(k, a+b)$ est l'addition bornée à $k$. Dans le reste du document, nous supposons que tous les CFN sont sous forme normale (calculable en temps linéaire). Dans ce cas, $w_\emptyset$ est un minorant de $J_N(\mathbf{X})$. Une fonction $w_\mathbf{S}$ dont l'image est égale à $\{0, k\}$ est une *contrainte*. Une affectation de $\mathbf{X}$ dont le coût est strictement inférieur à $k$ est *faisable*. Le problème de satisfaction de contraintes pondérées (WCSP) consiste à trouver une affectation faisable de $\mathbf{X}$ minimisant $J_N(\mathbf{X})$. Le WCSP est décision NP-complet. Si $k = 1$, le WCSP est le CSP. Si les variables sont booléennes et les contraintes exprimées en clauses, le WCSP est SAT. Un champ aléatoire de Markov additif discret peut être approché par un CFN avec une précision arbitraire : toutes les énergies peuvent être décalées, multipliées et arrondies à l'entier le plus proche (représentation décimale fixe). Le WCSP couvre donc les problèmes MAP/MRF et MPE/BN. Comme décrit dans [1], le problème de CPD se réduit à un WCSP binaire où il y a une variable par position mutable dans la séquence de la protéine considérée. Chaque valeur contient une paire qui décrit la nature de l'acide aminé utilisé et sa conformation (typiquement quelques centaines de valeurs par domaine). L'énergie de la protéine est décrite par des fonctions de coût unaires et binaires formant un graphe presque complet. Aucune propriété garantissant que les problèmes de CPD sont sous-modulaires n'est connue (et aucune des instances simples que nous avons testées n'a satisfait à un test de sous-modularité permutée [31]). Les problèmes typiques de CPD impliquent moins de 100 acides aminés ou variables mutables.

L'une des techniques les plus efficaces pour résoudre le WCSP consiste à maintenir une cohérence locale souple (MCLS) pendant une recherche en Branch and Bound [18, 12, 3]. Ces algorithmes explorent un arbre (généralement) binaire dans lequel un CFN est associé à chaque nœud. La racine est le réseau $N$ à résoudre. Pour créer une branche, une variable non affectée $X_i$ et une valeur $r$ de son domaine sont sélectionnées. Sur la branche de gauche, $X_i$ est affectée à $r$ et à droite, $r$ est supprimée des valeurs possibles pour $X_i$. Pour éviter d'explorer l'arbre entier, un minorant $lb$ est calculé. Si $lb$ est supérieur ou égal au coût $k$, la branche est élaguée. Si une feuille est atteinte, elle décrit une solution (faisable) et le majorant $k$ est mis à jour (une solution améliorante est cherchée). En pratique, cette combinaison comporte deux composants essentiels : la méthode de sélection de la variable de branchement, qui affecte en priorité les variables qui entraînent un élagage puissant et l'utilisation de $w_\varnothing$ comme minorant, après l'application de cohérences locales souples [12].

Les cohérences locales souples (SLC) sont des propriétés qui peuvent être établies sur un CFN $(\mathbf{X}, \mathbf{W})$ pour le transformer en un CFN équivalent $(\mathbf{X}, \mathbf{W}')$ qui satisfait la propriété SLC appliquée, qui a un minorant $w_\varnothing$ plus grand et des domaines éventuellement réduits : toute valeur pour laquelle on peut facilement établir qu'elle mènera à des affectation de coût $k$ peut être supprimée sans risque. Il existe différentes SLC qui offrent des minorants de plus en plus précis : cohérences de nœuds, $\emptyset$-inverse, directionnelle, d'arc, existentielle, virtuelle et optimale [30, 12]. À l'exception des trois premières, elles se réduisent toutes à la cohérence d'arc [28] lorsque $k = 1$ (CSP). Dans le cas MRF, les cohérence locales peuvent être vues comme des versions incrémentales des algorithmes de *message passing* (comme TRWS [20]), qui traitent en plus les informations déterministes (coût $k$) comme la cohérence locale le ferait en programmation par contraintes. Cette dernière propriété est cruciale notamment dans le cadre d'une recherche en Branch and Bound : chaque fois que le majorant $k$ est mis à jour parce qu'une nouvelle solution a été trouvée, la quantité d'information déterministe est plus importante, et les coupes plus nombreuses dans l'arbre de recherche. Comme les CFN ne font intervenir que des coûts entiers, les cohérences locales existantes peuvent être établies en temps polynômial en la taille des tables utilisées pour représenter les fonctions de coût. Cependant, si des représentations compactes des fonctions globales sont utilisées, établir les cohérences peut devenir NP-difficile.

Motivés par le problème de design de protéines, nous considérons des semi-métriques définies par la somme de dissimilarités pour chaque variable, données par une matrice symétrique positive à diagonale nulle $D$. Pour deux affectations $\mathbf{t_S}$ et $\mathbf{t'_S}$ d'un ensemble de variables $\mathbf{S}$, la dissimilarité $d(\mathbf{t_S}, \mathbf{t'_S}) = \sum_{X_i \in \mathbf{S}} D(\mathbf{t_S}[X_i], \mathbf{t'_S}[X_i])$

est bien une semi-métrique. La distance de Hamming est définie par $H(i,j) = \mathbb{1}(i \neq j)$. Notons qu'une matrice de similarité $S$ pour les protéines peut être transformée en une matrice de dissimilarité comme suit : $D(i,j) = \frac{1}{2}(S(i,i) + S(j,j)) - S(i,j)$.

**Definition 2.** *Pour un ensemble de solutions* $\mathbf{Z}$*, sa dissimilarité moyenne est est définie par* $\bar{d}(\mathbf{Z}) = \frac{1}{|\mathbf{Z}|}\sum_{\mathbf{t}\neq\mathbf{t}'\in\mathbf{Z}} d(\mathbf{t},\mathbf{t}')$ *et sa dissimilarité minimale par* $\check{d} = \min_{\mathbf{t}\neq\mathbf{t}'\in\mathbf{Z}} d(\mathbf{t},\mathbf{t}')$.

**Definition 3.** *Soient* $\mathbf{S},\mathbf{S}' \subseteq \mathbf{X}, |\mathbf{S}| = |\mathbf{S}'|$*. Soient* $D$ *une matrice de dissimilarité et* $\delta$ *un minorant de diversité. La fonction de coût globale* $\textsc{Dist}(\mathbf{S},\mathbf{S}',D,\delta)$ *est égale à* 0 *si* $sign(\delta).d(S,S') \geq \delta$ *et* $k$ *sinon.*

Soient $N = (\mathbf{X},\mathbf{W})$ un CFN, $D$ une matrice de dissimilarité, $M$ un entier et $\delta$ un seuil de diversité. Le problème $\textsc{DiverseSet}(N,D,M,\delta)$ est celui de produire un ensemble $\mathbf{Z}$ de $M$ solutions de $N$ tel que $\forall \mathbf{t} \neq \mathbf{t}' \in \mathbf{Z}$, $\textsc{Dist}(\mathbf{t},\mathbf{t}',D,\delta) = 0$ et $\sum_{\mathbf{t}\in\mathbf{Z}} J_N(\mathbf{t})$ soit minimale. Ce problème peut être très difficile à résoudre même sur de petites instances, aussi nous considérons le problème proche $\textsc{DiverseSeq}(N,D,M,\delta)$ qui consiste à trouver une liste $\mathbf{Z}$ de $M$ solutions de $N$ telle que pour tout $1 \leq i \leq M$, $\mathbf{Z}[i]$ est telle que $\forall j < i, \textsc{Dist}(\mathbf{Z}[i],\mathbf{Z}[j],D,\delta) = 0$ et de coût minimal.

Pour un CFN $N$ à $n$ variables, résoudre $\textsc{DiverseSet}$ nécessite d'affecter simultanément $nM$ variables. Ce problème peut être résolu en créant $M$ copies de $N$ avec des ensembles de variables $\mathbf{X}^1 \ldots \mathbf{X}^M$ et en ajoutant $\frac{M.M-1}{2}$ contraintes $\textsc{Dist}(\mathbf{X}^i,\mathbf{X}^j,D,\delta)$ pour tous $1 \leq i < j \leq M$ (si $k < \infty$, toutes les occurrences de $k$ doivent être remplacées par $M(k-1)+1$).

A la place, nous nous attaquons au problème glouton $\textsc{DiverseSeq}$ en affectant itérativement $n$ variables $M$ fois, ce qui peut être exponentiellement plus rapide, vu la NP-difficulté du WCSP. Étant donné un ensemble de solutions $\mathbf{Z}$ déjà déterminé, définissons la fonction de coût globale $\textsc{Div}_{\min}(\mathbf{X},\mathbf{Z},D,\delta) = \bigoplus_{\mathbf{t}\in\mathbf{Z}} \textsc{Dist}(\mathbf{X},\mathbf{t},D,\delta)$. Le CFN $(\mathbf{X},\mathbf{W}\cup\{\textsc{Div}_{\min}(\mathbf{X},\mathbf{Z},D,\delta)\})$ est résolu à plusieurs reprises, avec $\mathbf{Z} = \emptyset$ au début, puis en ajoutant la solution trouvée à $\mathbf{Z}$ itérativement jusqu'à atteindre $|\mathbf{Z}| = M$, ou jusqu'à ce qu'il n'existe plus de solution. Comme les CFN résolus sont de plus en plus contraints, le coût des solutions successives est croissant. Si des solutions $\mathbf{t}$ (brevets) sont données a priori, elles peuvent être prises en compte en ajoutant les contraintes correspondantes $\textsc{Dist}(\mathbf{X},\mathbf{t},D,\delta)$ avant la première résolution.

## 3 Travaux connexes

Dans le cas des fonctions booléennes, [16] considère l'optimisation de $M$ ou $\delta$ en utilisant la dissimilarité moyenne ou minimale. Les auteurs prouvent que l'application de la cohérence d'arc sur une contrainte de dissimilarité moyenne $\bar{d}$ suffisante est polynomiale mais NP-complete pour la dissimilarité minimale $\check{d}$ et évaluent un algorithme de production incrémentale d'un ensemble maximisant $\bar{d}$. [17] et [15] ont abordé les mêmes problèmes en utilisant des contraintes globales et de la compilation de connaissances. [26] a récemment proposé une approche de type COP (Constraint Optimization Problem) pour fournir des solutions diverses de haute qualité, mais au détriment de la diversité.

La production de solutions diverses a également été explorée dans les modèles graphiques stochastiques discrets (principalement les MRF). [5] exploite le fait que la relaxation Lagrangienne des contraintes de dissimilarité n'ajoute que des fonctions de coût unaires, préservant ainsi la sous-modularité. Cependant, l'écart de dualité est non nul, même pour les dissimilarités simples et une méthode exacte disponible uniquement pour les problèmes sous-modulaires. Ceci a été étendu dans [27] à l'aide de fonctions d'arité élevée afin d'optimiser approximativement un compromis entre diversité et qualité. Plus récemment, [19] a résolu le problème $\textsc{DiverseSet}$, mais en utilisant des techniques d'optimisation qui ne fournissent aucune garantie.

Aucune de ces approches ne fournit simultanément de garantie de qualité et de diversité. L'approche la plus similaire à la notre est celle de [8] qui considère le problème de production incrémentale de l'ensemble des $M$ meilleurs $\delta$-modes de la distribution jointe $J_N(X)$.

**Definition 4** ([11])**.** *Une solution* $\mathbf{t}$ *est un* $\delta$*-mode lorsqu'il n'existe aucune solution meilleure que* $\mathbf{t}$ *dans la boule de Hamming de rayon* $\delta$ *centrée en* $\mathbf{t}$ *(*$\mathbf{t}$ *est donc un minimum local).*

Un algorithme de programmation dynamique, combiné à A* et une décomposition arborescente a été proposé[8, 9, 10, 11], pour résoudre le problème de façon exact, uniquement avec la distance de Hamming. Du point de vue de l'optimisation, ils exploitent des minorants MAP de coût exponentiel qui contraignent la recherche à un ordre de variable statique. Les solveurs SAT/CP modernes ont montré l'importance cruciale d'heuristiques adaptatives telles que VSIDS pour SAT [23] ou *weighted degree* pour (W)CSP [7].

Cette approche offre cependant une garantie de diversité [11] : un $\delta$-mode sera toujours *strictement* à plus de $\delta$ d'un autre $\delta$-mode et sera produit en résolvant le problème $\textsc{DiverseSeq}$.

**Theorem 1.** *Soient* $N$ *et* $\delta$ *donnés et* $H$ *la matrice de dissimilarité de Hamming, pour tout* $\delta$*-mode* $\mathbf{t}$*, il existe une valeur* $M'$ *telle que la solution de* $\textsc{DiverseSeq}(N,H,M',\delta+1)$ *contienne* $\mathbf{t}$.

*Preuve.* Si un $\delta$-mode $\mathbf{t}$ n'est pas dans la solution de $\textsc{DiverseSeq}(N, H, M', \delta + 1)$, cela doit être dû aux contraintes $\textsc{Dist}$. Considérons l'itération $i$ qui interdit $\mathbf{t}$ pour la première fois : une solution de coût inférieur au coût de $\mathbf{t}$ a été produite (sinon $\mathbf{t}$ aurait été produite) mais cette solution est à moins de $\delta + 1$ de $\mathbf{t}$ (puisque $\mathbf{t}$ est interdit) et $\mathbf{t}$ n'est pas un $\delta$-mode. □

Pour un $M$ suffisamment grand, la séquence $\mathbf{Z}$ solution de $\textsc{DiverseSeq}(N, H, M', \delta + 1)$ contiendra donc tous les $\delta$-modes et éventuellement d'autres solutions. Il n'est pas difficile de séparer les modes des non-modes :

**Theorem 2.** *Toute affectation* $\mathbf{t}$ *d'un CFN* $N = (\mathbf{X}, \mathbf{W})$ *est un* $\delta$*-mode ssi il s'agit d'une solution optimale du CFN* $(X, W \cup \{\textsc{Dist}(\mathbf{X}, \mathbf{t}, H, -\delta\}))$*, un problème qui est dans P pour* $\delta$ *borné.*

*Preuve.* La fonction $\textsc{Dist}(\mathbf{X}, \mathbf{t}, H, -\delta\})$ restreint $\mathbf{X}$ à être à moins de $\delta$ de $\mathbf{t}$. Si $\mathbf{t}$ est une solution optimale de $(X, W \cup \{\textsc{Dist}(\mathbf{X}, \mathbf{t}, H, -\delta\}))$, il n'y a pas de meilleure affectation que $\mathbf{t}$ dans la boule de Hamming de rayon $\delta$ et $\mathbf{t}$ est un $\delta$-mode. Pour $\delta$ borné, un CFN avec $n$ variables et au maximum $d$ valeurs par domaine, il y a $O((nd)^\delta)$ n-uplets dans la boule de Hamming et le problème de vérifier si $\mathbf{t}$ est optimal est dans P. □

## 4 Cohérences locales souples et $\textsc{Div}_{\min}$

Pour résoudre itérativement le CFN $(X, W \cup \{\textsc{Div}_{\min}(X, Z_{\min}, D, \delta)\})$ en utilisant une approche MSLC, il faut établir des SLC sur $\textsc{Div}_{\min}$. Dans le cas CSP, [16] a montré qu'établir la cohérence d'arc (AC) sur $\textsc{Div}_{\min}$ était NP-complet. Puisque la cohérence d'arc souple et ses variantes sont des généralisations de AC dans les CSP, l'établir sur $\textsc{Div}_{\min}$ est aussi NP-complet. Cependant, il existe des SLC plus faibles qu'AC dans les WCSP, comme la cohérence $\emptyset$-inverse [34], qu'on pourrait espérer établir efficacement. Une fonction de coût $w_{\mathbf{S}}$ est dite $\emptyset$-inverse cohérente si son coût minimum est 0. Pour établir la cohérence $\emptyset$-inverse, on soustrait à $w_S$ son minimum et on l'ajoute à $w_\emptyset$. Cette transformation qui préserve l'équivalence (EPT) est appelée reparamétrisation dans les MRF.

**Theorem 3.** *La cohérence* $\emptyset$*-inverse est NP-difficile à établir sur* $\textsc{Div}_{\min}(\mathbf{X}, \mathbf{Z}, D, \delta)$.

*Proof.* Vérifier AC sur $\textsc{Div}_{\min}$ est NP-complet. Par définition de AC, $\textsc{Div}_{\min}$ est AC ssi pour tout $X_i \in \mathbf{X}$ et pour tout $r \in D_i$, il existe une affectation de $\mathbf{X} - \{X_i\}$ de coût 0 ou, de façon équivalente, si le domaine de $X_i$ est réduit à $\{r\}$, $\textsc{Div}_{\min}$ est $\emptyset$-inverse cohérent. Il est donc possible de réduire AC à un nombre d'appels linéaires à un oracle de cohérence $\emptyset$-inverse. □

Pour cette raison, les cohérences locales souples ne seront pas établies sur la contrainte $\textsc{Div}_{\min}$ directement, mais plutôt sur sa décomposition en fonctions $\textsc{Dist}(\mathbf{X}, \mathbf{t}, D, \delta)$ sur lesquelles elles peuvent être établies en temps polynômial.

## 5 Avec les automates

Étant donnés un CSP et un ordre sur ses variables, une contrainte est définie par son ensemble fini de $n$-uplets autorisés. Cet ensemble définit un langage régulier qui peut être encodé par un automate [25] et AC peut être établie en temps linéaire en la taille de l'automate. Les contraintes $\textsc{Div}_{\min}$ et $\textsc{Dist}$ peuvent donc être encodées par un automate. Pour $\textsc{Dist}$, il est établi qu'un automate compact existe [25]. La NP-complétude de $\textsc{Div}_{\min}$ signifie qu'il n'existe pas d'automate compact pour $\textsc{Div}_{\min}$ (sauf si P=NP). Pour un CFN, les SLC transforment la contrainte $\textsc{Dist}$ en fonctions qui font intervenir des coûts autres que 0 et $k$. Les SLC peuvent toujours être établies avec un automate *pondéré* [22] qui définit un langage des $n$-uplets autorisés avec leur coût associé.

Un automate pondéré encode la contrainte $\textsc{Dist}(\mathbf{X}, \mathbf{t}, D, \delta)$ avec :

- $(\delta + 1).(n + 1)$ états $s_i^d$ qui représentent le fait que les valeurs de $X_1 \dots X_i$ sont à distance $d$ de $\mathbf{t}[X_1 \dots X_i]$ (ou $\geq \delta$ pour les états $s_i^\delta$).
- pour les valeurs $r$ de $X_i$, il y a une transition de coût 0 de $s_i^d$ vers $s_{i+1}^{\min(d+D(r,t[i+1]),\delta)}$.
- l'état initial est $s_0^0$ et l'état acceptant est $s_n^\delta$

Cet automate pondéré contient $O(n.(\delta + 1).d)$ transitions. Une affectation $\mathbf{t}'$ de $\mathbf{X}$ est acceptée par cet automate ssi $d(\mathbf{t}', \mathbf{t}) \geq \delta$.

Les SLC peuvent être établies en temps polynômial en la taille de l'automate grâce à des algorithmes de flot minimal ou de programmation dynamique [22]. Par ailleurs, comme dans le cas CSP [6], les SLC peuvent être établies sur un automate pondéré représentant une fonction de coût en décomposant la fonction globale en une séquence de fonctions de coûts ternaires [2]. Cet objectif peut être atteint par l'ajout de $n+1$ variables états $Q_i, 0 \leq i \leq n$ et $n$ fonctions ternaires $w^A_{\{Q_i, X_{i+1}, Q_{i+1}\}}$ telles que $w^A_{\{Q_i, X_{i+1}, Q_{i+1}\}} = c$ ssi il existe une transition de l'état $Q_i$ vers l'état $Q_{i+1}$ avec la valeur $X_{i+1}$ et le poids $c$ dans l'automate. Les variables $Q_0$ et $Q_n$ ont des domaines réduits contenant uniquement les états initiaux et terminaux respectivement.

Dans le cas CSP, établir AC sur la décomposition est équivalent à l'établir sur la contrainte globale originale. Ceci permet d'établir AC sur des fonctions de coût globales avec un automate compact sans nouvel algorithme. Pour les CFN cependant, établir une SLC sur la décomposition peut être plus faible que l'établir

sur la fonction globale. Pour retrouver la propriété d'équivalence, il faut que l'ordre des variables utilisé pour construire l'automate soit compatible avec l'ordre utilisé pour la cohérence d'arc directionnelle [2]. Hors, les contraintes $\text{Div}_{\min}$ et Dist sont insensibles à l'ordre des variables, et la condition d'ordonnancement de [2] peut toujours être satisfaite. Étant donné la diversité des SLC définies sur les CFN, la décomposition semble être une approche intéressante pour encoder Dist et $\text{Div}_{\min}$.

Dans le cas de la contrainte Dist, chaque variable état $Q_i$ a $(\delta+1).(n+1)$ valeurs et la table de coûts de la fonction $w^A$ est de taille $(\delta+1)^2.(n+1)^2.d_i$, où $d$ est la taille du domaine de $X_i$. La contrainte $\text{Div}_{\min}$ nécessiterait quant à elle une table de taille $(\delta+1)^{2M}.(n+1)^2.d$. Pour accélérer l'établissement des SLC, nous exploitons les propriétés de Dist et $D$ pour réduire cette complexité.

## 6 Compression de l'encodage de Dist

A partir d'une large collection de jeux de données, il a été déterminé que la SLC qui semble la plus efficace (compromis temps de calcul/précision du minorant) empiriquement est EDAC (Existential Directional Arc Consistency) [14, 29]. EDAC a une complexité en $O(ed^a.\max(nd,k))$ pour $e$ fonctions de coût d'arité $a \in \{2,3\}$ ; ce qui prouve que, contrairement à ce qui se passe avec AC sur les fonction booléennes (CSP), chaque table de fonction de coût peut être parcourue plusieurs fois, même à la racine. Pour une approche itérative qui nécessite de résoudre $M$ problèmes, optimiser l'encodage devient d'autant plus intéressant.

Dans cette section, nous montrons que l'encodage d'une contrainte Dist en une séquence de $n+1$ contraintes ternaires décrites par des tables de taille $(\delta+1)^2.(n+1)^2.d$ peut être réduit de plusieurs façons. Pour Dist, on sait que les états $s_i^d$ ne peuvent être atteints qu'après exactement $i$ transitions et sont propres à la variable $Q_i$ : les domaines des variables $Q_i, 0 < i < n$ peuvent être réduits aux $\delta+1$ états $s_i^d$. De plus, nos semi-métriques sont définies par une somme croissante d'éléments positifs de $D$, donc chaque état $s_i^d$ peut atteindre l'état acceptant $s_n^\delta$ ssi la dissimilarité maximale (notée $md_i$) qui peut être atteinte à partir de la variable $i$ jusqu'à la variables $n$ est supérieure à $\delta - d$. Toutes ces dissimilarités maximales peuvent être calculées en amont en un parcours des variables de $\mathbf{X}$, puisque $md_1 = 0; md_i = md_{i-1} + max_{r,s \in D_i \times D_{i+1}} D(r,s)$. Dans le cas de la distance de Hamming, la distance ne peut augmenter que de 1 à chaque variables, et cette valeur est simplement $n-i$. Un raisonnement symétrique peut être appliqué à partir de l'état initial $s_0^0$. Ces simplifications réduisent la taille des tables des fonctions de coût à $O((\delta+1)^2.d)$. Même si de tels états non atteignables seraient systématiquement éliminés par AC dans le cas booléen, un grand nombre d'EPT pourraient être appliquées dans le cas CFN.

### 6.1 Représentations duale et cachée pour Dist

Pour une matrice de dissimilarité $D$ donnée, on note $\#D$ le nombre de valeurs distinctes qui apparaissent dans $D$. Si les variables ont des domaines de taille inférieure à $d$, et si on ignore le cas trivial de la matrice nulle, on sait que $2 \leq \#D \leq 1 + \frac{d(d-1)}{2}$. Cependant, les matrices de distances sont souvent plus structurées. Pour Hamming $H$, on a $\#H = 2$. Cela signifie qu'un état $s_i^d$ ne peut qu'atteindre les états $s_{i+1}^d$ ou $s_{i+1}^{d+1}$. Pour exploiter la parcimonie de la matrice de transition, il faut la rendre visible. Pour cela, on peut utiliser les représentations duale ou cachée introduites dans les réseaux de contraintes [4].

En satisfaction de contraintes, la représentation *duale* d'un réseau de contraintes (un CFN $(\mathbf{X},\mathbf{W})$ avec $k=1$) est un nouveau réseau $(\mathbf{X}',\mathbf{W}')$ composé d'une variables $X_\mathbf{S}$ pour chaque fonction $w_\mathbf{S} \in W$ avec un domaine défini par tous les $n$-uplets $\mathbf{t} \in D^\mathbf{S}$ tels que $w_\mathbf{S}(t) \neq k$. De plus, pour chaque paire de fonctions $w_\mathbf{S}, w_{\mathbf{S}'} \in W$ telle que $\mathbf{S} \cap \mathbf{S}' \neq \varnothing$, il existe une fonction de portée $X_\mathbf{S}$ et $X_{\mathbf{S}'}$ qui attribue un coût à toutes les paires de $n$-uplets des domaines de $X_\mathbf{S}$ et $X_{\mathbf{S}'}$ de 0 si $\mathbf{t}[\mathbf{S} \cap \mathbf{S}'] = \mathbf{t}'[\mathbf{S} \cap \mathbf{S}']$ et de $k$ sinon.

La représentation *cachée* de $(\mathbf{X},\mathbf{W})$ est un CFN $(\mathbf{X}'',\mathbf{W}'')$ qui contient les variables de $\mathbf{X}$ et les variables $X_\mathbf{S}$ du réseau dual. Pour tout variable $X_\mathbf{S}$, et toute variable $X_i \in \mathbf{S}$, $\mathbf{W}''$ contient une fonction de portée $X_i$ et $X_\mathbf{S}$ qui attribue un coût de 0 à la paire $(a,\mathbf{t})$ si $\mathbf{t}[\{X_i\}] = a$ et de $k$ sinon. Dans les CSP [4] ainsi que dans les CFN [21], il a été montré que ces transformations préservaient l'ensemble des solutions et leur coût. Au lieu d'appliquer ces transformations à toutes les fonctions du CFN, l'idée est de ne les appliquer qu'aux fonctions de décomposition $w^A_{Q_i,X_i,Q_{i+1}}$.

La variable duale de $w^A_{Q_i,X_i,Q_{i+1}}$ est une variable $X_i^A$ qui contient toutes les paires $(s,s')$ de $Q_i \times Q_{i+1}$ telles qu'il existe une transition de $s$ vers $s'$ dans l'automate. Dans le cas Hamming, cette variable a au plus $2\delta+1$ valeurs. Elle est connectée à la variable $X_i$ par une fonction binaire qui attribut un coût de 0 à la paire $((s,s'),a)$ des domaines de $X_i^A$ et $X_i$ respectivement ssi il y a une transition de $s$ vers $s'$ étiquetée par $a$ et un coût $k$ sinon. Elle ne contient que $O(d.\delta)$ paires.

Dans la représentation duale, pour toute paire de variables duales $X_{i-1}^A$ et $X_i^A$, on ajoute une fonction entre ces deux variables qui attribue un coût de 0 à toute paire $((s_{i-1},s'_{i-1}),(s_i,s'_i))$ si $s'_{i-1} = s_i$ et $k$ sinon. En pire cas, cette fonction a une taille $O(\#D^2.\delta^2)$ et

$O(\delta^2)$ dans le cas Hamming. Seulement $n$ variables supplémentaires sont nécessaires.

Dans la représentation cachée, les variables $Q_i$ sont conservées et on crée deux fonctions binaires qui lient chaque $Q_i$ avec $X_i^A$ et $X_{i-1}^A$ respectivement. Elles attribuent un coût 0 aux paires $(s'', (s, s'))$ ssi $s'' = s$ pour la fonction liant $Q_i$ à $X_i^A$ (respectivement $s'' = s'$ pour la fonction liant $Q_i$ à $X_{i-1}^A$). Ces fonctions sont de taille $O(\#D.\delta^2)$ et $O(\delta^2)$ dans le cas Hamming.

Les représentations duale et cachée nécessitent la description de $O(\delta.d+\#D^2\delta^2)$ and $O(\delta.d+\#D\delta^2)$ $n$-uplets respectivement ($O(\delta.d + \delta^2)$ dans le cas Hamming) au lieu des $O(d.\delta^2)$ $n$-uplets de $w^A_{Q_i,X_i,Q_{i+1}}$.

## 7 DiverseSeq glouton

Pour résoudre le problème $\textsc{DiverseSeq}(N, D, M, \delta)$, on peut utiliser l'approche gloutonne suivante : nous résolvons le CFN $N$ via un Branch and Bound. Si une solution $\mathbf{t}$ est trouvée, elle est ajoutée à la séquence de solutions en cours $\mathbf{Z}$. Si $M$ solutions ont été produites, nous nous arrêtons. Sinon, une contrainte $\textsc{Dist}(\mathbf{X}, \mathbf{t}, D, \delta)$ est ajoutée au problème précédent. Nous effectuons une boucle et le résolvons à nouveau. Si aucune solution n'existe, la séquence $\mathbf{Z}$ ne peut pas être étendue à la longueur $M$ et le problème n'a pas de solution (mais une séquence plus courte a été produite). Ce schéma peut être amélioré de trois manières qui peuvent chacune offrir des gains exponentiels en temps.

1) les problèmes étant de plus en plus contraints, le filtrage appliqué à l'itération $i-1$ reste valide pour les itérations suivantes. Au lieu de redémarrer à partir d'un problème $N = (\mathbf{X}, \mathbf{W} \cup_{1\leq j<i} \{\ Dist(\mathbf{X}, \mathbf{Z}[j], D, \delta\})$, nous réutilisons le problème résolu à l'itération $i-1$ après filtrage pour ajouter la contrainte $\textsc{Dist}(\mathbf{X}, \mathbf{Z}[i-1], D, \delta)$ et filtrer de nouveau. De plus, les paramètres des heuristiques de choix de variables appris à l'itération $i-1$ sont réutilisés à l'itération $i$.

2) les problèmes étant de plus en plus contraints, le coût optimal $oc^i$ obtenu à l'itération $i$ ne peut avoir un coût inférieur au coût $oc^{i-1}$. Lorsque de grands plateaux sont présents, cela permet d'arrêter la recherche dès qu'une solution de coût $oc^{i-1}$ est atteinte, évitant ainsi une preuve d'optimalité.

3) même s'il n'y a pas de plateau dans le paysage énergétique, il peut y avoir de grandes régions avec des variations d'énergie similaires. Dans ce cas, la différence d'énergie entre $oc^{i-1}$ et $oc^i$ restera similaire pour plusieurs itérations. Soit $\Delta_i^h = \max_{\max(2,ih)\leq j<i}(oc^j - oc^{j-1})$, la variation maximale dans les dernières $h$ itérations ($h = 5$ semble raisonnable). A l'itération $i$, nous résolvons le problème avec un majorant temporaire $k' = \min(k, oc_{i-1} + 2, \Delta_i^h)$. Si $k' < k$, cela entraînera un déterminisme accru et un élagage éventuellement plus efficace. Si aucune solution n'est trouvée, le problème est résolu à nouveau avec un majorant initial égal à $k$. Le majorant est donc prédit.

## 8 Expérimentations

Nous avons implémenté l'approche itérative décrite ci-dessus au moyen d'une décomposition directe (ternaire) ainsi que ses versions cachée et duale pour $\textsc{Div}_{\min}$ décomposée en une conjonction de fonctions $\textsc{Dist}$ au-dessus du solveur de CFN C++ open source toulbar2 [18]. Toulbar2 implémente divers algorithmes de pré-traitement dédiés à l'optimisation exacte. Nous avons dû désactiver au niveau du nœud racine tous les algorithmes de pré-traitement ne préservant pas les solutions sous-optimales : élimination des variables, substituabilité et fusion de variables. Nous avons choisi d'appliquer la cohérence d'arc virtuelle au premier problème résolu. Le coût de calcul de VAC, en $O(\frac{ed^2k}{\epsilon})$, où $\epsilon$ est la plus petite énergie représentable en décimal dans la résolution utilisée, est important. Mais il est amorti sur les $M$ résolutions. Lors de la recherche arborescente, la cohérence locale EDAC usuelle a été utilisée. Le code a été exécuté sur un cœur d'un processeur Xeon E5-2680 à 2,50 GHz avec $\epsilon = 10^{-6}$.

Nous avons utilisé des réseaux Bayésiens issus de `http://www.bnlearn.com/bnrepository` de taille *medium*, *large*, *very large* et *massive*. Nous avons utilisé tous les réseaux complets, en l'état. La table 1 indique le nombre de variables $n$, la taille de domaine maximum $d$ et le nombre de solutions 4-diverses trouvées en moins de 5 minutes par instance, pour chaque décomposition. Bien que ce ne soit pas notre objectif principal, dans le cas dual (le plus rapide), nous avons également calculé le nombre de solutions 4-diverses qui sont des 3-modes [11]. Vérifier si une assignation $\mathbf{t}$ est un $\delta$-mode du CFN $(\mathbf{X}, \mathbf{W})$ peut être effectué efficacement en résolvant $(\mathbf{X}, \mathbf{W} \cup \{\textsc{Dist}(\mathbf{X}, \mathbf{t}, D, -\delta)\}$ (Voir Th. 2). Pour comparer avec [11], nous donnons également le temps total de calcul pour trouver 4 de ces 3 modes. Ce temps inclut le temps nécessaire pour rechercher tous les modes 4 divers et vérifier s'il s'agit de 3-modes. Ce temps est dominé par le temps de calcul des solutions 4-diverses (la vérification que ce sont des 3-modess a pris moins de 6 minutes au total). Pour les deux problèmes (*child* et *alarm*) résolus dans [11], nous obtenons des temps similaires. Le paysage énergétique de chaque problème influe sur la fraction de solutions $(\delta + 1)$-diverses qui sont des $\delta$-modes : sur *link* et *pigs*, qui présentent un large plateau, notre approche est très rapide. L'instance *hepar2* montre quant à elle un large bassin autour de l'optimum avec de nombreuses solutions 4-diverses mais un seul minimum local.

Nous avons également comparé notre approche glou-

| | | | dual | | | hid | 3ary |
|---|---|---|---|---|---|---|---|
| **name** | $n$ | $d$ | 4-d | 3-m | cpu | 4-d | 4-d |
| alarm | 37 | 4 | 147 | 28 | 0.38 | 128 | 141 |
| andes | 223 | 2 | 102 | 8 | 4.96 | 87 | 110 |
| barley | 48 | 67 | 126 | 92 | 2,74 | 125 | 113 |
| child | 20 | 6 | 172 | 8 | 1.65 | 153 | 154 |
| diabete | 413 | 21 | 0 | 0 | TO | 0 | 0 |
| hailfi | 56 | 11 | 109 | 24 | 0.65 | 92 | 102 |
| hepar2 | 70 | 4 | 96 | 1 | TO | 87 | 93 |
| insuran | 27 | 5 | 186 | 30 | 0.40 | 164 | 153 |
| link | 724 | 4 | 117 | 117 | 6.69 | 113 | 93 |
| mildew | 35 | 100 | 107 | 25 | 7.71 | 100 | 97 |
| munin | 1041 | 21 | 16 | 2 | TO | 16 | 14 |
| pathfi | 109 | 63 | 118 | 15 | 2.46 | 90 | 124 |
| pigs | 441 | 3 | 162 | 162 | 6.28 | 134 | 138 |
| water | 32 | 4 | 183 | 17 | 0.48 | 161 | 174 |
| win95 | 76 | 2 | 132 | 21 | 0.93 | 115 | 119 |

TABLE 1 – Pour chaque réseau, nous donnons son nom, nombre de variables et taille de domaine. Pour l'encodage dual, nous donnons le nombre de solutions 4-diverses trouvées en 5 minutes, le nombre de 3-modes et le temps nécessaire pour produire quatre 3-modes, comme dans [11]. Nous donnons ensuite le nombre de solutions 4-diverses produites dans le même temps par l'encodage caché et ternaire.

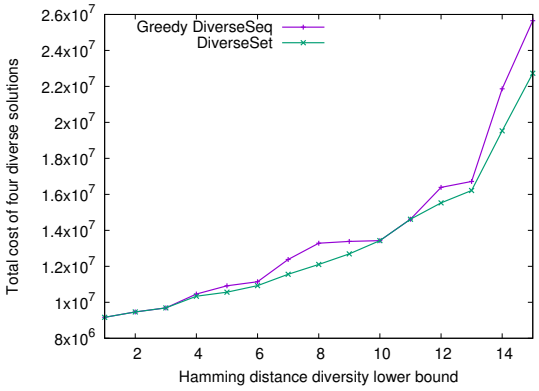



FIGURE 1 – Comparison entre DIVERSESEQ et DIVERSESET sur *child*.

tonne DIVERSESEQ à une approche globale optimale permettant de résoudre le problème DIVERSESET. Comme décrit dans la section 2, nous avons créé $M = 4$ copies du réseau bayésien *child* (catégorie medium). Nous avons décomposé chaque contrainte $\text{DIST}(\mathbf{X}^i, \mathbf{X}^j, H, \delta)$ sur les ensembles de variables copie $\mathbf{X}^i$ et $\mathbf{X}^j$ en une séquence de fonctions ternaires utilisant une variable booléenne $\mathbf{E}^{i,j}$, avec $\forall 1 \leq l \leq n, \mathbf{E}^{i,j}[l] = \mathbb{1}(\mathbf{X}^i[l] \neq \mathbf{X}^j[l])$ contraintes d'inégalité supplémentaires. Le problème DIVERSESET résultant a 320 variables. Nous l'avons résolu avec toulbar2 version 1.0, en variant $\delta$ de 1 à 15. Il a fallu plus de 23 heures (resp. 3 heures) pour résoudre DIVERSESET avec $\delta = 15$ (resp. $\delta = 14$). L'approche gloutonne prend 0.28 seconde pour $\delta = 15$ et produit une séquence de quinze solutions en 1.1 secondes. Cette efficacité n'entraîne que peu d'effets secondaires en termes d'énergie moyenne : la figure 1 montre le coût total $\sum_{\mathbf{t} \in \mathbf{Z}} J_N(\mathbf{t})$ pour les deux approches. L'approche gloutonne itérative reste optimale sur de grandes distances (jusqu'à $\delta = 11$) et est à moins de 12.9 de l'optimum pour $\delta = 15$.

Pour le CPD, nous avons extrait un ensemble de 20 squelettes de protéines pour un redesign complet du jeu de benchamrk de [32]. Nous avons sélectionné les 20 protéines résolues le plus rapidement (indiqué dans la colonne $S$ dans les informations supplémentaires de l'article : 1aho, 1b9w, 1f94, 1hyp, 1uln, 1uoy , 1yzm, 2cg7, 2erw, 2fht, 2fjz, 2gkt, 2pne, 2pst, 2qt4, 3ca7, 3i8z, 3rdy, 3vdj et 4pti). Nous avons utilisé les scripts fournis avec le champ de force Rosetta `ref2015` [13] pour créer le modèle énergétique CFN pour chaque réseau. Le nombre de variables dans ces problèmes varie de 46 à 109, les tailles de domaine maximales allant de 329 à 414. Pour chaque problème, nous avons généré des ensembles de $M = 10$ solutions diverses avec $\delta = 1$ et $\delta \in \{7, 8, 9, 10\}$. Pour $\delta = 1$, l'ensemble des solutions diverses produit est l'ensemble des 10 meilleures séquences. Le temps CPU maximum passé sur un ensemble est de 32 minutes, et est réduit à 17 minutes avec un majorant prédit. Le temps moyen est de 201 secondes par ensemble.

Les métriques usuelles pour évaluer les protocoles de CPD sont la «native sequence recovery» (nsr) et la «native sequence similarity recovery» (nssr). Elles mesurent à quel point la protéine conçue ressemble à la protéine naturelle (nsr : pourcentage de positions avec le même acide aminé, nssr : pourcentage de positions avec un score positif dans la matrice de similarité BLOSUM62). Si la diversité des solutions est utile, la meilleure nsr/nssr sur les 10 séquences devrait augmenter lorsque $\delta$ dépasse 1, tant que l'énergie reste proche de l'énergie optimale. Même avec $\delta = 10$, la différence maximale d'énergie observée avec le minimum global n'a jamais dépassé 2.7 kcal/mol (avec une moyenne de 1.21 kcal/mol), ce qui reste raisonnable. Nous avons donc comparé, pour chaque protéine, les meilleurs nsr et nssr pouvant être obtenus avec $\delta \in \{7, 8, 9, 10\}$ avec les meilleurs obtenus avec $\delta = 1$. Nous observons une augmentation systématique des meilleures valeurs nsr et nssr moyennes pour $\delta > 1$ par rapport à $\delta = 1$ (voir tableau 2, $p$-valeur pour un test de Wilcoxon signé unilatéral comparant l'échantillon des 10 meilleurs nsr/nssr pour chaque $\delta$ par rapport à $\delta = 1$).

| $\delta =$ | 1 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|
| nsr | 45.5 | 47.0 | 46.6 | 46.9 | 46.7 |
| $p$-value | N/A | 0.004 | 0.007 | 0.008 | 0.016 |
| nssr | 60.1 | 61.1 | 61.5 | 61.9 | 61.3 |
| $p$-value | N/A | 0.015 | 0.003 | 0.002 | 0.021 |

TABLE 2 – Pour chaque $\delta \in \{1, 7, 8, 9, 10\}$, nous donnons la moyenne sure toutes les protéines des meilleures nsr (ligne 2) et nssr (ligne 4) parmi les 10 séquences $\delta$-diverses produites. Les lignes 3 et 5 donnent les $p$-valeurs testant le fait que nsr (resp nssr) s'améliore pour chaque $\delta$ par rapport à $\delta = 1$.

## 9 Conclusion

Produire une séquence de solutions diverses est une exigence très courante lorsqu'un modèle approximatif est utilisé. Dans cet article, nous montrons que des séquences de solutions qui satisfont aux garanties de diversité peuvent être obtenues sur de grands réseaux Bayésiens ainsi que pour de grandes instances de CPD. Cette garantie est obtenue sur des problèmes denses de fonctions non sous-modulaires tout en garantissant que chaque nouvelle solution produite est la meilleure compte tenu des solutions précédemment identifiées.

Bien que la garantie de la diversité soit importante, dans le contexte d'exigences légales par exemple, dans le contexte de l'optimisation d'une fonction approximative, l'exigence d'une solution optimale pourrait être considérée comme extrême. Elle reste préférable à une optimisation stochastique offrant des garanties asymptotiques, avec une erreur non bornée en pratique [32].

Nous avons également montré que la séquence de solutions diverses produite par notre algorithme peut être filtrée et chaque solution efficacement identifiée comme étant un $\delta$-mode (ou non). Dans les quelques problèmes où nous avons pu comparer notre algorithme aux résultats de [11], notre approche semble tout aussi efficace en temps de calcul, tout en fournissant des informations plus détaillées sur la forme du paysage énergétique autour de l'optimum.

Il y a deux suites possibles à ces travaux. La première direction est mise en évidence par le problème *diabetes* pour lequel le premier élément de DIVERSESEQ ne peut être produit rapidement. Ce problème a une largeur d'arbre inférieure à 4 et peut être résolu en moins de 40 secondes si l'on exploite une décomposition arborescente (min-fill) pendant la recherche. Il faudrait donc montrer que les fonctions décomposées que nous utilisons pour encoder DIST peuvent être agencées de manière à ce que la largeur de l'arbre puisse être préservée ou peu augmentée. La seconde direction consisterait à identifier une version relâchée de la contrainte NP-difficile $\text{DIV}_{\min}$ capable de produire de bons minorants par filtrage. Malgré des efforts non négligeables dans cette direction en utilisant des diagrammes de décision multi-valués, nous n'avons pas abouti à des résultats convaincants.

## Références

[1] D. ALLOUCHE, I. ANDRÉ, S. BARBE, J. DAVIES, S. DE GIVRY, G. KATSIRELOS, B. O'SULLIVAN, S. PRESTWICH, T. SCHIEX et S. TRAORÉ : Computational protein design as an optimization problem. *Artificial Intelligence*, 212:59–79, 2014.

[2] D. ALLOUCHE, C. BESSIERE, P ; BOIZUMAULT, S. DE GIVRY, P. GUTIERREZ, J HM LEE, K L LEUNG, S. LOUDNI, J-P. MÉTIVIER, T. SCHIEX *et al.* : Tractability-preserving transformations of global cost functions. *Artificial Intelligence Journal*, 238:166–189, 2016.

[3] D. ALLOUCHE, S. DE GIVRY, G. KATSIRELOS, T. SCHIEX et M. ZYTNICKI : Anytime hybrid best-first search with tree decomposition for weighted csp. *In Proc. of Principles and Practice of Constraint Programming (CP'15)*, pages 12–29. Springer, 2015.

[4] F. BACCHUS et P. VAN BEEK : On the conversion between non-binary and binary constraint satisfaction problems. *In Proceedings of AAAI 1998*, pages 310–318, 1998.

[5] D. BATRA, P. YADOLLAHPOUR, A. GUZMAN-RIVERA et G. SHAKHNAROVICH : Diverse m-best solutions in markov random fields. *In European Conference on Computer Vision*, pages 1–16. Springer, 2012.

[6] C. BESSIÈRE et P. VAN HENTENRYCK : To be or not to be ... a global constraint. *In Proceedings of CP'03*, pages 789–794, 2003.

[7] F. BOUSSEMART, F. HEMERY, C. LECOUTRE et L. SAIS : Boosting systematic search by weighting constraints. *In ECAI*, volume 16, page 146, 2004.

[8] C. CHEN, V. KOLMOGOROV, Y. ZHU, D. METAXAS et C. LAMPERT : Computing the m most probable modes of a graphical model. *In Proc. of Artificial Intelligence and Statistics*, pages 161–169, 2013.

[9] C. CHEN, H. LIU, D. METAXAS et T. ZHAO : Mode estimation for high dimensional discrete tree graphical models. *In Proceedings of Advances in neural information processing systems*, pages 1323–1331, 2014.

[10] C. CHEN, C. YUAN et C. CHEN : Solving m-modes using heuristic search. *In Proc. of IJCAI'16*, pages 3584–3590, 2016.

[11] C. Chen, C. Yuan, Z. Ye et C. Chen : Solving m-modes in loopy graphs using tree decompositions. *In Proc. of the International Conference on Probabilistic Graphical Models*, pages 145–156, 2018.

[12] M. Cooper, S. de Givry, M. Sanchez, T. Schiex, M. Zytnicki et Tomas Werner : Soft arc consistency revisited. *Artificial Intelligence Journal*, 174:449–478, 2010.

[13] R. Das et D. Baker : Macromolecular modeling with rosetta. *Annu. Rev. Biochem.*, 77:363–382, 2008.

[14] S. de Givry, .s Zytnicki, F. Heras et J. Larrosa : Existential arc consistency : Getting closer to full arc consistency in weighted CSPs. *In Proc. of IJCAI-05*, pages 84–89, Edinburgh, Scotland, 2005.

[15] T. Hadžić, A. Holland et B. O'Sullivan : Reasoning about optimal collections of solutions. *In International Conference on Principles and Practice of Constraint Programming*, pages 409–423. Springer, 2009.

[16] E. Hebrard, B. Hnich, B. O'Sullivan et T. Walsh : Finding diverse and similar solutions in constraint programming. *In Proceedings of AAAI 2005*, volume 5, pages 372–377, 2005.

[17] E. Hebrard, B. O'Sullivan et T. Walsh : Distance constraints in constraint satisfaction. *In IJCAI*, volume 2007, pages 106–111, 2007.

[18] B. Hurley, B. O'Sullivan, D. Allouche, G. Katsirelos, T. Schiex, M. Zytnicki et S. de Givry : Multi-language evaluation of exact solvers in graphical model discrete optimization. *Constraints*, pages 1–22, 2016.

[19] A. Kirillov, B. Savchynskyy, D. Schlesinger, D. Vetrov et C. Rother : Inferring m-best diverse labelings in a single one. *In Proceedings of the IEEE International Conference on Computer Vision*, pages 1814–1822, 2015.

[20] V. Kolmogorov : Convergent tree-reweighted message passing for energy minimization. *IEEE transactions on pattern analysis and machine intelligence*, 28(10):1568–1583, 2006.

[21] J. Larrosa et R. Dechter : On the dual representation of non-binary semiring-based CSPs. *In CP'2000 workshop on soft constraints*, 2000.

[22] J. H. M. Lee et K. L. Leung : Consistency Techniques for Global Cost Functions in Weighted Constraint Satisfaction. *Journal of Artificial Intelligence Research*, 43:257–292, 2012.

[23] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang et S. Malik : Chaff : Engineering an efficient sat solver. *In 38$^{th}$ Design Automation Conference (DAC'01)*, pages 530–535, juin 2001.

[24] H. Noguchi, C. Addy, D. Simoncini, S. Wouters, B. Mylemans, L. Van Meervelt, T. Schiex, K.Y. Zhang, JRH Tame et ARD Voet : Computational design of symmetrical eight-bladed $\beta$-propeller proteins. *IUCrJ*, 6(1), 2019.

[25] G. Pesant : A regular language membership constraint for finite sequences of variables. *In International conference on principles and practice of constraint programming*, pages 482–495. Springer, 2004.

[26] T. Petit et A. C. Trapp : Finding diverse solutions of high quality to constraint optimization problems. *In Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[27] A. Prasad, S. Jegelka et D. Batra : Submodular meets structured : Finding diverse subsets in exponentially-large structured item sets. *In Advances in Neural Information Processing Systems*, pages 2645–2653, 2014.

[28] F. Rossi, P. van Beek et T. Walsh, éditeurs. *Handbook of Constraint Programming*. Elsevier, 2006.

[29] M. Sánchez, S. de Givry et T. Schiex : Mendelian error detection in complex pedigrees using weighted constraint satisfaction techniques. *Constraints*, 13(1):130–154, 2008.

[30] T. Schiex : Arc consistency for soft constraints. *In Principles and Practice of Constraint Programming - CP 2000*, volume 1894 de *LNCS*, pages 411–424, Singapore, septembre 2000.

[31] D. Schlesinger : Exact solution of permuted submodular minsum problems. *In International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 28–38. Springer, 2007.

[32] D. Simoncini, D. Allouche, S. de Givry, C. Delmas, S. Barbe et T. Schiex : Guaranteed discrete energy optimization on large protein design problems. *Journal of chemical theory and computation*, 11(12):5980–5989, 2015.

[33] S. Traoré, D. Allouche, I. André, S. De Givry, G. Katsirelos, Thomas S. et S. Barbe : A new framework for computational protein design through cost function network optimization. *Bioinformatics*, 29(17):2129–2136, 2013.

[34] M. Zytnicki, C. Gaspin, S. de Givry et T. Schiex : Bounds Arc Consistency for Weighted CSPs. *Journal of Artificial Intelligence Research*, 35:593–621, 2009.

# Une heuristique basée sur l'historique des conflits pour les problèmes de satisfaction de contraintes*

**Djamal Habet** **Cyril Terrioux**
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
{djamal.habet, cyril.terrioux}@lis-lab.fr

**Résumé**

L'heuristique de choix de variables est une brique importante pour les algorithmes de résolution du problème de satisfaction de contraintes (CSP). Elle a une influence souvent considérable sur l'efficacité de la recherche et permet, d'une certaine manière, d'exploiter la structure des instances. Dans cet article, nous proposons l'heuristique CHS (pour Conflict-History Search) qui est une heuristique dynamique et adaptative pour la résolution d'instances CSP. Elle repose sur les échecs rencontrés durant la recherche et considère leur temporalité tout au long de la résolution.

Une technique d'apprentissage par renforcement est exploitée pour estimer l'évolution de la difficulté des contraintes durant la recherche. Les expérimentations réalisées sur des instances au format XCSP3 permettent de montrer que l'intégration de CHS au sein d'un solveur basé sur l'algorithme MAC s'avère pertinente, conduisant notamment à des résultats meilleurs que ceux obtenus avec des heuristiques de l'état de l'art comme dom/wdeg et ABS.

Ce papier est un résumé de [2].

## 1 Contexte

Une instance CSP (pour Problème de Satisfaction de Contraintes) est définie par la donnée d'un triplet $(X, D, C)$, où $X = \{x_1, \ldots, x_n\}$ est un ensemble de $n$ variables, $D = \{d_{x_1}, \ldots, d_{x_n}\}$ est un ensemble de domaines finis de taille au plus $d$, et $C = \{c_1, \ldots, c_e\}$ est un ensemble de $e$ contraintes. Chaque contrainte $c_i$ est un couple $(S(c_i), R(c_i))$, où $S(c_i) = \{x_{i_1}, \ldots, x_{i_k}\} \subseteq X$ définit la *portée* de $c_i$, et $R(c_i) \subseteq d_{x_{i_1}} \times \cdots \times d_{x_{i_k}}$ est une *relation de compatibilité*. Une affectation d'un sous-ensemble de $X$ est dite *localement cohérente* si toutes les contraintes couvertes par ce sous-ensemble sont satisfaites. Une *solution* est une affectation localement cohérente de toutes les variables. Déterminer si une instance CSP possède une solution est un problème NP-complet.

Dans ce contexte, l'heuristique de choix de variables permet de désigner la prochaine variable à instancier. Elle joue un rôle important dans la résolution des instances CSP. Son influence sur l'efficacité de la recherche est souvent considérable. Dans la littérature, de nombreuses heuristiques ont été proposées. Les heuristiques dynamiques et adaptatives (comme, par exemple, dom/wdeg [1] et ABS [5]), sont généralement celles conduisant aux meilleurs résultats.

## 2 L'heuristique CHS

Nous proposons ici l'heuristique CHS (pour Conflict-History Search), inspirée de l'heuristique CHB [4] et introduite dans le cadre du problème SAT. L'idée est de considérer l'historique des échecs rencontrés et de favoriser les variables qui apparaissent dans des échecs récents. Aussi, les échecs sont datés et une technique d'apprentissage par renforcement (ERWA pour *exponential recency weighted average*) est exploitée pour la pondération des contraintes.

Plus précisément, CHS maintient pour chaque contrainte $c_j$ un score $q(c_j)$ qui est initialisé à 0 au début de la recherche. Lorsque que $c_j$ conduit à un échec en vidant le domaine d'une des variables de $S(c_j)$ par propagation, $q(c_j)$ est mis à jour avec la formule ci-dessous, dérivée d'ERWA :

$$q(c_j) = (1 - \alpha) \times q(c_j) + \alpha \times r(c_j)$$

Le paramètre $0 < \alpha < 1$ définit l'importance donnée à l'ancienne valeur de $q(c_j)$ par rapport à la valeur $r(c_j)$ de la récompense. Sa valeur, initialisée à une valeur $\alpha_0$ donnée, décroit au cours du temps par pas de $10^{-6}$ jusqu'à une valeur minimum fixée à $0,06$. Cette décroissance permet d'accorder plus d'importance à la dernière valeur de $q(c_j)$, l'idée sous-jacente étant que

*Ce travail est soutenu par l'Agence Nationale de la Recherche dans le cadre du projet DEMOGRAPH (ANR-16-C40-0028)

la valeur de $q(c_j)$ est de plus en plus pertinente au fur et à mesure que la recherche progresse.

La valeur de la récompense dépend directement des échecs rencontrés récemment au niveau de $c_j$. L'objectif est de récompenser davantage les contraintes qui conduisent régulièrement à des échecs sur des courtes périodes de temps. Cette valeur est calculée ainsi :

$$r(c_j) = \frac{1}{\#Conflicts - Conflict(c_j) + 1}$$

Initialisé à 0, $\#Conflicts$ est le nombre d'échecs rencontrés depuis le début de la recherche tandis que $Conflict(c_j)$ (initialisé à 0) mémorise, pour chaque contrainte $c_j$, la valeur qu'avait $\#Conflicts$ lors du dernier échec rencontré grâce à $c_j$. À chaque mise à jour de $r(c_i)$ et de $q(c_i)$, $\#Conflicts$ est incrémenté de un.

Nous pouvons maintenant définir le score associé à chaque variable $x_i$ :

$$chv(x_i) = \frac{\sum_{c_j \in C \ | \ x_i \in S(c_j) \wedge |Uvars(c_j)|>1} (q(c_j) + \delta)}{|D_i|}$$

Dans cette formule, $Uvars(c_j)$ désigne l'ensemble des variables de $S(c_j)$ qui ne sont pas encore affectées. Le paramètre $\delta$ permet de débuter la recherche avec des scores initiaux reflétant le nombre de contraintes dans lesquelles apparaît chaque variable. Sa valeur est généralement inférieure à $10^{-3}$ afin d'avoir très rapidement un poids négligeable par rapport aux $q(c_j)$. Au final, CHS choisit donc, comme prochaine variable, celle qui a le plus grand score $chv$. En procédant ainsi, CHS privilégie les variables ayant un petit domaine et intervenant dans des contraintes qui sont à l'origine d'échecs récents et récurrents.

Les redémarrages constituent également une brique importante des solveurs actuels. Ils permettent notamment de réduire les conséquences négatives de mauvais choix que pourrait faire l'heuristique de choix de variables. Lorsqu'un redémarrage survient, la valeur de $Conflict(c_j)$ est conservée. Par contre, la valeur du paramètre $\alpha$ est, elle, réinitialisée à la valeur $\alpha_0$ à chaque redémarrage. Dans le même temps, les valeurs des $q(c_j)$ sont lissées suivant la formule suivante :

$$q(c_j) = q(c_j) \times 0,995^{\#Conflicts - Conflict(c_j)}$$

Un redémarrage permet potentiellement d'explorer une nouvelle partie de l'espace de recherche. Aussi, ce lissage permet de diminuer l'importance d'anciens échecs correspondant à d'autres parties de l'espace de recherche. Par ailleurs, la réinitialisation de $\alpha$ a pour but de donner l'opportunité à l'heuristique d'explorer à son plein potentiel cette nouvelle partie de l'espace de recherche.

## 3 Résultats expérimentaux

Pour ces expérimentations, nous considérons 10 785 instances du dépôt XCSP3 [1], incluant notamment des instances structurées et excluant les instances purement aléatoires. Nous exploitons l'algorithme MAC avec redémarrages [3] pour la résolution des instances.

Nous avons d'abord fait varier les paramètres $\alpha_0$ et $\delta$. Nous avons alors constaté que l'heuristique CHS était peu sensible aux valeurs de ces paramètres. En effet, pour les valeurs étudiées, le nombre d'instances résolues varie de 9 515 à 9 525 pour un temps d'exécution cumulé allant de 491,7 h à 498,2 h. Nous avons également pu observer l'apport du lissage et de la réinitialisation de $\alpha$ à chaque redémarrage, la désactivation de l'une ou l'autre de ces fonctionnalités entraînant une dégradation de performances (jusqu'à 47 instances résolues en moins).

Enfin, comparée aux heuristiques de l'état de l'art (voir la table 1), CHS permet à MAC de résoudre plus d'instances qu'avec dom/wdeg ou ABS tout en requérant moins de temps.

| Solveur | #instances | temps (h) |
|---|---|---|
| **MAC+CHS** | **9,525** | **493.41** |
| MAC+*dom/wdeg* | 9,501 | 507.17 |
| MAC+ABS | 9,476 | 515.17 |

Table 1 – Nombre d'instances résolues par MAC avec dom/wdeg, ABS et CHS ($\alpha_0 = 0,4$ et $\delta = 10^{-4}$) et temps d'exécution cumulé en heures.

## 4 Conclusion

Nous avons proposé une nouvelle heuristique CHS qui repose sur l'historique des échecs rencontrés. Son utilisation au sein d'un solveur basé sur MAC a permis de résoudre plus d'instances que des heuristiques de l'état de l'art comme dom/wdeg ou ABS.

## Références

[1] F. Boussemart, F. Hemery, C. Lecoutre et L. Sais : Boosting systematic search by weighting constraints. *In ECAI*, pages 146–150, 2004.

[2] Djamal Habet et Cyril Terrioux : Conflict History based Search for Constraint Satisfaction Problem. *In SAC*, pages 1117–1122, 2019.

[3] C. Lecoutre, L. Saïs, S. Tabary et V. Vidal : Recording and Minimizing Nogoods from Restarts. *JSAT*, 1(3-4):147–167, 2007.

[4] J. H. Liang, V. Ganesh, P. Poupart et K. Czarnecki : Exponential Recency Weighted Average Branching Heuristic for SAT Solvers. *In AAAI*, pages 3434–3440, 2016.

[5] L. Michel et P. Van Hentenryck : Activity-based search for black-box constraint programming solvers. *In CP-AI-OR*, pages 228–243, 2012.

1. http ://www.xcsp.org/series

# Une autre règle de séparation pour des codages de CSP vers SAT

**Richard Ostrowski**[1] **Lionel Paris**[1] **Adrien Varet**[2]

[1] Laboratoire d'Informatique et Systèmes, UMR 7020
[2] Université d'Aix Marseille, France
{richard.ostrowski,lionel.paris}@lis-lab.fr adrien.varet@etu.univ-amu.fr

**Résumé**

Dans cet article, nous proposons une manière différente de parcourir l'arbre de recherche de problèmes CSP codés vers SAT (codage direct, codage des supports). Le parcours de l'espace de recherche s'effectue en utilisant une règle de séparation différente des solveurs habituels. De manière générale, pour le problème de satisfaisabilité (problème SAT), les algorithmes de type CDCL sont basés sur la procédure DPLL [6]. Le principe consiste à choisir de manière heuristique la prochaine variable à instancier, de l'affecter à vrai (ou à faux) et de tester si, après simplifications, il existe un modèle sous cette hypothèse. En cas de conflit, une clause est apprise et un retour arrière (le plus souvent non chronologique) est effectuée afin de tester l'autre branche de l'affectation. Cette règle de séparation (un littéral et son opposé) peut être généralisée à une formule quelconque à partir du moment où :

— la négation de cette formule peut se calculer facilement (par rapport au formalisme cnf)
— il est possible de faire des déductions (littéraux impliqués)
— la complexité n'est pas supérieure à celle actuelle pour le problème SAT.

En nous appuyant dans un premier temps sur le codage direct d'un CSP en SAT, nous proposons un algorithme de résolution de la formula SAT obtenue d'une meilleure complexité théorique. Ceci nous permet par la même occasion d'avoir une meilleure complexité pour un algorithme de résolution de CSP.

**Abstract**

In this article, we study a new way of parsing the search space when solving CSP encoded in SAT (using direct encoding and support encoding). We exploit a different separating rule during the exploration of the search tree than in classical solvers. Usually, for solving the satisfiability problem (SAT problem), the CDCL based algorithms are based on the DPLL procedure [6]. They consist in choosing the next variable to instantiate thanks to a heuristic, then to assign it to true (or false) and check if, after some simplifications via unit propagation, there exists a model or not under this hypothesis. If a conflict arises, a clause is learned and a backtrack (usually non-chronological, i.e. a backjump) is operated in order to check the other branch of the search tree. This separating rule (one literal and its opposite) can be generalized to any formula, as far as:

— its negation is easily calculated (with respect to the CNF formalism)
— it is possible to make some decisions (implied literals)
— the complexity isn't higher than the actual SAT problem complexity.

Using, for the beginning, the direct encoding of a CSP in SAT instance, we propose a resolution algorithm for the resulting SAT formula with a better theoretical complexity. This allows us to reach a better complexity for solving CSP too.

## 1 Introduction

Le problème de satisfaisabilité (SAT) et le problème de satisfaction de contraintes (CSP) sont deux problèmes assez proches. D'un côté, SAT consiste à déterminer si une formule booléenne mise sous la forme normale conjonctive est satisfaisable. D'autre part, les problèmes CSP sont représentés par un ensemble de variables prenant leurs valeurs dans un ensemble fini de valeurs et liées par des contraintes. Chaque contrainte peut être décrite soit en terme de tuples autorisés ou interdits. Résoudre un CSP consiste à trouver une instanciation des variables satisfaisant toutes les contraintes

du problème.

Ces deux formalismes sont largement utilisés en intelligence artificielle. Ils permettent de modéliser et de résoudre aussi bien des problèmes académiques (problème des pigeons, problème des dames) que des problèmes issus du monde réel et industriels (vérification formelle, planning, etc.)

L'efficacité de ces solveurs est en grande partie due à l'exploitation de propriétés structurelles naturellement présentes dans le codage des problèmes comme par exemple les portes logiques (ou plus généralement fonction booléennes), les symétries, les équivalences de variables, etc.

Dernièrement, un problème mathématique [10] resté ouvert jusqu'à présent a pu être résolu en fournissant un contre exemple en utilisant un solveur SAT. La taille de la preuve, qui a été vérifiée, faisait tout de même près de 200To.

Bien qu'il soit possible d'exprimer n'importe quel problème de contraintes dans différents formalismes, il arrive souvent qu'un formalisme soit plus adapté qu'un autre. Par exemple, il est plus facile de considérer un problème de coloration de graphe sous la forme d'un CSP plutôt que de le formaliser sous un ensemble de formules propositionnelles. Cependant de nombreux travaux se sont focalisés sur des transformations de CSP vers SAT. De Kleer[13] a introduit le codage direct alors que Kasif[12] proposa le codage des supports (AC) de CSP binaires permettant de maintenir la propriété d'arc consistance par propagation unitaire pour SAT. Finalement, Bessière *et al.* [4] a généralisé le codage des supports ($k$-AC) aux CSP non binaires.

Dans, cet article, nous nous intéressons à une généralisation de la règle de séparation couramment utilisée dans les solveurs SAT de type CDCL pour les codages directs et supports de problèmes CSP. Le principe étant d'effectuer un parcours arborescent en choisissant comme point de choix un littéral, de simplifier la formule et de réitérer ce processus jusqu'à ce qu'une solution soit trouvée ou bien qu'un conflit n'apparaisse. Dans ce dernier cas, une clause conflit est apprise, permettant d'effectuer un retour arrière non chronologique et le plus souvent d'impliquer le littéral opposé.

Dans un premier temps, nous rappelons quelques définitions concernant les formalismes SAT et CSP. Nous nous appuyons ensuite sur le codage direct d'un CSP afin de présenter une manière différente de parcourir l'arbre de recherche permettant de fournir un algorithme complet de même complexité que l'algorithme stochastique proposé dans [16].

Les premiers résultats expérimentaux, sont ensuite présentés, et des connexions avec le formalisme CSP sont mis en évidence.

Finalement, nous concluons par la présentation de quelques perspectives à ce travail.

## 2 Définitions préliminaires

### 2.1 Le problème de satisfaisabilité (SAT)

Soit $\mathscr{B}$ un langage booléen (*i.e.* propositionnel) de formules formées de manière standard, en utilisant les connecteurs usuels ($\vee$,$\wedge$, $\neg$, $\Rightarrow$, $\Leftrightarrow$) et un ensemble de variables propositionnelles. Une formule *CNF* $\Sigma$ est un ensemble (interprété comme une conjonction) de *clauses*, où chaque clause est un ensemble (interprété comme une disjonction) de *littéraux*. Un littéral est une occurrence d'une variable propositionnelle soit sous forme positive, soit sous forme négative.

Rappelons que toute formule booléenne peut se réécrire sous la forme d'une *CNF* en temps linéaire en utilisant la transformation de Tseitin [21]. La taille d'une formule *CNF* $\Sigma$ est définie par $\sum_{c\in\Sigma}|c|$ où $|c|$ représente le nombre de littéraux de $c$.

Une *interprétation* d'une formule booléenne est une affectation de ses variables à une valeur de vérité $\{true, false\}$. Un littéral $l$ est satisfait (resp. falsifié) pour $I$ si $l$ est positif et que $I[l] = vrai$ ou $l$ est négatif et $I[l] = faux$ (resp. $l$ est négatif et $I[l] = faux$ ou $l$ est positif et $I[l] = vrai$). Un *modèle* pour une formule est une interprétation qui satisfait la formule. Le problème SAT est le problème de décision de la satisfaisabilité d'une *CNF*.

### 2.2 Le problème de satisfaction de contraintes (CSP)

Un CSP est un quadruplet $P = (X, D, C, R)$ où $X = \{X_1, X_2, \ldots, X_n\}$ est un ensemble de $n$ variables, $D = \{D_1, D_2, \ldots, D_n\}$ est un ensemble de domaines finis discrets ($D_i$ est le domaine des valeurs possibles pour $X_i$), $C = \{C_1, C_2, \ldots, C_m\}$ est un ensemble de $m$ contraintes où la contrainte $C_i$ est définie sur un sous-ensemble de variables $\{X_{i_1}, X_{i_2}, \ldots, X_{i_{a_i}}\} \subseteq X$. L'arité de la contrainte $C_i$ est $a_i$ et $R = \{R_1, R_2, \ldots, R_k\}$ est un ensemble de $k$ relations, où $R_i$ est la relation correspondant à la contrainte $C_i$. $R_i$ contient les combinaisons de valeurs interdites (les tuples) pour les variables impliquées dans la contrainte $C_i$. Un CSP binaire est un CSP dont les contraintes sont toutes d'arité deux (contraintes binaires). Un CSP est non-binaire s'il contient au moins une contrainte dont l'arité est supérieure à deux (une contrainte n-aire). Une *instanciation* $I$ est une affecta-

tion qui assigne à chaque variable $X_i$ une valeur de son domaine $D_i$. Une contrainte $C_i$ est satisfaite par une instanciation $I$ si la projection de $I$ sur les variables impliquées dans $C_i$ diffère de tous les tuples de la relation $R_i$. Une instanciation $I$ d'un CSP $P$ est consistant (ou est une solution de $P$) si elle satisfait toutes les contraintes de $P$. Un CSP $P$ est consistent si il admet au moins une solution ; sinon il est inconsistant. Dans la suite de l'article nous considérons que $n$ est le nombre de variables du CSP, $m$ son nombre de contraintes, $a$ l'arité maximale de ses contraintes et $d$ la taille du plus grand de ses domaines.

## 3 Transformations d'un CSP vers un problème SAT

L'idée de cet article est de proposer une règle de séparation plus générale que ce qui se fait actuellement pour les solveurs de type CDCL. Ces derniers s'appuient sur la proposition suivante :

**Proposition 1** *Soit $\Sigma$ une CNF et $v$ une variable propositionnelle. $\Sigma$ est satisfaisable si et seulement si $(\Sigma \wedge v)$ est satisfaisable ou $(\Sigma \wedge \neg v)$ est satisfaisable.*

En répétant cette propriété aux différentes variables de $\Sigma$, il est possible de construire un algorithme de type retour arrière afin de tester si une interprétation satisfait $\Sigma$.

Nous proposons de généraliser cette propriété de la façon suivante :

**Proposition 2** *Soit $\Sigma$ une CNF et $f$ une formule booléenne construite sur les variables propositionnelles de $\Sigma$. $\Sigma$ est satisfaisable si et seulement si $(\Sigma \wedge f)$ est satisfaisable ou $(\Sigma \wedge \neg f)$ est satisfaisable.*

La proposition 2 se distingue des travaux de [17] et de son algorithme [18] par le fait que l'auteur considère des ensembles orthonormés construits sur un sous ensemble de variables représentant une fonction d'ordre trois au minimum. Par exemple, il considère l'ensemble orthonormé $\{x, \neg x.y, \neg x.\neg y\}$ afin de déterminer les sous-problèmes à résoudre. De notre côté, nous ne considérons que la branche d'égalité entre les deux variables $x$ et $y$ d'un côté et celle de différence de l'autre.

De la même manière, en considérant successivement différentes fonctions booléennes (comme points de choix), nous pouvons aussi construire un algorithme de type retour arrière permettant cette fois ci de déterminer si l'ensemble des fonctions booléennes satisfait la formule de départ.

La proposition 1 peut se voir comme un cas particulier de la proposition 2 où la formule $f$ est réduite à un littéral.

Cependant, les fonctions choisies devront respecter certaines contraintes : (1) afin de rester dans le formalisme CNF, la négation de la formule doit se calculer facilement (et rapidement), (2) les formules choisies comme point de choix doivent permettre de simplifier $\Sigma$ (3) la complexité en terme du nombre de nœuds ne doit pas être supérieure à $2^n$ où $n$ est le nombre de variables de la formule $\Sigma$.

La généralisation de la règle de séparation aux formules quelconques pour le problème SAT en général sont des travaux en perspectives. En effet de nombreux points sont à éclaircir comme le stockage des points de choix, l'apprentissage, le backjumping, les restarts, les heuristiques afin d'obtenir des résultats expérimentaux pouvant concurrencer les meilleurs solveurs CDCL actuels.

Nous allons dans un premier temps nous restreindre à la résolution de CSP en appuyant notre démarche sur les deux transformations de bases à savoir le codage direct et le codage des supports.

### 3.1 Le codage direct

C'est le plus utilisé et le plus ancien des codages. Il fut introduit par De Kleer [13]. Dans ce codage, une variable propositionnelle est utilisée pour chaque valeur du domaine de chaque variable du CSP. Par exemple, une variable CSP $X$ avec pour domaine $D_X = \{v_1, v_2, \ldots, v_k\}$ définit $k$ variables booléennes : $x_{v_1}, x_{v_2}, \ldots, x_{v_k}$. Interpréter $x_{v_i}$ à *vrai* signifie que la variable CSP $X$ est affectée à la valeur $v_i$. Ces variables propositionnelles apparaissent dans trois types de clauses :

**Les clauses de type *au-moins-un* :** Il y a une clause de type *au-moins-un*, par variable. Elles codent les domaines du CSP, exprimant le fait que chaque variable doit recevoir une valeur de son domaine. Considérons l'exemple d'une variable CSP $X$ avec un domaine $D_X = \{v_1, v_2, v_3\}$. La clause $c_X = x_{v_1} \vee x_{v_2} \vee x_{v_3}$ est ajoutée pour coder le domaine $D_X$. Ces clauses seront notées clauses *a.m.u.* par la suite.

**Les clauses de type *au-plus-un* :** Il y a une clause de type *au-plus-un* pour chaque couple de valeurs du domaine de chaque variable. Ce sont des clauses binaires codant le fait que chaque variable du CSP ne peut recevoir plus d'une valeur de son domaine(ou *clause d'exclusion mutuelle*). Reprenons l'exemple précédent. Pour le domaine

$D_X = \{v_1, v_2, v_3\}$, il faut ajouter les 3 clauses binaires suivantes : $\neg x_{v_1} \vee \neg x_{v_2}$, $\neg x_{v_2} \vee \neg x_{v_3}$ et $\neg x_{v_1} \vee \neg x_{v_3}$. Ces clauses seront notées clauses *a.p.u.* par la suite. Il est en général possible de s'affranchir de cet ensemble de clauses. En effet, si un modèle affectant plusieurs valeurs à une même variable CSP est trouvé, il est possible d'en choisir une seule aléatoirement et d'ignorer les autres. Une autre façon de voir que l'on peut éventuellement s'affranchir de ces clauses a.p.u est de voir que ce sont des clauses bloquées [14]. Une clause $c$ est bloquée s'il existe un littéral $l$ dans cette clause tel que toutes les résolvantes produites sur ce littéral sont tautologiques. La suppression des clauses bloquées préserve la satisfaisabilité de la formule : elle ne participent pas à la preuve de l'inconsistance de la formule.

**Les clauses de conflits :** Il y a une clause de conflit pour chaque *tuple interdit* de chaque contrainte du CSP. Elles expriment les contraintes en codant toutes les combinaisons interdites par celles-ci. Par exemple, considérons une contrainte entre trois variables CSP $X, Y$ et $Z$ et $[u \in D_X, v \in D_Y, w \in D_Z]$ un tuple interdit par la contrainte $C_{XYZ}$ (c'est-à-dire que $[u, v, w] \notin R_{XYZ}$). La clause de conflit $c_{XYZ} = \neg x_u \vee \neg y_v \neg z_w$ est ajoutée pour interdire l'affectation simultanée de $X$ à $u$,$Y$ à $v$ et $Z$ à $w$.

Soit $\mathcal{P}$ un CSP et $\Sigma_{Dir}$ la formule CNF résultant du codage direct de $\mathcal{P}$. Walsh a prouvé qu'effectuer un filtrage par consistance d'arc sur $\mathcal{P}$ est plus fort que d'effectuer la propagation unitaire sur $\Sigma_{Dir}$. Il a également démontré, comme cela l'avait été par Génisson et Jégou [8] que moyennant des heuristiques de branchement équivalentes, FC appliqué à $\mathcal{P}$ et DP appliqué à $\Sigma_{Dir}$ étaient équivalents.

La complexité du codage direct d'un CSP est en $\mathcal{O}(md^a)$, en rappelant que $m$ représente le nombre de contraintes du CSP, $a$ la plus grande arité des contraintes du CSP et $d$ la taille du plus grand domaine du CSP. Plus précisément, dans le codage direct d'un CSP en CNF, nous retrouvons $n$ clauses *a.m.u.* de taille $d$. Chaque variable CSP ajoute $\frac{d(d-1)}{2}$ clauses *a.p.u.* de taille 2. Enfin chaque contrainte peut contenir au plus $d^a$ tuples interdits de longueur $a$. Ce qui donne une complexité totale de $(nd + 2n\frac{d(d-1)}{2} + mad^a) \equiv \mathcal{O}(md^a)$.

### 3.2 Le codage des supports

Le codage des supports fut introduit par Kasif [12] et spécialement conçu pour les CSP binaires. Dans ce codage, nous disposons des même variables propositionnelles que pour le codage direct, mais également des même clauses *a.m.u.* et *a.p.u.*. La seule différence réside dans les clauses de conflits qui sont remplacées par des clauses appelées clauses de supports.

**Les clauses de supports :** Une clause de support est ajoutée pour chaque couple (*valeur, liste de supports*) de chaque contrainte binaire. Elle code le fait que tant qu'une *valeur* reste dans le domaine d'une des variables impliquées dans la contrainte, alors au moins un de ces supports doit rester dans le domaine de la seconde variable impliquée. Par exemple, considérons $X$ et $Y$ deux variables CSP et le couple $(v \in D_X, \{s_1, s_2, \ldots, s_k\} \in D_Y)$ où $v$ est une valeur du domaine de $X$ et $\{s_1, s_2, \ldots, s_k\}$ sont les supports de $X = v$ dans le domaine de $Y$ pour la contrainte $C_{XY}$. La clause $\neg x_v \vee y_{s_1} \vee y_{s_2} \vee \ldots \vee y_{s_k}$ est rajoutée pour exprimer le fait que tant que $v$ n'est pas retiré (filtré) du domaine de $X$, au moins un de ses supports dans $D_Y$ ne doit pas être supprimé. Cette clause est équivalente à $x_v \rightarrow (y_{s_1} \vee y_{s_2} \vee \ldots \vee y_{s_k})$. Si tous les supports sont retirés (falsifiés), $\neg x_v$ est impliqué, c'est-à-dire que $v$ est retirée du domaine de $X$ lors d'un filtrage par consistance d'arc.

L'intérêt de ce codage pour les CSP binaires par rapport au codage direct est qu'il permet de récupérer la consistance d'arc dans la CNF. Soit $\mathcal{P}$ un CSP et $\Sigma_{Sup}$ la formule CNF résultant du codage des supports de $\mathcal{P}$. Kasif a montré que la propagation unitaire appliquée à $\Sigma_{Sup}$ est équivalente à un filtrage par consistance d'arc sur $\mathcal{P}$. Cela a permis à Gent [9] de montrer que moyennant des heuristiques de branchement équivalentes, DP appliqué à $\Sigma_{Sup}$ est équivalent à MAC appliqué à $\mathcal{P}$.

La complexité du codage des supports d'un CSP est en $\mathcal{O}(nd^2 + md^2)$. Précisément, nous retrouvons les même clauses *a.m.u.* et *a.p.u.* $(nd^2)$, et pour chaque clause, il y a $2d$ couples (*valeur, liste de supports*) possibles, chaque liste pouvant contenir au plus $d$ valeurs $(2md^2)$. La complexité totale est donc de $(nd^2 + 2md^2) \equiv \mathcal{O}((n + m)d^2)$.

## 4 Un parcours différent de l'arbre de recherche

Dans cette partie, en nous appuyant dans un premier temps sur le codage direct d'un CSP, nous proposons un parcours différent de l'arbre de recherche. Pour se faire, par rapport à la proposition 2 de la page 3, nous utilisons comme formule $f$ une formule de différence entre deux variables propositionnelles du codage direct. Cependant le choix de ces deux variables devra se faire dans une clause *a.m.u.* Reprenons l'exemple d'une variable CSP $X$ avec un domaine $D_X = \{v_1, v_2, v_3, v_4\}$. La clause $c_X = x_{v_1} \vee x_{v_2} \vee x_{v_3} \vee x_{v_4}$ représente la clause *a.m.u* pour coder le domaine $D_X$. En choisissant par exemple, comme point de choix, une formule du type $f = (x_{v_1} \neq x_{v_2})$, nous supposons l'existence d'un modèle où soit $x_{v_1}$ est à vrai soit $x_{v_2}$ est à vrai. Cela implique pour les autres variables propositionnelles de la clause *a.m.u* (i.e. $x_{v_3}$ et $x_{v_4}$) d'avoir la valeur fausse. Nous verrons un peu plus tard comment détecter cette valeur de vérité dans l'algorithme 1. Nous réitérons cette règle de séparation à d'autres couples de variables propositionnelles, toujours en les choisissant dans des clauses *a.m.u.* S'il n'existe pas de solution sous l'hypothèse $f = (x_{v_1} \neq x_{v_2})$, il faut tester l'autre branche à savoir $\neg f$. Cette négation devient la formule $\neg f = (x_{v_1} = x_{v_2})$. Comme ces deux variables font partie d'une clause *a.m.u* nous en déduisons que $x_{v_1}$=faux et $x_{v_2}$=faux. Le prochain point de choix, par rapport à l'exemple, consistera à prendre comme hypothèse $f = (x_{v_3} \neq x_{v_4})$.

Le parcours effectué ici est assez proche d'un algorithme consistant à séparer par dichotomie le domaine d'une variable du CSP en deux tout en effectuant de l'arc consistance avant de séparer à nouveau les sous domaines. Ainsi si une variable CSP $X$ a pour domaine $D_X = \{v_1, v_2, v_3, v_4, v_5, v_6\}$ sépare le domaine en deux en considérant les deux sous domaine $\{v_1, v_2, v_3\}$ et $\{v_4, v_5, v_6\}$. Le processus est répété tant que la taille des sous domaines est au moins égale à 2. Un autre travail sur le partitionnement des domaines est utilisé dans [15].

Notre algorithme se distingue des travaux ci-dessus par le fait qu'on ne partitionne pas en dessous d'une taille de domaine au moins égale à 2. En effet, supposons qu'il n'existe aucune contraintes entre 3 variables $X$ $Y$ et $Z$ d'un CSP avec $D_X = \{x_1, x_2\}$, $D_Y = \{y_1, y_2\}$ et $D_Z = \{z_1, z_2\}$. Le partitionnement des domaines essayera successivement les affectations $\{x_1, y_1, z_1\}$,$\{x_1, y_1, z_2\}$,...,$\{x_2, y_2, z_2\}$. Si le sous problème restant n'est pas consistant, il aura été effectué $2^3$ affectations. Dans notre cas, en prenant comme points de choix $(x_1 \neq x_2)$,$(y_1 \neq y_2)$ et $(z_1 \neq z_2)$ une seule affectation suffira.

D'un point de vue CSP, le parcours que nous effectuons en considérant des contraintes de différences sur les valeurs des domaines, consiste à construire de manière incrémentale un CSP binaire bivalent. Des algorithmes polynomiaux existent afin de déterminer si un tel CSP admet une solution. D'un point de vue SAT, pour le codage direct, ce parcours consiste à construire incrémentalement une formule 2-SAT qui est aussi une classe polynomiale.

La question que l'on doit se poser ici est : faut-il à chaque point de choix vérifier si le CSP ou la formule 2-SAT est inconsistant ?

Dans notre cas, pour le codage direct, déterminer si une formule 2-SAT est inconsistante peut se faire soit en calculant les composantes fortement connexes par l'algorithme de Tarjan [20] soit en montrant qu'il existe une variable $v$ de la formule 2-SAT qui soit inconsistante en affectant $v$ à vrai ou en l'affectant à faux.

Prenons par exemple la formule 2-SAT suivante $\Sigma = \{(\neg a \vee b) \wedge (\neg b \vee c) \wedge (\neg c \vee a)\}$. Le graphe des composantes fortement connexes est donné à la figure 1. Il n'existe pas de chemin reliant un littéral et son opposé dans les deux sens. La formule est donc satisfaisable. Maintenant si nous prenons l'hypothèse $(a \neq b)$, le graphe devient celui de la figure 2. Dans ce graphe, il existe un chemin de $a$ vers $\neg a$ et un chemin de $\neg a$ vers $a$. Sous cette hypothèse la formule est donc insatisfaisable.

**Remarque 1** *De manière générale, c'est la dernière hypothèse (ou point de choix) qui construit des ponts bidirectionnels entre les composantes fortement connexes.*

Nous pouvons en déduire la proposition suivante :

**Proposition 3** *Soit $\Sigma$ une formule 2-SAT consistante, et une contrainte de différence $(l_1 \neq l_2)$ représentant notre point de choix. La formule $(\Sigma \wedge (l_1 \neq l_2))$ est insatisfaisable si et seulement si $(\Sigma \wedge l_1 \wedge \neg l_2)$ est insatisfaisable et $(\Sigma \wedge \neg l_1 \wedge l_2)$ aussi.*

L'utilisation de la propagation unitaire ([3]) dans ce cas suffit à déterminer la satisfaisabilité de la formule.

Dans le cas où la propagation unitaire est contradictoire sur une des deux branches, l'autre branche est impliquée. S'il n'y a pas de contradiction sur les deux branches, nous profitons du travail effectué de la propagation unitaire afin d'inférer des littéraux impliqués des deux côtés. Ce travail (de littéraux impliqués) constituera la méthode de filtrage de notre algorithme.

Si nous prenons par exemple la formule $\Sigma = \{(x_{v1} \vee x_{v2} \vee x_{v3}) \wedge (\neg x_{v1} \vee \neg x_{v2}) \wedge (\neg x_{v1} \vee \neg x_{v3}) \wedge (\neg x_{v2} \vee \neg x_{v3})\}$ représentant une clause de domaine d'une variable d'un CSP. Sous l'hypothèse $(x_{v1} \neq x_{v2})$, le littéral $\neg x_{v3}$ est impliqué.

Le filtrage dans les domaines des autres variables se fera aussi par détection de littéraux impliqués. Si par exemple nous avons la formule $\Sigma = \{(x_{v1} \vee x_{v2} \vee x_{v3}) \wedge (\neg x_{v1} \vee \neg x_{v2}) \wedge (\neg x_{v1} \vee \neg x_{v3}) \wedge (\neg x_{v2} \vee \neg x_{v3}) \vee (y_{v1} \vee y_{v2} \vee y_{v3}) \wedge (\neg y_{v1} \vee \neg y_{v2}) \wedge (\neg y_{v1} \vee \neg y_{v3}) \wedge (\neg y_{v2} \vee \neg y_{v3}) \vee (\neg x_{v1} \vee \neg y_{v1}) \vee (\neg x_{v2} \vee \neg y_{v1}))\}$. Alors, sous l'hypothèse $(x_{v1} \neq x_{v2})$, il sera possible d'impliquer le littéral $\neg x_{v3}$ mais aussi $\neg y_{v1}$.

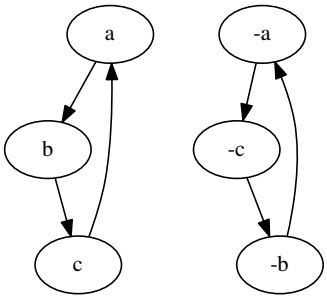



FIGURE 1 – graphe des composantes fortement connexes

L'algorithme 1 résume notre approche. Il prend en entrée une formule CNF issue du codage direct d'un problème CSP. A chaque étape, une clause de domaine est choisie (ligne 1). S'il n'y en a plus de disponible, l'algorithme s'arrête et retourne satisfaisable. Dans le cas contraire, deux variables non affectées sont choisies afin de constituer notre point de choix (ligne 6). Si une seule variable est disponible elle est propagée et le parcours continue (ligne 8). En cas de contradiction un retour arrière est effectué. Si deux variables non affectées sont disponibles (par exemple $x$ et $y$), le point de choix devient $(x \neq y)$.On teste ensuite (ligne 11) la propagation de $\{x,\neg y\}$ d'un côté et $\{\neg x,y\}$ de l'autre. Si les deux branches sont contradictoires par cette propagation, nous inférons directement $\{\neg x, \neg y\}$. Dans le cas contraire, la propagation unitaire des littéraux communs calculés précédemment est effectuée et le parcours de l'arbre se poursuit (ligne 17).

**Remarque 2** *D'un point de vue CSP, notre algorithme consiste à restreindre itérativement les tailles des domaines des variables à 2 tout en effectuant un filtrage. En cas de contradiction, deux autres valeurs du domaine de la dernière variable choisie sont testées.*

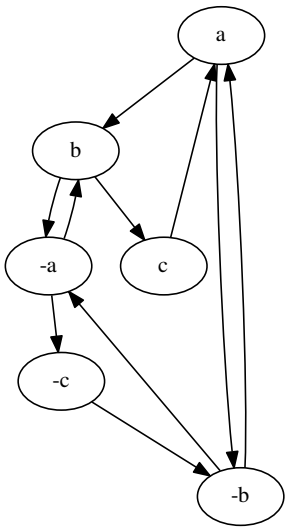



FIGURE 2 – graphe des composantes fortement connexes sous l'hypothèse $(a \neq b)$

**Remarque 3** *Concernant le codage des supports, le parcours effectué par notre algorithme fonctionne de manière identique et permet d'inférer plus de littéraux.En effet la propagation unitaire revient à faire de l'arc consistance. Finalement, nous n'avons pas besoin de tester la satisfaisabilité de la formule produite par les points de choix. C'est la propagation unitaire dans les deux branches qui va permettre de détecter cette inconsistance.*

**Proposition 4** *La complexité théorique de notre algorithme est la même qu'un algorithme CSP de complexité $\mathcal{O}((\frac{d}{2})^n \times coutFiltrage())$ où $d$ représente la taille du plus grand domaine et $n$ le nombre de variables.*

## 5 Résultats expérimentaux préliminaires

Nous présentons dans cette partie les premiers résultats préliminaires sur quelques problèmes afin de valider notre algorithme. Dans un premier temps, nous nous sommes intéressé au fameux problème des pigeons qui reste encore actuellement un challenge pour

**Algorithme 1** FCEquiv

**Require:** $\Sigma$
**Ensure:** Solve $\phi$

```
1: cla ← choose_clause_domain(Σ)
2: if (cla = null) then
3:    return ⊤
4: end if
5: while (true) do
6:    (x, y) ← choose_couple_litteral(cla)
7:    if (x ≠ null AND y = null) then
8:       solve(Σ, level + 1)
9:       return
10:   end if
11:   L ← test_and_filter(Σ, x, y)
12:   if (⊥ ∈ L) then
13:      if (BCP(Σ, {¬x, ¬y}) = ⊥ then
14:         return ⊥
15:      end if
16:   else
17:      return solve(Σ, level + 1)
18:   end if
19: end while
```

les solveurs CDCL car sans , par exemple, l'exploitation des symétries, ils deviennent assez vite difficiles à résoudre. Le tableau 1 synthétise ces résultats. La première colonne représente la taille du problème, puis nous comparons notre algorithme avec une version de Forward checking (sans apprentissage, backjumping, redémarrages, etc.) et de minisat [7]. Nous évaluons ces trois algorithmes sur le codage direct et le codage des supports. Pour chacun d'eux nous reportons le temps de résolution (donné en millisecondes) ainsi que le nombre de nœuds parcourus. Nous voyons rapidement qu'à partir du problème à 12 pigeons, minisat montre ses limites (le timeout étant fixé à 5 minutes) alors que notre algorithme le résout en un peu plus d'une minute. Pour ce type de problème, la résolution, en utilisant le codage des supports n'apporte pas grand chose en terme de filtrage. Cependant les temps de résolutions restent toujours en faveur de notre algorithme.

Dans un second temps, nous avons testé, toujours avec les mêmes algorithmes, des problèmes de coloration de graphe. Nous avons fixé le nombre de sommets à 20 et 30. Pour les problèmes à 20 sommets nous avons fait varier la dureté de 0.6 à 0.9 et avons fixé le nombre de couleurs en fonction de la dureté du problème (aux alentours du pic de difficulté). Pour la résolution des problèmes de coloration de graphes, nous avons fait en sorte que notre algorithme (et Forward checking), choisisse le prochain sommet possédant le plus petit nombre de couleurs disponibles (heuristique DSATUR[5]) et en choisissant les deux valeurs (couleurs) dans l'ordre de disponibilité. Si l'on ne tient pas compte cette dernière règle, le filtrage sera beaucoup moins important. Le tableau 2 synthétise les résultats. Notre algorithme, dans l'ensemble, se comporte plutôt bien comparativement à minisat. Par contre, le codage des supports pour les problèmes de coloration de graphe n'apportent pas grand chose. Ceci est probablement due au type de contraintes qui ne permet pas de filtrer plus.

## 6 Conclusion et perspectives

Dans cet article, nous avons proposé une généralisation de la règle de séparation habituellement utilisée pour les solveurs SAT de type CDCL. Au lieu de choisir comme règle de branchement un littéral et son opposé, nous avons considéré une différence et une égalité entre deux variables propositionnelles. L'idée, pour le moment, n'est pas de l'appliquer pour la résolution du problème SAT de manière générale. Nous avons restreint notre règle de séparation à des problèmes CNF issus des transformations de base d'un CSP vers SAT (codage direct et codage des supports). Comparativement, à un algorithme de type Forward Checking, cela revient à baisser la complexité théorique de $\mathcal{O}(d^n \times \text{coutFiltrage}())$ à $\mathcal{O}((\frac{d}{2})^n \times \text{coutFiltrage}())$ avec $d$ la taille du plus grand domaine et $n$ le nombre de variables. Les premiers résultats expérimentaux, bien que préliminaires, sont très encourageants. Le problème des pigeons peut se résoudre plus efficacement sans pour autant ajouter, par exemple un traitement sur la détection les symétries.

En perspective, il reste à implémenter de nombreuses techniques très utilisées dans les solveurs modernes comme l'apprentissage (couplée à la gestion du graphe d'implications et de clauses assertives), le backjumping, la gestion des contraintes apprises, etc. Concernant l'apprentissage par exemple, lorsqu'un conflit apparaît, il faudra être capable, de générer une contrainte assertive afin de se calquer sur le schéma actuel des solveurs CDCL. La contrainte de conflit sera du genre $(x_1 = x_2 \vee y_1 = y_2 \vee z_1 = z_2)$ qui générera un nombre exponentiel de clauses si le formalisme CNF est utilisé.

Nous pourrions aussi voir si notre règle de séparation s'applique à d'autres transformations de CSP vers SAT comme le codage logarithmique [11] ou bien le codage order ( [1],[2],[19]).

Il serait aussi intéressant de considérer d'autres règles de séparations respectant les trois conditions comme par exemple le *et* logique ou bien le *ou* logique entre certaines variables.

| | Codage direct | | | Codage supports | | |
|---|---|---|---|---|---|---|
| n | FCEquiv | FC | minisat | FCEquiv | FC | minisat |
| 7 | **3**(252) | 3(1438) | 6(1407) | 6(252) | 6(1438) | 7(1139) |
| 8 | **6**(1660) | 7(10078) | 24(7664) | 25(1660) | 39(10078) | 42(7411) |
| 9 | **27**(7004) | 63(80638) | 180(44545) | 125(7004) | 390(80638) | 307(40534) |
| 10 | **140**(60161) | 453(725758) | 6120(1000172) | 1301(60161) | 4643(725758) | 5959(525536) |
| 11 | **814**(313410) | 5111(7257598) | 124356(11828510) | 8707(313410) | 59995(7257598) | 119739(6277093) |
| 12 | **9256**(3322496) | 64081(79833598) | TIMEOUT | 113000(3322496) | TIMEOUT | TIMEOUT |
| 13 | **64720**(20615382) | TIMEOUT | TIMEOUT | TIMEOUT | TIMEOUT | TIMEOUT |

Table 1 – résultats sur le problème des pigeons

| | | | Codage direct | | | Codage supports | | |
|---|---|---|---|---|---|---|---|---|
| n | couleurs | densité | FCEquiv | FC | minisat | FCEquiv | FC | minisat |
| 20 | 6 | 0.6 | 0(2448) | 0(1438) | 0(1921) | 0.1(2448) | 0.1(1438) | 0.1(2163) |
| 20 | 7 | 0.7 | 0.18(30637) | 0.12(50398) | 0.18(24994) | 1.4(30637) | 1(50398) | 0.5(32776) |
| 20 | 9 | 0.8 | 7(1385001) | 14(3628798) | 42(201141) | 93(1385001) | 165(3628798) | 36(1200601) |
| 20 | 10 | 0.9 | 6.36(1386975) | 12(7257598) | 124(5380698) | 105(1386975) | 211(7257598) | 288(6256227) |
| 30 | 8 | 0.6 | 25(3165468) | 14(4677118) | 257(8979189) | 313(3165468) | 159(4677118) | 141(3027918) |
| 30 | 9 | 0.7 | 9(1252857) | 14(4354558) | 44(1754928) | 132(1252857) | 208(4354558) | 78(1490465) |

Table 2 – résultats sur des problèmes de coloration de graphe

# 7 Bibliographie

## Références

[1] James Crawford Andrew et Andrew B. Baker : Experimental results on the application of satisfiability algorithms to scheduling problems. *In In Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 1092–1097, 1994.

[2] Carlos Ansótegui et Felip Manyà : Mapping problems with finite-domain variables to problems with boolean variables. *In Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing*, SAT'04, pages 1–15, Berlin, Heidelberg, 2005. Springer-Verlag.

[3] Daniel Le Berre : Exploiting the real power of unit propagation lookahead. *Electronic Notes in Discrete Mathematics*, 9:59–80, 2001.

[4] C. Bessière, E. Hebrard et T. Walsh : Local consistency in SAT. *In International Conference on Theory and Application of satisfiability testing*, pages 400–407, 2003.

[5] Daniel Brélaz : New methods to color the vertices of a graph. *Commun. ACM*, 22(4):251–256, avril 1979.

[6] Martin Davis, George Logemann et Donald Loveland : A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[7] Niklas Eén et Niklas Sörensson : An extensible sat-solver. *In Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT2003)*, pages 502–518, 2003.

[8] Richard Génisson et Philippe Jégou : Davis and Putnam were already forward checking. *In ECAI'96 : Proceedings of the 12th European Conference on Artificial Intelligence*, pages 180–184, 1996.

[9] Ian P. Gent : Arc consistency in SAT. *ECAI'02 : Proceedings of the 15th European Conference on Artificial Intelligence*, pages 121–125, 2002.

[10] Marijn J. H. Heule, Oliver Kullmann et Victor W. Marek : Solving and verifying the boolean pythagorean triples problem via cube-and-conquer. *In Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, pages 228–245, 2016.

[11] Kazuo Iwama et Shuichi Miyazaki : Sat-variable complexity of hard combinatorial problems. *In In Proceedings of the world computer congress of the IFIP*, pages 253–258. ELSEVIER SCIENCE B.V, 1994.

[12] S. Kasif : On the Parallel Complexity of Discrete Relaxation in Constraint Satisafaction Networks. *Journal of Artificial Intelligence*, 45:275–286, 1990.

[13] J. De Kleer : A comparison of ATMS and CSP techniques. *In Proceedings of IJCAI'89*, pages 290–296, Detroit, 1989.

[14] Oliver Kullmann : On the use of autarkies for satisfiability decision. *Electronic Notes in Discrete Mathematics*, 9:231–253, 2001.

[15] Javier Larrosa : Merging constraint satisfaction subproblems to avoid redundant search. *In Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI 97, Nagoya, Japan, August 23-29, 1997, 2 Volumes*, pages 424–433, 1997.

[16] Uwe Schöning : A probabilistic algorithm for k-sat and constraint satisfaction problems. *In Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, FOCS '99, pages 410–, Washington, DC, USA, 1999. IEEE Computer Society.

[17] Virendra Sule : Generalization of boole-shannon expansion, consistency of boolean equations and elimination by orthonormal expansion. *CoRR*, abs/1306.2484, 2013.

[18] Virendra Sule : An algorithm for boolean satisfiability based on generalized orthonormal expansion. *CoRR*, abs/1406.4712, 2014.

[19] Naoyuki Tamura, Akiko Taga, Satoshi Kitagawa et Mutsunori Banbara : Compiling finite linear csp into sat. *Constraints*, 14(2):254–272, juin 2009.

[20] Robert Endre Tarjan : Depth first search and linear graph algorithms. *SIAM J. Comput.*, 1:146–160, 1972.

[21] G.S. Tseitin : On the complexity of derivations in the propositional calculus. *In* H.A.O. Slesenko, éditeur : *Structures in Constructives Mathematics and Mathematical Logic, Part II*, pages 115–125, 1968.

# Sur la pertinence des décompositions arborescentes optimales pour la résolution de CSP*

**Philippe Jégou**[1] **Hélène Kanso**[2] **Cyril Terrioux**[1]

[1] Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
[2] Effat University, Jeddah, Arabie Saoudite
{philippe.jegou, cyril.terrioux}@lis-lab.fr hkanso@effatuniversity.edu.sa

**Résumé**

La notion de décomposition arborescente est un sujet important pour l'étude et la résolution de problèmes NP-difficiles en intelligence artificielle, et notamment en programmation par contraintes. D'un point de vue théorique, son exploitation, dans le cadre général des modèles graphiques (réseaux bayésiens, CSP (pondérés), ...), a conduit, sous certaines hypothèses, à la définition d'algorithmes de résolution polynomiaux. Par ailleurs, ces dernières années, des solveurs l'exploitant ont fait montre de leur intérêt pratique pour la résolution de problèmes de décision, d'optimisation ou de comptage.

Dans cet article, nous nous intéressons aux décompositions arborescentes optimales et à leur intérêt pratique pour la résolution d'instances CSP. La motivation de ce travail est double. D'une part, des progrès considérables ont été accomplis au niveau des méthodes calculant de telles décompositions, rendant envisageable leur exploitation pratique. D'autre part, la complexité temporelle des méthodes de résolution exploitant une décomposition arborescente est directement liée à la largeur de la décomposition employée et donc employer une décomposition optimale permet théoriquement de minimiser cette complexité. Nous évaluons d'abord la capacité des méthodes calculant des décompositions optimales à décomposer des instances CSP avant de mesurer l'apport de ces décompositions optimales au niveau de l'efficacité de la résolution.

Ce papier est un résumé de [2].

## 1 Contexte

Une instance CSP (pour Problème de Satisfaction de Contraintes) est définie par la donnée d'un triplet $(X, D, C)$, où $X = \{x_1, \ldots, x_n\}$ est un ensemble de $n$ variables, $D = \{d_{x_1}, \ldots, d_{x_n}\}$ est un ensemble de domaines finis de taille au plus $d$, et $C = \{c_1, \ldots, c_e\}$ est un ensemble de $e$ contraintes. Chaque contrainte $c_i$ est un couple $(S(c_i), R(c_i))$, où $S(c_i) = \{x_{i_1}, \ldots, x_{i_k}\} \subseteq X$ définit la *portée* de $c_i$, et $R(c_i) \subseteq d_{x_{i_1}} \times \cdots \times d_{x_{i_k}}$ est une *relation de compatibilité*. La structure d'une instance CSP est donnée par un hypergraphe, appelé *hypergraphe de contraintes*, dont les sommets correspondent aux variables et les arêtes aux portées des contraintes. Une affectation d'un sous-ensemble de $X$ est dite *cohérente* si toutes les contraintes portant sur ce sous-ensemble sont satisfaites. Une *solution* est une affectation cohérente de toutes les variables.

Déterminer si une instance CSP possède une solution est un problème NP-complet. Il en découle que les algorithmes de résolution habituellement employés ont une complexité en $O(exp(n))$. Toutefois, certains algorithmes, comme BTD [3], sont capables de fournir de meilleures bornes de complexité temporelle en exploitant une *décomposition arborescente* [4] de l'hypergraphe de contraintes pour identifier des sous-problèmes indépendants.

**Définition 1** *Une* décomposition arborescente *d'un graphe $G = (X, C)$ est un couple $(E, T)$ où $T = (I, F)$ est un arbre ($I$ est un ensemble de nœuds et $F$ un ensemble d'arêtes) et $E = \{E_i : i \in I\}$ une famille de sous-ensembles de $X$, telle que chaque sous-ensemble (appelé cluster) $E_i$ est un nœud de $T$ et vérifie : (i) $\cup_{i \in I} E_i = X$, (ii) pour chaque arête $\{x, y\} \in C$, il existe $i \in I$ avec $\{x, y\} \subseteq E_i$, et (iii) pour tout $i, j, k \in I$, si $k$ est sur un chemin de $i$ à $j$ dans $T$, alors $E_i \cap E_j \subseteq E_k$. La largeur d'une décomposition est égale à $max_{i \in I}|E_i| - 1$. La largeur arborescente dite* treewidth *$w^*$ de $G$ est la largeur minimale pour toutes les décompositions arborescentes de $G$.*

*Ce travail est soutenu par l'Agence Nationale de la Recherche dans le cadre du projet DEMOGRAPH (ANR-16-C40-0028)

Leur complexité temporelle peut alors s'exprimer en $O(exp(w))$, avec $w$ la largeur de la décomposition arborescente employée ($w < n$), pour une complexité spatiale en $O(exp(s))$, avec $s$ la taille de la plus grande intersection entre deux clusters ($s \leq w$). D'un point de vue théorique, elle semble d'autant plus intéressante que $w$ est proche de $w^*$ et, à ce titre, le calcul d'une décomposition arborescente de petite largeur paraît constituer une étape importante en vue de la résolution efficace d'instances CSP.

## 2 Calculs de décompositions

De nombreuses méthodes de calculs de décompositions arborescentes ont été proposées dans la littérature. Nous pouvons les diviser en trois catégories : les méthodes exactes, les méthodes approchées et les méthodes heuristiques. Les méthodes exactes ont pour objectif de calculer des décompositions optimales (c'est-à-dire dont la largeur est $w^*$). Ce problème étant NP-difficile, elles ont une complexité en temps exponentielle, rendant généralement trop coûteuse leur utilisation. Toutefois, des progrès considérables ont été réalisés dans le cadre du défi PACE [1], rendant désormais possible la décomposition de graphes ayant plusieurs centaines de sommets contre quelques dizaines précédemment. Les méthodes approchées, elles, fournissent des garanties sur la qualité de la largeur obtenue. Malheureusement, à l'heure actuelle, elles s'avèrent trop coûteuses en temps, surtout vis-à-vis de la qualité des décompositions calculées. Enfin, les méthodes heuristiques permettent de calculer rapidement des décompositions arborescentes, mais n'offrent aucune garantie sur la qualité de la largeur obtenue.

## 3 Résultats expérimentaux

Nous comparons les méthodes exactes *larisch*, *tamaki* et *bannach* [1] à des méthodes heuristiques avec, d'une part, Min-Fill et MCS (qui visent à minimiser la largeur) et, d'autre part, Connected et Small-sep (dont l'objectif est de calculer des décompositions adaptées à la résolution d'instances CSP). Pour cela, nous avons considéré 7 597 instances CSP au format XCSP3.

Du point de vue de la capacité à décomposer, les méthodes heuristiques parviennent à décomposer quasiment toutes les instances. Si les méthodes exactes ne font pas aussi bien dans le temps imparti (1 800 s par instance), elles réussissent tout de même à décomposer au moins 75 % des instances. De plus, grâce aux largeurs optimales ainsi calculées, nous avons pu établir que Min-Fill produit des décompositions dont la largeur est souvent proche de l'optimum.

Maintenant, si nous nous intéressons à l'efficacité de

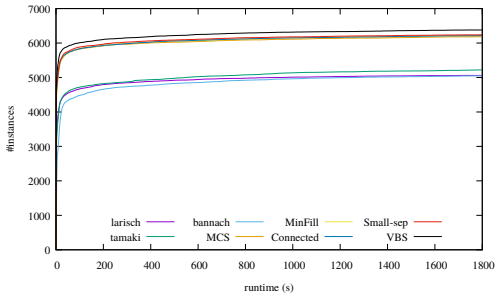



FIGURE 1 – Nombre cumulé d'instances résolues.

la résolution avec un algorithme comme BTD (figure 1), nous pouvons à nouveau constater que les méthodes heuristiques conduisent à obtenir de meilleurs résultats. Au niveau de ces heuristiques, Connected et Small-sep permettent à BTD de résoudre plus d'instances qu'avec Min-Fill ou MCS. Une des explications réside dans la valeur du paramètre $s$. En effet, nous avons pu observer que, pour Min-Fill et MCS comme pour les méthodes exactes, la valeur de $s$ est souvent proche, voire égale, de la largeur de la décomposition calculée. Si nous considérons uniquement les temps de résolution (c'est-à-dire sans compter le temps de décomposition), l'emploi de décompositions optimales peut se révéler pertinent pour certaines instances. Par contre, si nous prenons en considération le temps total, le calcul de décompositions optimales reste souvent encore trop coûteux pour permettre une résolution d'instances CSP efficace.

## 4 Conclusion

Contrairement à ce que peut laisser penser la théorie, l'emploi d'une décomposition optimale n'est pas le gage d'une résolution d'instances CSP plus efficace. Différents paramètres influent sur l'efficacité de la résolution, la largeur de la décomposition employée n'étant que l'un d'eux. Il est à noter que nous avons réalisé des observations similaires dans le cadre de la résolution d'instances WCSP ou du problème #CSP.

## Références

[1] H. DELL, C. KOMUSIEWICZ, N. TALMON et M. WELLER : The PACE 2017 Parameterized Algorithms and Computational Experiments Challenge : The Second Iteration. *In IPEC*, pages 30 :1–30 :12, 2018.

[2] P. JÉGOU, H. KANSO et C. TERRIOUX : On the Relevance of Optimal Tree Decompositions for Constraint Networks. *In Proceedings of ICTAI*, pages 738–743, 2018.

[3] P. JÉGOU et C. TERRIOUX : Hybrid backtracking bounded by tree-decomposition of constraint networks. *AIJ*, 146:43–75, 2003.

[4] N. ROBERTSON et P.D. SEYMOUR : Graph minors II : Algorithmic aspects of treewidth. *Algorithms*, 7:309–322, 1986.

# Heuristiques de recherche : un bandit pour les gouverner toutes

**Hugues Wattez**[1,2*] **Frédéric Koriche**[1,2]
**Christophe Lecoutre**[1,2] **Anastasia Paparrizou**[1] **Sébastien Tabary**[1,2]
[1] CRIL, UMR CNRS 8188, Lens, France
[2] Université d'Artois
{wattez,koriche,lecoutre,paparrizou,tabary}@cril.fr

**Résumé**

L'automatisation de la configuration des solveurs a reçu une grande attention ces dernières années notamment pour la capacité à ajuster ses différents paramètres selon l'instance à résoudre. De plus, cette automatisation évite à l'utilisateur le choix d'une configuration appropriée à chaque instance et ainsi, évite une tâche requérant des qualités d'expert et rend plus accessible les outils de la programmation par contraintes. Les techniques basées sur les portfolios ont été grandement étudiées dans le but de configurer les solveurs de contraintes. Son principe est d'entraîner le solveur à résoudre plusieurs instances afin de trouver la meilleure configuration lors de la résolution d'une instance inconnue. Dans cette étude, nous portons notre attention sur le choix de la meilleure stratégie de recherche, à savoir la meilleure heuristique de choix de variables. Nous proposons une structure algorithmique générique qui optimise la sélection des différentes heuristiques de choix de variables grâce à l'apprentissage en ligne, c'est-à-dire sans aucune connaissance apriori. Dans ce but, nous utilisons une technique en apprentissage, appelée les bandits multi-bras, permettant d'automatiser la sélection de l'heuristique la plus appropriée et guidant au mieux la recherche. Nous essayons deux politiques différentes basées sur les bandits multi-bras gérant la proportion d'exploration-exploitation. Une évaluation expérimentale démontre que la machine proposée résulte en un solveur plus efficace et stable.

**Abstract**

Automating solver's configuration has received a great attention last years as it can adjust the various parameters of a solver to the instance to be solved. In addition, this automation releases the user from choosing the suitable configuration each time, task that requires a great expertise and thus, burdens the widespread of constraint programming technology and tools. Portfolio based techniques have been vastly used to configure constraint solvers. The principle of portfolios is to train themselves by solving many instances in advance , in order to find the right combination of parameters of the solver when solving an unknown instance. In this work, we turn our attention to the selection of the best search strategy, namely the best variable ordering. We propose a generic algorithmic framework that selects among several variable ordering heuristics using on-line learning, i.e. without any prior knowledge. For this purpose, we deploy a machine learning technique, called multi-armed bandits, that allows to automatically select the appropriate heuristic to guide the search. We tried two different policies for multi-armed bandits that manage the proportion of exploration-exploitation. We make a thorough experimental evaluation that demonstrates that the proposed framework results in a more efficient and stable solver.

## 1 Introduction

Les solveurs de contraintes sont équipés de plusieurs composants génériques dont le choix minutieux ou l'omission peut drastiquement améliorer leur efficacité. Un point important dans le choix d'une bonne configuration concerne la sélection de l'heuristique de choix de variables. Une heuristique savamment choisie permet de réduire significativement la taille de l'espace de recherche parcouru. Mis à part pour un expert, sélectionner une heuristique n'est pas une chose aisée. De plus, étant donnée leur nature, aucune heuristique ne peut être continuellement meilleure qu'une autre. Cela induit qu'utiliser une seule et même heuristique pour un large évantail de problèmes peut être désavantageux quant à l'efficacité de la recherche.

Les travaux que nous menons tendent à remplacer le

*Papier doctorant : Hugues Wattez[1,2] est auteur principal.

rôle de l'humain par un mécanisme intelligent permettant d'orienter automatiquement le solveur au cours de la recherche. A chaque *run*, le solveur décide quelle heuristique choisir. C'est un problème de décisions séquentielles et en tant que tel, il peut être modélisé comme un problème de bandit multi-bras (MAB). En apprentissage par renforcement, la proportion entre exploration et exploitation est spécifiée par différentes politiques. Nous nous concentrons sur epsilon-greedy [16] et upper confidence bound [1] (ucb). Le bandit utilise une fonction de récompense lui permettant de converger vers le meilleur choix. L'apprentissage se fait en ligne, sans l'usage de connaissances apriori. Il s'agit uniquement de la maximisation des récompenses décernées. Ces récompenses sont basées sur le nombre de solutions inférées fausses durant un *run*.

Les expérimentations menées sur de nombreuses séries d'instances montrent que l'utilisation du bandit surpasse l'heuristique la plus efficace.

## 2 Travaux connexes

La propagation de contraintes est l'un des procédés permettant de réduire efficacement la taille de l'arbre de recherche. En pratique, il est connu que changer de niveau de propagation au cours de la recherche produit de meilleurs résultats que dans le cas statique. Dans ces circonstances, l'application d'un bandit permet de dynamiser ce choix au cours de la recherche sans pour autant faire appel à un expert ou fixer de paramètres. Le procédé appliqué dans [2] correspond au positionnement de différents bandits à différentes profondeurs de l'arbre de recherche. Ainsi, chaque bandit choisit le niveau de propagation le plus adéquat en fonction de sa position dans l'arbre, à l'aide de l'implantation d'ucb. La fonction de récompense est basée sur le temps nécessaire au renforcement de la cohérence locale. Plus le solveur progresse dans la recherche d'une solution, plus les bandits gagnent en connaissances et plus fines deviennent les décisions dans le choix du niveau de propagation optimal. Les résultats de cette étude montrent que cette approche apporte une meilleure efficacité et rend le solveur plus stable.

L'étude proposée dans [17] est une variante du sujet que nous traitons : l'usage des bandits pour le choix d'une heuristique de choix de variables. Dans cette étude, deux algorithmes issus des bandits stochastiques ont été implantés : ucb et Thomson sampling (ts). À chaque nœud de la recherche, une heuristique est choisie par l'algorithme et une récompense lui est ensuite décernée. Cette récompense est basée sur le nombre de nœuds parcourus dans le sous-arbre de recherche. Les conclusions de l'étude montrent que l'usage dynamique des différentes heuristiques d'un solveur le rend plus robuste et permet de meilleures performances que dans le cas d'un choix statique.

En résumé, l'usage des bandits dans le choix d'un niveau de propagation tout comme dans le choix d'une heuristique, évite de solliciter un expert pour fixer les valeurs des paramètres de contrôle du solveur, ce qui permet de garantir de meilleures performances.

## 3 Réseaux de contraintes

Un *réseau de contraintes* $P$ est composé d'un ensemble fini de variables $\texttt{vars}(P)$, et d'un ensemble fini de contraintes $\texttt{ctrs}(P)$. Nous utilisons $n$ pour définir le nombre de variables. Chaque variable $x$ prend une valeur dans un domaine fini, noté $\texttt{dom}(x)$. Chaque contrainte $c$ représente une relation mathématique $\texttt{rel}(c)$ associé à un ensemble de variables, appelé la portée (*scope*) de $c$, et noté $\texttt{scp}(c)$. L'*arité* d'une contrainte $c$ est la taille de sa portée. Le *degré* d'une variable $x$ est le nombre de contraintes dans $\texttt{ctrs}(P)$ impliquant $x$.

Une *solution* de $P$ correspond à l'affectation d'une valeur pour chaque variable de $\texttt{vars}(P)$ de façon à ce que toutes les contraintes dans $\texttt{ctrs}(P)$ soient satisfaites. Un réseau de contraintes est *cohérent* lorsqu'il admet au moins une solution. Le *problème de satisfaction de contraintes* (CSP pour *Constraint Satisfaction Problem*) consiste à déterminer si un réseau de contraintes donné est cohérent ou non. Une procédure classique pour résoudre ce problème NP-complet est d'effectuer une recherche avec retours-arrières dans l'espace des solutions partielles, et d'établir une propriété appelée l'*arc-cohérence généralisée* [11] après chaque décision. Cette procédure, appelée le *maintien de l'arc-cohérence* (MAC pour *Maintaining Arc Consistency*) [14], construit un arbre binaire de recherche $\mathcal{T}$ : pour chaque nœud interne $\nu$ de $\mathcal{T}$, un couple $(x, v)$ est sélectionné où $x$ est une variable non affectée et $v$ est une valeur dans $\texttt{dom}(x)$. Ainsi, nous considérons deux cas : l'affectation $x = v$ (décision dite positive) et la réfutation $x \neq v$ (décision dite négative).

L'ordre dans lequel les variables sont choisies durant la recherche en profondeur de l'espace de recherche est décidé par une *heuristique de choix de variables*, notée $H$. Autrement dit, à chaque nœud interne $\nu$ de l'arbre de recherche $\mathcal{T}$, la procédure MAC sélectionne une nouvelle variable $x$ en utilisant $H$, et affecte à $x$ une valeur $v$ en accord avec l'heuristique de choix de valeurs, qui suit habituellement l'ordre lexicographique de $\texttt{dom}(x)$. Choisir la bonne heuristique de choix de variables $H$ pour un réseau de contraintes donné est une question clé. Dans la section suivante, nous présentons l'état de l'art des heuristiques de choix de variables qui ont été adoptées par les solveurs de contraintes

modernes.

**Résolution de contraintes** Les algorithmes de recherche avec retours-arrières liés à des heuristiques de choix de variables déterministes exhibent un comportement à longue traine (*heavy-tailed behavior*) autant sur les instances aléatoires que les instances CSP du monde réel [6]. Ce problème peut être pallié par l'usage d'une politique naviguant efficacement entre les différentes heuristiques de choix de variables durant la recherche et l'usage d'une stratégie de *redémarrage*. Le redémarrage de la recherche se fait de façon itérative depuis la racine de l'arbre et associe à chaque *run* une heuristique de choix de variables. Conceptuellement, une stratégie de redémarrage est une application $\texttt{restart} : \mathbb{N} \to \mathbb{N}$, où $\texttt{restart}(t)$ est le nombre maximal d'« étapes » qui peuvent être réalisées par l'algorithme de recherche avec retours-arrières pendant un *run* $t$.

Un solveur de contraintes, équipé de la procédure MAC et d'une stratégie de redémarrage `restart`, construit une séquence d'arbres binaires de recherche $\langle \mathcal{T}^{(1)}, \mathcal{T}^{(2)}, \ldots \rangle$, où $\mathcal{T}^{(t)}$ est l'arbre de recherche exploré par MAC au *run* $t$. Lorsque le solveur redémarre depuis le début, celui-ci est capable de mémoriser les informations importantes de la séquence $\langle \mathcal{T}^{(1)}, \mathcal{T}^{(2)}, \ldots, \mathcal{T}^{(t-1)} \rangle$. Par exemple, le solveur peut maintenir en cache les *nogoods* qui apparaissent sur la branche courante de l'arbre construit pendant le *run* courant.

Le *cutoff*, qui correspond au nombre d'étapes autorisés, peut être défini par le nombre de nœuds, le nombre de mauvaises décisions [3], le nombre de secondes, ou toutes autre mesures adéquates. Pour une stratégie de redémarrage au *cutoff* fixe, le nombre $T$ d'essais est fixé à l'avance, et $\texttt{restart}(t)$ est une constante pour chaque *run* $t$, exception faite pour le $T^{\text{ème}}$ *run* qui autorise un nombre illimité d'étapes (dans le but de maintenir un algorithme complet). Cette stratégie est reconnue pour son efficacité en pratique [7], toutefois une bonne valeur pour le *cutoff* doit être trouvée à travers différents essais et erreurs. Alternativement, pour une stratégie de redémarrage dont le *cutoff* est dynamique, le nombre $T$ de *runs* est inconnu, mais `restart` croît géométriquement, ce qui garantit que l'espace entier des solutions partielles est exploré après $O(n)$ *runs*. Une stratégie de redémarrage communément utilisée ces dernières années est basée sur la suite de Luby [9]. Dans cette stratégie, la longueur du *run* $t$ correspond à $u \times l_t$ où $u$ est une constante servant d'unité pour le *run* et

$$l_t = \begin{cases} 2^{k-1}, & \text{if } t = 2^k - 1 \\ l_{t-2^{k-1}+1}, & \text{if } 2^{k-1} \leq t < 2^k - 1. \end{cases}$$

## 4 Heuristiques de choix de variables

Nous proposons dans cette section une vue d'ensemble des heuristiques de choix de variables. L'heuristique de choix de variables `dom` [8], qui sélectionne les variables en séquence selon l'ordre croissant de la taille de leur domaine, a longtemps été considérée comme la plus robuste des heuristiques. Cependant, il y a une quinzaine d'années, les heuristiques modernes *adaptatives* ont été introduites : elles prennent en compte l'information collectée à travers l'espace de recherche déjà explorée. Les deux premières heuristiques adaptatives génériques proposées ont été `impact` [13] et `wdeg` [4]. La première s'appuie sur une mesure de l'impact sur l'espace de recherche des différentes affectations, et la seconde associe un compteur à chaque contrainte (et indirectement, à chaque variable) indiquant combien de fois chaque contrainte a mené à un *domain wipeout*. Les heuristiques basées sur le comptage [12] et sur l'activité [10] sont aussi deux techniques adaptatives apparues plus récemment.

Dans ce papier, nous portons principalement notre attention aux trois heuristiques adaptatives et génériques `wdeg`, `impact` et `activity`. Nous considérons aussi d'autres heuristiques de référence, les classiques `dom`, `cos` (conflict-ordering search [5]) qui réordonne progressivement les variables. Il est à noter que pour `ddeg` et `wdeg`, nous choisissons les variantes les plus efficaces `dom/ddeg` et `dom/wdeg` comme décrit ci-après :

— `dom/ddeg` sélectionne en priorité la variable avec le plus petit rapport « taille du domaine courant sur degré dynamique » [15]. Le degré dynamique d'une variable $x$ est le nombre de contraintes impliquant $x$ et au moins une autre variable non affectée.

— `dom/wdeg` sélectionne en priorité la variable avec le plus petit rapport « taille du domaine courant sur degré pondéré » [4]. Un poids est calculé pour chaque variable $x$ comme étant la somme des poids de toutes les contraintes impliquant $x$ et au moins une autre variable non affectée. Chaque contrainte $c$ possède un poids, initialisé à 1, qui est incrémenté à chaque *domain wipeout* apparaissant lorsque $c$ est sollicité dans le processus de propagation de contraintes.

— `impact`, ou IBS (*Impact-Based Search*) sélectionne en priorité la variable avec le plus fort impact. L'impact d'une variable $x$ correspond à une mesure de l'importance qu'a $x$ dans la réduction de l'espace de recherche [13]. La taille $\texttt{size}(P)$ de l'espace de recherche de $P$ est le produit de la taille de tous les domaines courants :

$$\texttt{size}(P) = \prod_{x \in \texttt{vars}(P)} |\texttt{dom}(x)|$$

L'impact $I$ de l'affectation d'une variable $x = a$ sur $P$ est calculé comme suit :

$$I(x = a) = 1 - \frac{\texttt{size}(P')}{\texttt{size}(P)}$$

où $P' = AC(P|_{x=a})$ donne le réseau de contraintes après affectation de $x$ à $a$ et l'établissement de l'arc-cohérence. Il est à noter que si $P'$ mène à un échec, alors $I(x = a) = 1$. Cette heuristique peut être aussi utilisée comme choix de valeur.

— `activity`, ou ABS (*Activity-Based Search*) sélectionne en priorité la variable avec la plus forte activité. L'activité d'une variable $x$ est mesurée par le nombre de fois où le domaine de $x$ est réduit durant la recherche [10]. Cette heuristique est basée sur le rôle clé de la propagation en programmation par contraintes et s'appuie sur un facteur de vieillissement pour affaiblir progressivement les précédentes statistiques. Les activités sont initialisées par une distribution aléatoire de probabilités sur l'espace de recherche.
Plus formellement, l'activité $A(x)$ d'une variable $x$ est mise à jour à chaque nœud de l'arbre de recherche peu importe le résultat (succès ou échec) suivant ces deux règles :

i. $A^{P'}(x) = A^P(x) * \gamma$, où $0 \leq \gamma \leq 1$, $|\texttt{dom}^{P'}(x)| > 1$ et $\texttt{dom}^{P'}(x) = \texttt{dom}^P_x$

ii. $A^{P'}(x) = A^P(x) + 1$ si $\texttt{dom}^{P'}(x) \subset \texttt{dom}^P(x)$

— `cos` (*Conflict Ordering Search* [5]). L'idée est d'associer un *timestamp* à l'ensemble des variables participant aux derniers conflits. Quand le solveur doit sélectionner une nouvelle variable non fixée, celle possédant le plus récent *timestamp* est choisie. Au commencement de la recherche, quand aucune variable n'a de *timestamp*, l'heuristique de choix de variables du solveur est utilisée pour guider la recherche. `cos` n'oublie pas les variables conflictuelles et réordonne progressivement ses variables pour donner une priorité à celles apparaissant dans des conflits plus récents. Les résultats expérimentaux ont montré l'efficacité de cette approche, en particulier pour certains problèmes structurés connus comme les problèmes d'ordonnancement.

Lors d'une égalité, qui correspond au cas de deux variables (au moins) jugées équivalentes par l'heuristique, on peut utiliser un second critère (par exemple, le degré dynamique de chaque variable). Cependant, pour les heuristiques adaptatives, il est habituel d'utiliser `lexico` donnant la priorité à la première variable ayant le meilleur score.

# 5 Les modèles de bandits

Comme indiqué dans la Section 3, le processus de résolution de contraintes par l'application de la politique de redémarrage peut être vu comme une séquence $\langle 1, 2, \cdots, T\rangle$ de *runs*. Pour les politiques de redémarrage mentionnées précédemment, la séquence de *runs* est finie, mais l'horizon $T$ n'est pas nécessairement connu à l'avance. Durant chaque *run* $t$, le solveur appelle l'algorithme MAC pour construire un arbre de recherche $\mathcal{T}_t$, dont la taille est déterminée par le *cutoff* de la politique de redémarrage. Si le solveur n'a accès qu'à une heuristique de choix de variables, dit $H_1$, il va exécuter MAC avec $H_1$ après chaque redémarrage. Toutefois, si le solveur a accès à d'autres heuristiques de choix de variables, il doit faire face à un choix fondamental à chaque *run* $t$ : soit appeler MAC en exploitant l'heuristique $H_i$ maximisant l'efficacité du solveur au *run* $t$ ou appeler MAC en explorant une heuristique dont les résultats paraissent moins fructueux mais qui a tout de même ses chances de battre l'heuristique la plus efficace actuellement.

## 5.1 Généralité et notations

Par essence, la tâche d'exploration et d'exploitation d'un solveur de contraintes avec redémarrages est de décider, à chaque *run* $t$, quelle heuristique $H_i$ utiliser. En empruntant la terminologie de la théorie des jeux, une stratégie d'*exploration-exploitation* sélectionne à chaque *run* $t$ l'une des actions disponibles dans $A = \{H_1, ..., H_K\}$ où $K$ est le nombre d'actions disponibles. Le problème est d'équilibrer la part d'exploration par rapport à la part d'exploitation de la meilleure heuristique, afin d'estimer au mieux la probabilité qu'a chacune des heuristiques d'apporter le meilleur gain pour la recherche.

Cette succession de décisions peut donc être représentée par un problème de bandit multi-bras stochastique. Nous partons du principe que ce problème est stochastique et que donc, la distribution de probabilités de succès des différentes heuristiques est fixe. Le comportement d'un bandit stochastique peut être décrit par le cadre général suivant :

1. (choix) A chaque début de *run*, le bandit est appelé à sélectionner l'heuristique qui sera utilisée pour poursuivre la recherche.
2. (mise-à-jour) À chaque fin de *run*, le bras courant (représentant l'heuristique) est mis à jour avec la récompense qui lui est décernée.

Avant de présenter nos différents modèles de bandit, nous devons introduire ces notations :

— $\mu(a)$ est la moyenne (réelle) de la récompense $r(a)$ ;

— $\mu^*$ est la moyenne de la meilleure action (heuristique) ;
— $\Delta_a = \mu^* - \mu(a)$ est l'intervalle de sous-optimalité du choix $a$ ;
— $n_t(a)$ est le nombre de fois où le joueur a sélectionné $a$ durant les $t$ premières étapes (*runs*) ;
— $\widehat{\mu}_t$ est la moyenne empirique de $r(a)$ sur les $n_t(a)$ jeux.

Enfin, la performance d'un bandit sur la totalité $T$ de son exécution est défini comme la différence entre l'espérance de la récompense que l'on obtiendrait en utilisant $T$ fois le meilleur bras et l'espérance de la récompense après $T$ essais effectués par le bandit. Il s'agit du *pseudo-regret* :

$$\mathcal{R}_T = \max_{a \in A} \sum_{t=1}^{T} \mathbb{E}[r_t(a)] - \sum_{t=1}^{T} \mathbb{E}[r_t(a_t)]$$

Cette mesure indique la performance théorique qu'un bandit nous garantit en appliquant sa politique.

### 5.2 Fonction de récompense

Nous décrivons dans cette partie la fonction de récompense nous permettant de récompenser le plus fidèlement possible les différents bandits décrits ci-après. Notre fonction de récompense est définie par le nombre de solutions inférées fausses (`sif`) lors d'un *run*. Ce nombre de solutions est calculé lors de chaque *domain wipeout* par le produit de la taille des domaines de chaque variable non affectée du réseau de contraintes $P'$ :

$$\texttt{sif}(P') = \prod_{x \in \texttt{FutVars}(P')} |\texttt{dom}(x)|$$

où $\texttt{FutVars}(P')$ est l'ensemble des variables non affectées. Ainsi, nous récompensons le bandit par la somme de ces solutions inférées fausses normalisée par le nombre total de solutions du problème :

$$r_t(a) = \frac{\log(\sum_{\nu \in \texttt{dwo}(T_t)} \texttt{sif}(P_\nu))}{\log(\texttt{size}(P))} \quad (1)$$

Une mise à l'échelle logarithmique est nécessaire afin d'obtenir une récompense plus équilibrée entre 0 et 1. Plus le nombre de solutions inférées fausses est important, meilleure est la qualité de la récompense attribuée à l'heuristique de choix de variables.

### 5.3 Le modèle d'epsilon-greedy

Epsilon-greedy [16] (Algorithme 1) correspond à la stratégie la plus simple et naïve afin de venir à bout du problème des bandits stochastiques. Il suffit de préciser une valeur $\varepsilon \in [0, 1]$, permettant de savoir dans quelle proportion nous souhaitons explorer ou exploiter.

**Algorithme 1 :** epsilon-greedy

1 **pour chaque** $t = 1$ *à* $T$ **faire**
2 Choix du paramètre $\varepsilon_t \in [0, 1]$
3 Choix de $a_t \in \{H_1, ..., H_K\} =$
$\begin{cases} \text{aléatoirement,} & \text{avec la probabilité } \varepsilon_t \\ \text{maximisant } \widehat{\mu}_{t-1}(a_t), & \text{sinon} \end{cases}$

En admettant que $d < \min_{j \neq i^*} \Delta_j$ et qu'on initialise $\varepsilon_t = \frac{K}{(d^2 t)}$, epsilon-greedy accomplit un *pseudo-regret* en $O(\frac{K \ln(T)}{d^2})$. Une borne inférieure $d$ basée sur l'intervalle minimal doit être connue à l'avance.

En application à ce sujet, cela revient à dire qu'au bout de $T$ *runs* à choisir parmi $K$ heuristiques, epsilon-greedy garantit un pseudo-regret en $O(\frac{K \ln(T)}{d^2})$, et que donc, le regret cumulé est borné par cette précédente formule.

### 5.4 Le modèle d'ucb

Avec ucb (upper confidence bound) [1], il n'est plus nécessaire de préciser un $\varepsilon$ afin de faire varier l'exploration et l'exploitation. En effet, la politique appliquée (Algorithme 2) lui permet de gérer la part d'exploration et d'exploitation sans l'usage d'une fonction aléatoire.

**Algorithme 2 :** ucb

1 **pour chaque** $t = 1$ *à* $T$ **faire**
2 Choix de $a_t \in \{H_1, ..., H_K\}$ maximisant
$\widehat{\mu}_{t-1}(a_t) + \sqrt{\frac{8 \ln(t)}{n_{t-1}(a_t)}}$

$\widehat{\mu}_{t-1}(i)$ correspond à la partie exploitation de l'algorithme, tandis que $\sqrt{\frac{8 \ln(t)}{n_{t-1}(i)}}$ correspond à un poids supplémentaire donné au bras notamment quand celui-ci n'a pas été souvent sélectionné (partie exploration).

Ucb accomplit un *pseudo-regret* en $O(\sqrt{KT})$.

## 6 Évaluation expérimentale

Nous menons une évaluation sur la base d'une grande diversité de classes de problèmes (37) provenant de la compétition XCSP3 (`http://xcsp.org`), incluant : AllInterval, Bibd, Blackhole, CarSequencing, ColouredQueens, CostasArray, CoveringArray, Crossword, DiamondFree, Dubois, Fischer, frb, gp10, GracefulGraph, Haystacks, KnightTour, Langford, MagicHexagon, MagicSquare, MarketSplit, MultiKnapsack, NumberPartitioning, Ortholatin, Primes,

| | | vbs | epsilon | ucb | rand | dom/wdeg | impact | activity | dom/ddeg | cos | dom |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **All Interval** | **commun (32)** | 107 | 117 | 118 | 117 | 2977 | 108 | 108 | 173 | 110 | 117 |
| | **total (#res)** | **107 (32)** | 117 (32) | 118 (32) | 117 (32) | 2977 (32) | **108 (32)** | **108 (32)** | 173 (32) | 110 (32) | 117 (32) |
| **Bibd** | **commun (2)** | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| | **total (#res)** | **3737 (7)** | 1508 (6) | 1758 (6) | **3978 (7)** | 3506 (3) | 309 (4) | 2180 (5) | 6 (2) | 93 (3) | 1297 (5) |
| **Blackhole** | **commun (0)** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | **total (#res)** | **891 (13)** | 2130 (11) | 64 (10) | 4775 (11) | 1471 (2) | **892 (13)** | 68 (7) | 0 (0) | 11 (5) | 717 (5) |
| **Car Sequencing** | **commun (1)** | 2 | 3 | 3 | 3 | 3 | 2 | 2 | 3 | 2 | 3 |
| | **total (#res)** | **2592 (14)** | 5250 (10) | **9479 (14)** | 9508 (11) | 3124 (13) | 5348 (8) | 1971 (6) | 3 (1) | 11 (2) | 596 (2) |
| **Coloured Queens** | **commun (5)** | 10 | 14 | 15 | 14 | 15 | 10 | 11 | 15 | 11 | 15 |
| | **total (#res)** | 25 (6) | **299 (7)** | 42 (6) | **314 (7)** | 15 (5) | 10 (5) | 11 (5) | 15 (5) | 11 (5) | 29 (6) |
| **Costas Array** | **commun (4)** | 9 | 13 | 13 | 13 | 12 | 10 | 10 | 12 | 35 | 1197 |
| | **total (#res)** | **702 (9)** | 1767 (9) | 3662 (9) | 1356 (9) | 1793 (9) | 1331 (8) | 770 (8) | **719 (9)** | 2420 (7) | 1197 (4) |
| **Covering Array** | **commun (2)** | 4 | 5 | 6 | 6 | 6 | 4 | 4 | 6 | 4 | 6 |
| | **total (#res)** | 41 (6) | 216 (6) | 57 (6) | **31 (6)** | 3500 (6) | 41 (6) | 4 (2) | 6 (2) | 74 (3) | 1202 (3) |
| **Crossword** | **commun (6)** | 16 | 21 | 21 | 22 | 21 | 47 | 16 | 21 | 1365 | 536 |
| | **total (#res)** | **2382 (12)** | 378 (9) | 2880 (10) | 1150 (10) | **3921 (12)** | 3148 (10) | 1460 (9) | 2570 (11) | 1833 (7) | 536 (6) |
| **Diamond Free** | **commun (0)** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | **total (#res)** | **128 (6)** | 256 (6) | 847 (6) | 578 (6) | 2017 (6) | 2146 (6) | 339 (6) | 1372 (4) | 0 (0) | **128 (6)** |
| **Dubois** | **commun (2)** | 74 | 52 | 186 | 78 | 1286 | 207 | 74 | 430 | 88 | 89 |
| | **total (#res)** | 2028 (5) | **1708 (6)** | 3248 (5) | 689 (4) | 2353 (3) | 3968 (4) | 1097 (4) | 2677 (3) | 3769 (5) | 1624 (4) |
| **Fischer** | **commun (0)** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | **total (#res)** | 2251 (8) | **9192 (9)** | **9156 (9)** | **9681 (9)** | 8235 (7) | 2869 (8) | 136 (6) | 7967 (7) | 774 (1) | 0 (0) |
| **frb** | **commun (0)** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | **total (#res)** | **5096 (7)** | 2039 (4) | 19 (3) | 2255 (4) | 2097 (6) | 34 (3) | 17 (3) | **4280 (6)** | 236 (1) | 0 (0) |
| **gp10** | **commun (0)** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | **total (#res)** | **4269 (5)** | **7461 (4)** | 5242 (3) | 5755 (3) | 1724 (1) | 3714 (3) | 2685 (3) | 0 (0) | 0 (0) | 0 (0) |
| **Graceful Graph** | **commun (11)** | 24 | 33 | 33 | 33 | 33 | 24 | 50 | 33 | 132 | 33 |
| | **total (#res)** | **943 (20)** | 4192 (19) | 2086 (19) | 3107 (19) | 253 (18) | 3337 (20) | 50 (11) | 2576 (20) | 460 (13) | **971 (20)** |
| **Haystacks** | **commun (1)** | 2 | 2 | 3 | 3 | 3 | 2 | 2 | 3 | 2 | 3 |
| | **total (#res)** | **42 (4)** | 50 (4) | 11 (3) | 127 (4) | 8 (2) | **42 (4)** | 437 (4) | 473 (4) | 33 (3) | 3 (1) |
| **Knight Tour** | **commun (3)** | 16 | 26 | 97 | 25 | 23 | 16 | 20 | 777 | 153 | 446 |
| | **total (#res)** | **110 (5)** | 2317 (5) | 984 (5) | 1457 (5) | 1757 (4) | **110 (5)** | 20 (3) | 777 (3) | 153 (3) | 446 (3) |
| **Langford** | **commun (10)** | 152 | 409 | 486 | 365 | 215 | 519 | 482 | 159 | 3225 | 3845 |
| | **total (#res)** | **219 (13)** | 1285 (13) | 601 (13) | 1701 (13) | **283 (13)** | 748 (13) | 499 (12) | 179 (12) | 3225 (10) | 3845 (10) |
| **Magic Hexagon** | **commun (0)** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | **total (#res)** | **293 (5)** | **715 (5)** | 766 (5) | 810 (5) | 924 (5) | 1290 (4) | 1018 (4) | 1357 (5) | 2221 (1) | 0 (0) |
| **Magic Square** | **commun (0)** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | **total (#res)** | **2843 (24)** | 3362 (23) | 542 (21) | 1386 (22) | **2471 (23)** | 12502 (17) | 2173 (18) | 1325 (12) | 0 (0) | 0 (0) |
| **Market Split** | **commun (10)** | 109 | 929 | 386 | 1084 | 5374 | 913 | 221 | 948 | 621 | 953 |
| | **total (#res)** | **109 (10)** | 929 (10) | 386 (10) | 1084 (10) | 5374 (10) | 913 (10) | **221 (10)** | 948 (10) | 621 (10) | 953 (10) |
| **Multi Knapsack** | **commun (24)** | 63 | 83 | 81 | 84 | 1116 | 130 | 63 | 111 | 84 | 91 |
| | **total (#res)** | **93 (27)** | 170 (27) | **149 (27)** | 156 (27) | 1116 (24) | 1042 (25) | 157 (27) | 126 (26) | 3520 (27) | 3911 (26) |
| **Number Partitioning** | **commun (6)** | 20 | 21 | 22 | 22 | 20 | 20 | 41 | 22 | 92 | 880 |
| | **total (#res)** | **148 (19)** | 497 (19) | **224 (19)** | 722 (19) | 626 (19) | 589 (19) | 4681 (17) | 6031 (16) | 3101 (9) | 880 (6) |
| **Ortholatin** | **commun (1)** | 6 | 7 | 16 | 13 | 8 | 17 | 90 | 185 | 17 | 6 |
| | **total (#res)** | 3959 (4) | **152 (5)** | 199 (5) | 183 (5) | 830 (2) | 1744 (2) | 3194 (2) | 185 (1) | 17 (1) | 1318 (3) |
| **Primes** | **commun (15)** | 32 | 44 | 44 | 45 | 44 | 33 | 33 | 45 | 33 | 45 |
| | **total (#res)** | **55 (23)** | 74 (23) | 76 (23) | 76 (23) | **71 (23)** | 86 (23) | 99 (23) | 821 (23) | 33 (15) | 45 (15) |
| **Quasi Group** | **commun (3)** | 6 | 9 | 9 | 9 | 9 | 6 | 6 | 9 | 6 | 9 |
| | **total (#res)** | **213 (4)** | 384 (4) | 425 (4) | 769 (4) | 474 (4) | 6 (3) | **213 (4)** | 1153 (4) | 6 (3) | 9 (3) |
| **Queens Knights** | **commun (6)** | 19 | 22 | 21 | 21 | 21 | 19 | 19 | 4027 | 19 | 86 |
| | **total (#res)** | **1838 (18)** | 8174 (18) | 5619 (18) | 4685 (18) | 6618 (17) | **1840 (18)** | 681 (12) | 4027 (6) | 2102 (18) | 86 (6) |
| **qwh** | **commun (5)** | 12 | 17 | 16 | 16 | 15 | 22 | 12 | 38 | 79 | 2290 |
| | **total (#res)** | **14722 (53)** | 25982 (47) | 31605 (49) | 29480 (49) | **24042 (52)** | 14544 (20) | 21269 (48) | 18924 (37) | 96 (6) | 2302 (6) |
| **Radar Surveillance** | **commun (9)** | 28 | 29 | 29 | 29 | 28 | 29 | 29 | 29 | 29 | 29 |
| | **total (#res)** | 1662 (39) | **7528 (42)** | 10642 (41) | 11413 (41) | 1664 (39) | 54 (13) | 142 (14) | 417 (12) | 2759 (19) | 29 (9) |
| **Rlfap** | **commun (0)** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | **total (#res)** | **3524 (12)** | 4725 (11) | 3813 (11) | 1697 (10) | **3524 (12)** | 3353 (10) | 2451 (6) | 0 (0) | 2483 (3) | 0 (0) |
| **RoomMate** | **commun (9)** | 2586 | 2647 | 2705 | 2680 | 2774 | 2651 | 2637 | 3020 | 2602 | 2644 |
| | **total (#res)** | **2586 (9)** | **2647 (9)** | 2705 (9) | 2680 (9) | 2774 (9) | 2651 (9) | 2637 (9) | 3020 (9) | 2602 (9) | 2644 (9) |
| **Social Golfers** | **commun (22)** | 59 | 77 | 77 | 77 | 78 | 60 | 60 | 81 | 625 | 804 |
| | **total (#res)** | 575 (27) | 612 (27) | **453 (27)** | 467 (27) | 912 (27) | 2494 (26) | 851 (27) | 1242 (25) | 625 (22) | 809 (23) |
| **Sports Scheduling** | **commun (5)** | 13 | 29 | 19 | 51 | 37 | 26 | 13 | 28 | 781 | 593 |
| | **total (#res)** | **87 (6)** | 372 (6) | 155 (6) | 526 (6) | 350 (6) | 715 (6) | **87 (6)** | 2452 (6) | 781 (5) | 593 (5) |
| **Steiner3** | **commun (3)** | 6 | 9 | 9 | 9 | 9 | 7 | 6 | 9 | 6 | 9 |
| | **total (#res)** | **10 (4)** | 13 (4) | 13 (4) | 13 (4) | 13 (4) | 14 (4) | **10 (4)** | 13 (4) | 11 (4) | 9 (3) |
| **Strip Packing** | **commun (0)** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | **total (#res)** | 384 (4) | 1499 (5) | 2872 (5) | **1460 (5)** | 600 (4) | 225 (2) | 8 (2) | 275 (1) | 38 (1) | 0 (0) |
| **Subiso-morphism** | **commun (1)** | 4 | 11 | 5 | 7 | 5 | 141 | 4 | 4 | 2297 | 30 |
| | **total (#res)** | 1126 (8) | 2799 (8) | **5098 (9)** | 2902 (8) | 3736 (7) | 2649 (5) | 8 (2) | 1294 (8) | 2312 (2) | 30 (1) |
| **Travelling Salesman** | **commun (43)** | 218 | 632 | 686 | 557 | 298 | 1852 | 613 | 422 | 624 | 4079 |
| | **total (#res)** | **347 (45)** | 742 (45) | 910 (45) | 641 (45) | **442 (45)** | 3099 (45) | 819 (45) | 557 (45) | 1020 (45) | 4079 (43) |
| **Global** | **commun (241)** | 3616 | 5281 | 5123 | 5400 | 14447 | 6894 | 4643 | 10630 | 13061 | 18856 |
| | **total (#res)** | **60153 (513)** | **101562 (498)** | **106923 (497)** | **107748 (497)** | 95613 (474) | 77982 (413) | 52585 (406) | 67977 (371) | 37575 (300) | 30418 (275) |
| | | vbs | epsilon | ucb | rand | dom/wdeg | impact | activity | dom/ddeg | cos | dom |

TABLE 1 – Temps cpu (cumulé en secondes) et nombre d'instances résolues (#res) pour différentes stratégies de recherche.

QuasiGroup, QueensKnights, qwh, RadarSurveillance, RectPacking, Rlfap, RoomMate, SocialGolfers, SportsScheduling, Steiner3, StripPacking, Subisomorphism and TravellingSalesman, totalisant 746

instances. Celles-ci sont exécutées sur des nœuds possédant un processeur Intel Xeon 2,66 GHz et une mémoire de 32 GB. Nous utilisons le solveur AbsCon (`http://www.cril.fr/~lecoutre/#/softwares`) dans lequel nous avons intégré les différents bandits présentés précédemment. AbsCon est configuré pour utiliser AC comme niveau de cohérence, pour suivre le suite de Luby comme politique de redémarrage basée sur le nombre de nœuds visités et enfin, le temps limite d'exécution est fixé à une heure.

Pour mener à bien cette évaluation, nous testons l'ensemble des heuristiques de choix variables suivantes : `dom/wdeg`, `dom/ddeg`, `dom`, `activity`, `impact` et `cos`. Nous testons nos bandits `epsilon` et `ucb` dont les bras sont composés à partir des différentes heuristiques citées précédemment. La valeur $\varepsilon$ du bandit `epsilon` a été fixée à 0.9 par recherche linéaire. Les bandits sont récompensés par la fonction présentée dans la Section 5.2. De nombreuses fonctions de récompenses ont été testées, mais seule celle-ci a retenu notre attention. En plus de ces bandits, nous expérimentons une politique basée sur les bandits mais dont le choix des différents bras se fait de façon aléatoire, que nous appelons `rand`. Enfin, nous agrémentons ces tests de la génération d'un `vbs` (virtual best solver) : il se compose lui aussi des six précédentes heuristiques, mais ne garde que les meilleurs résultats de chacune d'entre elles. À travers ces candidats témoins, nous voyons la capacité qu'ont nos bandits à être sensiblement plus efficaces que la politique de `rand`, mais aussi à quel point ceux-ci se rapprochent de l'optimalité du `vbs`.

Il est à noter que, pour le cas d'`epsilon` et `rand`, les expérimentations sont menées trois fois avec trois graines différentes pour leur fonction de randomisation. Les expérimentations sont agrégées en prenant la médiane des temps pour chaque instance. Cette mesure nous semble la plus juste dans le sens où :

— quand une instance n'est résolue qu'une seule fois sur les trois, celle-ci est considérée comme irrésolue ;
— dans le cas où deux graines ont réussi à résoudre l'instance, celle possédant le temps le plus élevé est conservée ;
— enfin, dans le cas où pour chaque graines l'instance est résolue, le temps médian semble tout aussi appropriée.

Il est aussi important de préciser que, dans les stratégies hybrides, les heuristiques sont isolées les une des autres afin de ne pas se perturber entre elles. Il s'agit d'un point important car, dans de précédentes expérimentations, ce partage que nous pensions collaboratif s'est avéré être une perturbation négative et néfaste à côté des résultats que nous décrirons dans la suite de cette section.

Le Tableau 1 montre (de gauche à droite), le `vbs`, nos bandits (`epsilon` et `ucb`), la stratégie choisissant aléatoirement ses bras `rand` et enfin les six heuristiques simples. Les colonnes sont ordonnées de façon décroissante par rapport à leurs résultats globaux. Pour chaque variante, nous présentons le nombre d'instances qu'elle a résolues et le temps CPU requis pour les résoudre, mais aussi le temps CPU requis pour résoudre les instances résolues par toutes les méthodes. Nous observons de façon globale que, les stratégies hybrides (`epsilon`, `ucb` et `rand`) ont la plupart du temps résolu plus d'instances que les stratégies simples. Nous remarquons que les bandits ne dépassent que de peu la stratégie aléatoire. Ce phénomène s'explique par le fait que, contrairement à `rand` qui donne arbitrairement et équitablement sa chance à chaque bras, les bandits nécessitent un temps d'apprentissage afin d'équilibrer les parts d'exploration et d'exploitation. De plus, malgré l'étude de différentes fonctions de récompense et le choix de la plus prometteuse, il est compliqué d'en construire une qui puisse s'adapter à chacune des instances.

De façon plus détaillée, nous remarquons que pour certains problèmes le `vbs` est surclassé par les bandits : ColouredQueens, Dubois, Ortholatin, RadarSurveillance, StripPacking et Subisomorphism. Pour la plupart des autres cas, les bandits égalent ou surclassent la meilleure des heuristiques (`dom/wdeg`), excepté pour frb, Rlfap, qwh où ils perdent quelques instances. Les résultats globaux montrent que les bandits conservent la stabilité de `dom/wdeg` et dépassent même cette heuristique par la résolution de 24 instances supplémentaires et la réduction du temps commun de 67%.

Pour les instances "faciles" (c'est-à-dire résolues par toutes les méthodes), les stratégies hybrides s'en sortent avec un temps proche de la meilleure des heuristiques (`impact`).

La Figure 1 permet de visualiser les résultats du Tableau 1 à travers un *cactus plot* qui montre le nombre d'instances résolues par chaque stratégie par temps de résolution croissant. Sur ce genre de graphique, l'important est de projeter l'extrémité de la courbe représentant une stratégie de recherche sur l'axe des abscisses, renseignant ainsi le nombre d'instances résolues. La projection sur l'axe des ordonnées n'est que peu informative puisqu'elle nous informe du temps qu'a pris la plus longue instance de la stratégie correspondante.

Connu de la littérature, `dom/wdeg` est la plus efficace des heuristiques et cette particularité est mise en évidence à travers cette figure. En effet, nous voyons que `dom/wdeg` est loin devant ses cinq homologues. La courbe que nous observons la plus à droite correspond au témoin `vbs`, dominant l'ensemble des stratégies. Entre `dom/wdeg` et le `vbs` se situent les bandits et la

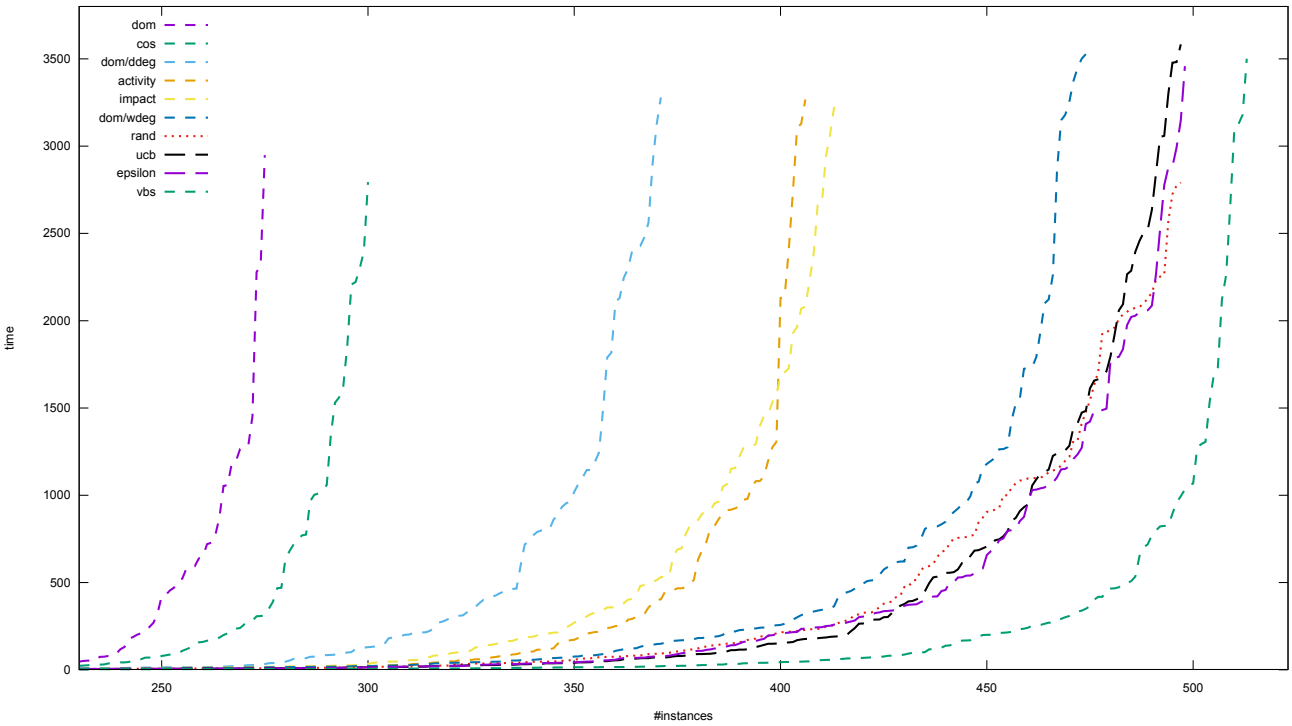



FIGURE 1 – Comparaison des heuristiques simples, de `rand`, des bandits et de `vbs`

stratégie aléatoire : tout comme dans le Tableau 1, nous constatons des résultats assez semblables entre ces stratégies hybrides. Il est tout de même rassurant de voir que la composition des heuristiques permette des résultats notables, visibles à la fois sur la Figure 1 et dans le Tableau 1.

## 7 Conclusion

Dans notre approche, nous tirons profit du mécanisme de redémarrage afin d'y placer une stratégie issue de l'apprentissage par renforcement, capable de sélectionner l'heuristique de choix de variables la plus adaptée à chaque *run*. Ainsi, par un jeu d'exploration et d'exploitation dont la politique des bandits multi-bras sait trouver un équilibre, nous observons une amélioration de l'efficacité du solveur. Il se montre ainsi plus efficace, dans le sens où il est capable de résoudre une plus grande diversité d'instances. Bien que la différence entre la stratégie basée sur la randomisation du choix des heuristiques et celle des bandits donnent des résultats très proches, le déterminisme de la politique d'`ucb` nous permet d'assurer ces résultats, contrairement à la variance des résultats pour les stratégies basées sur une fonction aléatoire.

Cette faible différence s'explique par le fait qu'il est compliqué de trouver une fonction de récompense suffisamment générique pour un bandit dans le cas d'un solveur de contraintes. En effet, au même titre qu'une heuristique ne peut pas être systématiquement la meilleure sur l'ensemble des instances, il s'agit ici de la même problématique pour les fonctions de récompense. Il est difficile pour une telle fonction d'être capable de discriminer de façon certaine les résultats préliminaires d'une heuristique, sans en avoir vu l'exécution complète. Celle-ci peut très bien être inefficace en début de recherche, puis trouver une certaine stabilité au bout d'un certain nombre de redémarrages, là où le bandit aurait écarté la piste de cette heuristique.

Peut-être serait-il plus judicieux de réduire cet ensemble d'heuristiques, par exemple à un couple d'heuristiques complémentaires, afin de récompenser plus finement le bandit ?

**Remerciements.** Les auteurs remercient les relecteurs anonymes pour leurs remarques constructives ayant permis d'améliorer la qualité de ce papier.

## Références

[1] Peter AUER, Nicolò CESA-BIANCHI et Paul FISCHER : Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256, May 2002.

[2] A. BALAFREJ, C. BESSIERE et A. PAPARRIZOU : Multi-armed bandits for adaptive constraint propa-

gation. *In Proceedings of IJCAI'15*, pages 290–296, 2015.

[3] C. Bessiere, B. Zanuttini et C. Fernandez : Measuring search trees. *In Proceedings of ECAI'04 workshop on Modelling and Solving Problems with Constraints*, pages 31–40, 2004.

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

[5] S. Gay, R. Hartert, C. Lecoutre et P. Schauss : Conflict ordering search for scheduling problems. *In Proceedings of CP'15*, pages 140–148, 2015.

[6] C. Gomes, B. Selman, N. Crato et H. Kautz : Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24:67–100, 2000.

[7] C. Gomes, B. Selman et H.A. Kautz : Boosting combinatorial search through randomization. *In Proceedings of AAAI'98*, pages 431–437, 1998.

[8] R. Haralick et G. Elliott : Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.

[9] Michael Luby, Alistair Sinclair et David Zuckerman : Optimal speedup of las vegas algorithms. *Inf. Process. Lett.*, 47(4):173–180, septembre 1993.

[10] L. Michel et P. Van Hentenryck : Activity-based search for black-box constraint programming solvers. *In Proceedings of CPAIOR'12*, pages 228–243, 2012.

[11] Ugo Montanari : Network of constraints : Fundamental properties and applications to picture processing. *Information Science*, 7:95–132, 1974.

[12] G. Pesant, C.-G. Quimper et A. Zanarini : Counting-based search : Branching heuristics for constraint satisfaction problems. *Journal of Artificial Intelligence Research*, 43:173–210, 2012.

[13] P. Refalo : Impact-based search strategies for constraint programming. *In Proceedings of CP'04*, pages 557–571, 2004.

[14] D. Sabin et E.C. Freuder : Contradicting conventional wisdom in constraint satisfaction. *In Proceedings of CP'94*, pages 10–20, 1994.

[15] B. Smith et S. Grant : Trying harder to fail first. *In Proceedings of ECAI'98*, pages 249–253, Brighton, UK, 1998.

[16] Richard Sutton et Andrew G. Barto : Reinforcement learning : An introduction. 9:1054, 02 1998.

[17] W. Xia et R. H. C. Yap : Learning robust search strategies using a bandit-based approach. pages 6657–6665, 2018.

# Sur l'UP-résilience des k-UCSs binaires*

**Mohamed Sami Cherif**† **Djamal Habet**
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
{mohamed-sami.cherif,djamal.habet}@univ-amu.fr

**Résumé**

Les solveurs basés sur la méthode séparation et évaluation (BnB) pour Max-SAT exploitent la max-résolution, la règle d'inférence pour Max-SAT, pour s'assurer que chaque sous-ensemble inconsistant (IS) détecté n'est compté qu'une seule fois. Cependant, les transformations par max-résolution peuvent affecter négativement leurs performances. Elles sont alors généralement apprises de manière sélective : quand elles respectent certains motifs. Dans ce papier, on s'intéresse à des motifs particuliers appelés $k$-UCSs binaires. On prouve que ces motifs vérifient une caractérisation récente des transformations par max-résolution appelée UP-résilience et on décrit comment ce résultat peut aider à étendre les motifs actuels. Enfin, ce travail s'inscrit dans le cadre d'une démarche globale de caractérisation de la pertinence des transformations par max-résolution.

**Abstract**

Branch and Bound (BnB) solvers for Max-SAT exploit max-resolution, the inference rule for Max-SAT, to ensure that every computed Inconsistent Subset (IS) is counted only once. However, learning max-resolution transformations can be detrimental to their performance so they are usually selectively learned if they respect certain patterns. In this paper, we focus on particular patterns called binary $k$-UCSs. We prove that these patterns verify a recent characterization of max-resolution transformations called UP-resilience and we show how this result can help extend the current patterns. Finally, this work is part of a global approach to characterize the relevance of transformations by max-resolution.

## 1 Introduction

Pour une formule propositionnelle sous Forme Normale Conjonctive (FNC), le problème Max-SAT consiste à trouver une affectation des variables qui maximise le nombre de clauses satisfaites. Les méthodes complètes pour résoudre Max-SAT incluent des algorithmes de type séparation et évaluation (BnB) (e.g. AHMAXSAT [2], MAXSATZ [7, 5]) qui calculent, à chaque nœud de l'arbre de recherche, la borne inférieure en comptant le nombre de sous-ensembles inconsistants (IS) disjoints de la formule avec la propagation unitaire simulée (SUP) [6]. Lorsqu'un IS est trouvé, il est soit supprimé temporairement dans le nœud courant soit transformé par max-résolution, la règle d'inférence pour Max-SAT [4], pour garantir qu'il ne sera compté qu'une seule fois. Cependant, l'apprentissage de ces transformations peut affecter négativement la qualité de l'estimation de la borne inférieure. Les solveurs de l'état de l'art apprennent alors les transformations de manière sélective sous la forme de motifs [7].

L'impact des transformations par max-résolution sur le mécanisme de la propagation unitaire a été étudié par la propriété de l'UP-résilience [3]. Elle met en relation l'influence de la max-résolution sur le mécanisme de la propagation unitaire, indispensable au calcul de la borne inférieure. De plus, De nouveaux motifs appelés $k$-UCSs ($k$-Unit Clause Subsets) ont été introduits et étudiés de manière empirique dans [1]. L'intérêt majeur de ces motifs est la production de clauses unitaires après la transformation par max-résolution. Dans ce travail, nous apportons une étude théorique de certains $k$-UCS et, plus précisément, de leur relation avec l'UP-résilence.

Ce papier est organisé comme suit. Dans la section 2, nous donnons quelques notations et définitions. Dans la section 3, nous prouvons notre conjecture sur l'UP-résilience des $k$-UCSs binaires. Enfin, nous discutons les implications de ce résultat et évoquons des perspectives dans la section 4.

*Ce travail est soutenu par l'Agence Nationale de la Recherche dans le cadre du projet DEMOGRAPH (ANR-16-C40-0028)

†Papier étudiant en master de recherche : Mohamed Sami Cherif est auteur principal.

## 2 Définitions & Notations

Dans cette section, nous rappelons quelques définitions et notions indispensables pour la suite.

**Définition 1 (max-résolution).** *La règle d'inférence pour Max-SAT, appelée max-resolution, est définie comme suit :*

$$\frac{c = \{x, y_1, ..., y_s\}, c' = \{\overline{x}, z_1, ..., z_t\}}{cr = \{y_1, ..., y_s, z_1, ..., z_t\}, cc_1, ..., cc_t, cc_{t+1}, ..., cc_{t+s}}$$

*avec les clauses de compensation :*

$$\begin{array}{c} cc_1 = \{y_1, ..., y_s, \overline{z_1}, ..., z_t\} \\ cc_2 = \{y_1, ..., y_s, \overline{z_2}, ..., z_t\} \\ ... \\ cc_t = \{y_1, ..., y_s, \overline{z_t}\} \\ cc_{t+1} = \{\overline{x}, z_1, ..., z_t, \overline{y_1}, y_2, ..., y_s\} \\ cc_{t+2} = \{\overline{x}, z_1, ..., z_t, \overline{y_2}, ..., y_s\} \\ ... \\ cc_{t+s} = \{\overline{x}, z_1, ..., z_t, \overline{y_s}\} \end{array}$$

La transformation produit une formule équivalente dans le sens où elle préserve le nombre des clauses insatisfaites pour toute interprétation [4]. Contrairement à la règle d'inférence pour SAT, la max-résolution remplace les prémisses de la règle par ses conclusions.

**Notation.** *Si $\psi$ est un IS d'une formule $\Phi$ et $S = \langle x_1, ..., x_k \rangle$ une séquence de variables dans $\psi$, on note $\Theta(\psi, S)$ l'ensemble des clauses obtenues après l'application des étapes de max-résolution relativement à la séquence $S$. Plus formellement, $\Theta(\psi, S) = \theta(\theta...(\theta(\psi, x_1), x_2)..., x_k)$ où $\theta(\psi, x)$ dénote l'application de l'étape de max-résolution définie ci-dessus sur la variable $x$.*

Un IS peut être détecté par différentes séquences de propagations unitaires correspondant chacune à un graphe d'implications [8]. Donc, avant de rappeler la notion d'UP-résilience, on donne la définition des voisinages possibles d'un littéral figurant dans un IS.

**Définition 2 (Voisinages possibles).** *Soit $\phi$ une formule et $\psi$ un IS. Pour un littéral $l$ dans $\psi$, on définit l'ensemble de ses voisinages possibles par $pneigh_\psi(l) = \{neigh_G(l) | G = (V, A)$ graphe d'implications de $\psi$ tel que $l \in V\}$ où $neigh_G(l)$ dénote les voisins de $l$ dans $G$.*

**Définition 3 (UP-résilience).** *Soit $\phi$ une formule, $\psi$ un IS et $S$ une séquence de variables dans $\psi$. On dit que la transformation $\Theta(\psi, S)$ est UP-résiliente pour un littéral $l$ dans $\psi$ ssi $\forall N \in pneigh_\psi(l)$ : $\square \in N$ ou $l$ peut être propagé dans $\Theta(\psi, S)|_N$ avec $\Theta(\psi, S)|_N$ l'ensemble des clauses de $\Theta(\psi, S)$ où les littéraux dans $N$ sont affectés à vrai. On dit que $\Theta(\psi, S)$ est UP-résiliente si elle est UP-résiliente pour tous les littéraux apparaissant dans $\psi$.*

**Définition 4 (k-UCS).** *Soit $\phi$ une formule et $k \geq 2$. On note k-UCS (k-Unit Clause Subset) tout ensemble de clauses $\psi \subseteq \phi$ tel qu'il existe un séquence d'étapes de max-résolution sur $\psi$ qui produit une résolvant intermédiaire unitaire. En particulier, si $\forall c \in \psi$ on a $|c| = 2$ alors $\psi$ est un k-UCS binaire, noté $k^b$-UCS.*

**Exemple 1.** *les motifs suivants qui sont appris dans les solveurs BnB de l'état de l'art, correspondent respectivement à un $2^b$-UCS et un $3^b$-UCS.*

$$\frac{\{l_1, l_2\}, \{l_1, \overline{l_2}\}}{\{l_1\}} (P_1) \qquad \frac{\{l_1, l_2\}, \{l_1, l_3\}, \{\overline{l_2}, \overline{l_3}\}}{\{l_1\}, \{l_1, l_2, l_3\}, \{\overline{l_1}, \overline{l_2}, \overline{l_3}\}} (P_2)$$

Certains $k$-UCSs sont facilement détectables en analysant le graphe d'implications de l'IS détecté [1]. En effet, les clauses qui sont entre le conflit et le premier point d'implication unique (FUIP) [8] produisent une clause résolvante unitaire si elles sont transformées par max-résolution. Dans la suite, on traite ce type de $k^b$-UCS.

## 3 UP-résilience des k-UCSs binaires

Dans cette section, on commence par prouver que les 2-UCSs et 3-UCSs sont UP-résilients pour tout ordre d'application de la max-résolution. Ensuite, on généralise ce résultat sur les $k$-UCSs binaires en introduisant un ordre qui assure leur UP-résilience.

**Proposition 1.** *Les motifs $2^b$-UCSs sont UP-résilients.*

**Preuve.** *Les $2^b$-UCSs sont tous de la forme $\psi = \{\{l_1, l_2\}, \{l_1, \overline{l_2}\}\}$. Ainsi, il existe deux séquences de propagations possibles pour les $2^b$-UCSs dont les graphes d'implications sont représentés dans Fig.1. Puisque tous les voisinages possibles de chaque littéral contiennent la clause vide, la transformation de $\psi$ est UP-résiliente.*

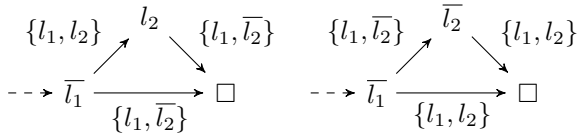



FIGURE 1 – Graphes d'implications représentant séquences de propagations possibles pour les $2^b$-UCSs

**Proposition 2.** *Les $3^b$-UCSs sont UP-résilients.*

**Preuve.** *Les $3^b$-UCSs sont tous de la forme $\psi = \{\{l_1, l_2\}, \{l_1, l_3\}, \{\overline{l_2}, \overline{l_3}\}\}$. Ainsi, il existe deux séquences de propagations possibles pour les $3^b$-UCSs dont les graphes d'implications sont représentés dans Fig.2. Puisque tous les voisinages possibles de $l_2$ et*

*$\overline{l_2}$ contiennent la clause vide, la transformation de $\psi$ par max-résolution est UP-résiliente pour $l_2$ et $\overline{l_2}$. De plus, la transformation de $\psi$ a la forme suivante suivante* $\dfrac{\{l_1, l_2\}, \{l_1, l_3\}, \{\overline{l_2}, \overline{l_3}\}}{\{l_1\}, \{l_1, l_2, l_3\}, \{\overline{l_1}, \overline{l_2}, \overline{l_3}\}}$ *Les littéraux $\overline{l_1}$ et $l_3$ admettent chacun un seul voisinage qui ne contient pas la clause vide, respectivement $\{l_2, l_3\}$ et $\{\overline{l_1}, \overline{l_2}\}$. Donc, les clauses $c = \{\overline{l_1}, \overline{l_2}, \overline{l_3}\}$ et $c' = \{l_1, l_2, l_3\}$ assurent la propagation des littéraux $\overline{l_1}$ et $l_3$ après la transformation par rapport à leurs voisinages respectifs.*

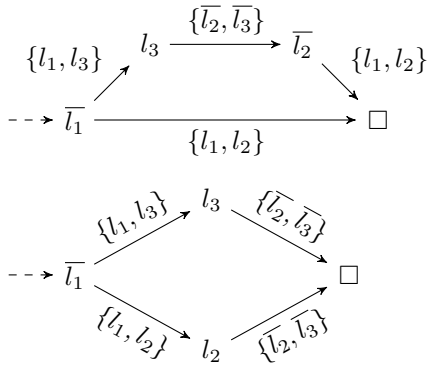



FIGURE 2 – Graphes d'implications représentant séquences de propagations possibles pour les $3^b$-UCSs

**Conjecture.** *$\forall k \geq 2$, les $k^b$-UCSs sont UP-résilients.*

Afin de prouver cette conjecture, on va commencer par caractériser les graphes d'implications représentant les séquences de propagations possibles des $k^b$-UCSs. Ensuite, en utilisant cette caractérisation, on définit un ordre d'application de la max-résolution qui assure l' UP-résilience après la transformation.

**Lemme 1.** *Soit $k \geq 2$ et $\psi$ un $k^b$-UCS reconnu par le FUIP $l$ dans un graphe d'implications $G$. Il existe exactement deux chemins disjoints de $l$ à $\square$ dans $G$.*

**Preuve.** *$l$ est un point d'implication unique (UIP) donc, tous les chemins des sommets représentant les clauses unitaires à $\square$ passent par $l$. La clause du conflit est binaire, donc $|pred(\square)| = 2$. Ainsi, il existe au moins deux chemins distincts de $l$ à $\square$ dans $G$. Soit $p_1$ and $p_2$ ces deux chemins. Supposons qu'il existe un chemin différent $p_3$ de $l$ à $\square$. On a deux cas possibles :*

- *$p_3$ est disjoint de $p1$ et $p_2$. Cela implique que $|pred(\square)| > 2$. C'est absurde car $|pred(\square)| = 2$.*
- *Sinon, puisque $|pred(\square)| = 2$, il existe $l' \neq l \in p_3$ tel que $l' \in p_1$ ou $l' \in p_2$ et $|pred(l')| > 1$. C'est aussi absurde car toutes les clauses de $k^b$-UCS sont binaires.*

*On déduit qu'il existe exactement deux chemins distincts de $l$ à $\square$ dans $G$. Le fait que $l$ est le FUIP et le même argument du second cas assurent que ces chemins sont disjoints.*

**Définition 5** (Ordre Circulaire)**.** *Soit $p_1 = \langle l, l_1^{p_1}, ..., l_m^{p_1}, \square\rangle (m \geq 0)$ et $p_2 = \langle l, l_1^{p_2}, ..., l_n^{p_2}, \square\rangle (n \geq 0)$ deux chemins disjoint de $l$ à $\square$ où $m + n > 0$. L'Ordre Circulaire (OC) des chemins $p_1$ et $p_2$ est défini par $OC(p_1, p_2) = \langle var(l_1^{p_1}), ..., var(l_m^{p_1}), var(l_n^{p_2}), ..., var(l_1^{p_2})\rangle$.*

**Théorème 1.** *$\forall k \geq 2$, les $k^b$-UCSs sont UP-resilients.*

**Preuve.** *Soit $k \geq 2$ et $\psi$ un $k^b$-UCS reconnu par le FUIP $l$ dans un graphe d'implications $G$. Par le lemme 1, soit $p_1 = \langle l, l_1^{p_1}, ..., l_m^{p_1}, \square\rangle (m \geq 0)$ et $p_2 = \langle l, l_1^{p_2}, ..., l_n^{p_2}, \square\rangle (n \geq 0)$ les deux chemins disjoints de $l$ à $\square$ où $m+n = k-1$ dans $G$. On suppose sans perte de généralité que $l_m^{p_1} = l'$ est le dernier littéral propagé. Il existe deux séquences de propagations possibles dont les graphes d'implications notés $G$ et $G'$ sont représentés dans Fig.3. On veut prouver que la transformation par max-résolution relativement à l'ordre $O = OC(p_1, p_2)$ est UP-résiliente :*

- *Les clauses propageant $l$ ne sont pas supprimées après la transformation par max-résolution relativement à l'ordre $O$. Donc $l$ peut être propagé si ses prédécesseurs dans $G$ sont affectés à vrai et la transformation est UP-résiliente pour $l$. Cet argument s'applique aussi à tous les littéraux impliqués dans la propagation de $l$.*
- *Tous les voisinages possibles des littéraux $l'$ et $\overline{l'}$ contiennent la clause vide. Ainsi, la transformation par max-résolution relativement à l'ordre $O$ est UP-résiliente pour $l'$ et $\overline{l'}$.*
- *Chaque littéral $l_j^{p_1}$ tel que $1 \leq j < m$ admet exactement un voisinage $neigh(l_j^{p_1}) = \{l_{j-1}^{p_1}, l_{j+1}^{p_1}\}$ qui ne contient pas la clause vide (on fixe $l_0^{p_1} = l$). L'étape de max-résolution sur $var(l_j^{p_1})(j \neq m-1)$ est de la forme :*

$$\frac{\{\bar{l}, l_j^{p_1}\}, \{\overline{l_j^{p_1}}, l_{j+1}^{p_1}\}}{\{\bar{l}, l_{j+1}^{p_1}\}, \{\bar{l}, l_j^{p_1}, \overline{l_{j+1}^{p_1}}\}, \{l, \overline{l_j^{p_1}}, l_{j+1}^{p_1}\}}$$

  *La clause $c = \{l, \overline{l_j^{p_1}}, l_{j+1}^{p_1}\}$ assure la propagation de $l_{j+1}^{p_1}$ si $l_j^{p_1} \in neigh(l_{j+1}^{p_1})$ est affecté à vrai puisque $\bar{l}$ est propagé par le dernier résolvant unitaire $\{\bar{l}\}$. De plus, pour $j = 1$, la clause $c' = \{\bar{l}, l_j^{p_1}, \overline{l_{j+1}^{p_1}}\}$ assure la propagation de $l_1^{p_1}$ si $l, l_2^{p_1} \in neigh(l_1^{p_1})$ sont affecté à vrai. On déduit que la transformation est UP-résiliente pour $l_j^{p_1}$ où $1 \leq j < m$.*
- *Chaque littéral $l_j^{p_2}$ tel que $1 \leq j \leq n$ admet exactement un voisinage $neigh(l_j^{p_2}) = \{l_{j-1}^{p_2}, l_{j+1}^{p_2}\}$ qui ne contient pas la clause vide (on fixe $l_0^{p_2} = l$ et*

*$l^{p_2}_{n+1} = \overline{l'}$). L'étape de max-résolution sur $var(l^{p_2}_j)$ ($j \neq 1$) est de la forme :*

$$\frac{\{\bar{l}, \overline{l^{p_2}_j}\}, \{l^{p_2}_j, \overline{l^{p_2}_{j-1}}\}}{\{\bar{l}, \overline{l^{p_2}_{j-1}}\}, \{\bar{l}, \overline{l^{p_2}_j}, l^{p_2}_{j-1}\}, \{l, l^{p_2}_j, \overline{l^{p_2}_{j-1}}\}}$$

*La clause $c = \{l, l^{p_2}_j, \overline{l^{p_2}_{j-1}}\}$ assure la propagation de $l^{p_2}_j$ si $l^{p_2}_{j-1} \in neigh(l^{p_2}_j)$ est affecté à vrai puisque $\bar{l}$ est propagé par le dernier résolvant unitaire $\{\bar{l}\}$. On déduit que la transformation est UP-résiliente pour $l^{p_2}_j$ où $1 < j \leq n$. Le seul littéral qui reste à vérifier est $l^{p_2}_1$ avec le voisinage $neigh(l^{p_2}_1) = \{l, l^{p_2}_2\}$. Pour cela, il suffit de regarder l'étape de max-résolution sur $var(l^{p_2}_2)$. En effet, cette étape génère la clause $c' = \{\bar{l}, \overline{l^{p_2}_2}, l^{p_2}_1\}$ qui assure la propagation de $l^{p_2}_1$ si $l, l^{p_2}_2 \in neigh(l^{p_2}_1)$ sont affectés à vrai.*

*On conclut que la transformation par max-résolution relativement à l'ordre O est UP-résiliente et, donc, que les $k^b$-UCSs sont UP-résilients.*

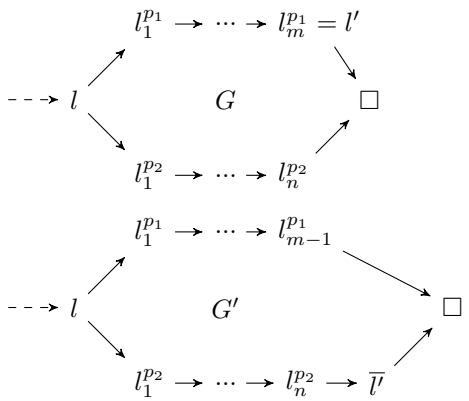



FIGURE 3 – Graphes d'implications représentant séquences de propagations possibles pour les $k^b$-UCSs

## 4 Discussion et Perspectives

Notre résultat établit l'UP-résilience des $k^b$-UCSs. Les résultats empiriques montrent que les $2^b$-UCSs et $3^b$-UCSs, correspondant respectivement aux motiifs ($P1$) et ($P2$) de l'exemple 1, appris dans les solveurs BnB de l'état l'art, ont un impact positif sur leurs performances [5, 1]. Plus précisemment, les propriétés 1 et 2 prouvent que les $2^b$-UCSs et $3^b$-UCSs sont UP-résilients par rapport à tout ordre d'application de la max-résolution, ce qui explique pourquoi leur apprentissage a un impact positif indépendamment de l'ordre utilisée.

En revanche, des études empiriques sur le solveur AHMAXSAT dans [1] montrent que l'apprentissage des $4^b$-UCSs et des $5^b$-UCSs a un impact négatif sur ses performances. Cette différence entre les résultats théoriques et empiriques peut s'expliquer par l'inadéquation des ordres d'application de la max-résolution utilisés. En effet, il a été montré dans [3] que l'ordre a un impact sur l'UP-résilience des transformations en comparant deux heuristiques : l'ordre de propagation inverse (RPO) et le plus petit résolvant intermédiaire (SIR). En particulier, le pourcentage moyen d'UP-résilience des transformations est comparativement plus élevé avec l'heuristique SIR. Dans le cas des $k^b$-UCSs, l'heuristique SIR devient inutilisable puisque tous les résolvants intermédiaires ont la même taille (binaires) comme indiqué dans la preuve du théorème 1. De plus, il est facile de vérifier que RPO n'assure pas toujours l'UP-résilience des transformations des $k^b$-UCSs quand $k > 3$.

La discussion précédente nous amène aux perspectives et applications de notre résultat, qui incluent la mise en œuvre et le test de l'ordre circulaire qui assure l'UP-résilience des $k^b$-UCSs afin d'étendre les motifs de l'état de l'art. De plus, nous avons réfléchi à plusieurs perspectives théoriques de notre résultat. La première est sa généralisation aux k-UCSs où exactement une clause de n'importe quelle taille participe au conflit et toutes les autres clauses sont binaires. La seconde est de trouver un ordre d'application de la max-résolution qui maximise le pourcentage de l'UP- résilience pour les k-UCSs et, plus généralement, pour tout ensemble inconsistant.

## References

[1] A. Abramé and D. Habet. On the extension of learning for Max-SAT. In U. Endriss and J. Leite, editors, *STAIRS 2014*, volume 241, pages 1–10, 2014.

[2] A. Abramé and D. Habet. AHMAXSAT : Description and evaluation of a branch and bound Max-SAT solver. *JSAT*, 9 :89–128, 2015.

[3] A. Abramé and D. Habet. On the resiliency of unit propagation to max-resolution. In Q. Yang and M. Wooldridge, editors, *IJCAI 2015*, pages 268–274, 2015.

[4] M. L. Bonet, J. Levy, and F. Manyà. Resolution for max-sat. *JAIR*, 171 :606 – 618, 2007.

[5] C. M. Li, F. Manyà, N. O. Mohamedou, and J. Planes. Resolution-based lower bounds in maxsat. *Constraints*, 15 :456–484, 2010.

[6] C.-M. Li, F. Manyà, and J. Planes. Detecting disjoint inconsistent subformulas for computing lower bounds for max-sat. In *AAAI-06*, pages 86–91, 2006.

[7] C. M. Li, F. Manyà, and J. Planes. New inference rules for max-sat. *JAIR*, 30 :321–359, 2007.

[8] J. P. Marques-Silva and K. A. Sakallah. Grasp : A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48 :506 – 521, 1999.

# Améliorations de l'hybridation PPC et PLNE pour la somme coloration

**Samba Ndojh NDIAYE**
Université de Lyon LIRIS
Université Lyon 1, LIRIS, UMR5205, F-69622 France
samba-ndojh.ndiaye@liris.cnrs.fr

**Résumé**

La somme coloration d'un graphe est une déclinaison du problème classique de coloration d'un graphe. Outre le calcul d'une coloration correcte dans laquelle deux sommets voisins doivent avoir des couleurs différentes, la somme des poids associés aux couleurs doit être minimale. Nous avons proposé dans [20] un cadre de résolution de ce problème qui repose sur une combinaison de la programmation par contraintes et la programmation linéaire en nombres entiers à travers une décomposition arborescente à deux niveaux. Nous proposons ici une nouvelle instanciation de ce cadre qui permet une amélioration significative des résultats expérimentaux.

## 1 Introduction

La somme coloration est une déclinaison du problème classique de coloration d'un graphe. L'objectif de ce problème demeure de trouver une coloration correcte, deux sommets voisins doivent avoir des couleurs différentes, mais en plus, la somme des poids associés aux couleurs doit être minimale. Il existe une grande variété de problèmes réels d'allocation de ressources pouvant être modélisés comme un problème de somme coloration de graphes [15]. Toutefois, la plupart des travaux de la littérature portant sur ce problème considèrent les premiers entiers non nuls comme les poids associés aux couleurs et n'imposent aucune contrainte au départ sur les couleurs utilisables pour les sommets du graphe. Cela rend le problème particulièrement difficile avec, notamment, un espace de recherche de très grande taille. Cela explique sans doute le peu de méthodes complètes proposées pour sa résolution. [9] constitue un survey complet des méthodes proposées pour sa résolution. Dans cet article, on note l'existence de méthodes complètes basées sur la programmation par contraintes (PPC) et la programmation en nombres entiers (PLNE) que nous avons améliorées de manière significative dans [20]. Toutefois, nous avons noté des comportements très différents de ces deux méthodes. D'une part, la PLNE est d'une très grande efficacité et parvient à calculer une solution optimale et en faire la preuve pour près de la moitié des instances du benchmark considéré. Toutefois, elle nécessite un espace mémoire très conséquent, dépassant la mémoire disponible pour 20% des instances du benchmark. D'autre part, la PPC nécessite très peu de mémoire et parvient donc à calculer une solution de qualité pour une grande partie des instances. Cependant, elle échoue généralement à réaliser des preuves d'optimalité. Nous avons donc proposé dans [20] un cadre générique pour la combinaison des deux approches pour limiter l'espace mémoire requis tout en conservant la capacité à calculer des solutions proches de l'optimum et à faire des preuves d'optimalité. Elle est basée sur une décomposition arborescente à deux niveaux du graphe : la racine et les feuilles. La programmation par contraintes est utilisée pour énumérer les solutions au niveau de la racine de l'arbre de la décomposition arborescente. Les sous-problèmes ainsi obtenus sont résolus par le biais de la programmation linéaire. L'objectif est de limiter l'espace mémoire nécessaire à la résolution du problème par la PLNE en limitant celle-ci à des sous-problèmes de taille réduite. Malgré des résultats encourageants sur un certain nombre d'instances, les résultats globaux étaient assez décevants. L'instanciation du cadre fait moins de preuves que la PLNE, mais elle a aussi une distance moyenne aux solutions de référence plus élevées que cette dernière. Nous proposons ici une nouvelle instanciation de ce cadre. Elle intègre un paramètre pour contrôler la taille de la racine de la décomposition nous permettant ainsi de naviguer d'une utilisation de la PPC unique-

ment à celle de la PLNE uniquement, en passant par des combinaisons avec un degré d'utilisation des deux approches plus ou moins important. Elle intègre également la stratégie LDS (Limited Decrepancy Search [6]) lors de l'énumération des solutions au niveau de la racine pour se concentrer sur les solutions les plus prometteuses dans un premier temps afin de trouver le plus rapidement possible une solution de qualité. Enfin, nous avons défini une stratégie de sélection automatique du meilleur paramétrage dans un portfolio de paramètres pré-définis. La méthode obtenue parvient à faire 66 preuves d'optimalité sur les 126 instances du benchmark, le plus grand nombre à l'heure actuelle. En même temps, elle parvient à maintenir une distance moyenne aux solutions de références à 6.20%.

La section suivante rappelle les principales définitions et propriétés du problème de la somme coloration. Ensuite, la section 3 présente les principales techniques de résolution complète. Elle détaille nos contributions déjà présentées dans [20], notamment le cadre générique BFD (Backtracking on flower decomposition). Notre nouvelle instanciation est définie dans la section 4, suivie par une étude expérimentale.

## 2 Le problème de la somme coloration

**Définition 1** *Un graphe non orienté $G = (V, E)$ est défini par un ensemble de sommets $V$ et un ensemble d'arêtes $E \subseteq V \times V$. Une arête est une paire de sommets non ordonnés.*

On note $\deg(v)$ le degré d'un sommet $v$ du graphe, *i.e.* $\deg(v) = |\{u \in V, \{u, v\} \in E\}|$, et $\Delta(G)$ le degré maximum d'un sommet du graphe, *i.e.* $\Delta(G) = \max\{\deg(v), v \in V\}$.

**Définition 2** *Une clique d'un graphe est un sous-ensemble de sommets du graphe tel que tous les sommets sont reliés deux à deux.*

**Définition 3** *Un stable ou ensemble de sommets indépendants d'un graphe est un sous-ensemble de sommets du graphe tel qu'il n'existe aucune arête reliant deux sommets du sous-ensemble.*

**Définition 4** *Une $k$-coloration propre d'un graphe $G = (V, E)$ est une fonction $c : V \to [1, k]$ telle que $\forall \{x, y\} \in E, c(x) \neq c(y)$.*

Dans une $k$-coloration, $k$ est le nombre maximum de couleurs différentes utilisées. Cette coloration peut également être vue comme une partition de $V$ en $k$ sous-ensembles indépendants de $G$. En effet, les sommets partageant la même couleur forment nécessairement un stable du graphe dans la mesure où il ne peut exister une arête reliant deux sommets de l'ensemble.

Le problème classique de coloration d'un graphe consiste à calculer une $k$-coloration propre qui minimise le nombre de couleurs utilisées. La valeur minimum de $k$ est appelée nombre chromatique du graphe. Dans cet article, nous nous intéressons au problème de somme coloration qui consiste à calculer une $k$-coloration propre qui minimise la somme des poids associés aux couleurs des sommets du graphe. Dans la littérature, les poids possibles pour les couleurs sont souvent restreints aux premiers entiers non nuls [16]. Dans ce cas, chaque couleur est identifiée par son poids et le but est de trouver une fonction $c : V \to [1, \Delta(G)]$ qui minimise $\sum_{x \in V} c(x)$. La valeur minimale de cette somme est appelée somme chromatique du graphe et est notée $\Sigma(G)$.

À l'image du problème de coloration classique, une solution de la somme coloration définit une partition des sommets du graphe en stables. En effet, chaque couleur de la coloration définit un stable dans la mesure où il n'existe pas d'arête entre deux sommets ayant la même couleur. [18] en déduit la notion de dominance entre colorations.

**Définition 5** *Une somme coloration du graphe est dominée si la somme des couleurs peut être réduite en réaffectant les couleurs aux différents stables de la partition qu'elle définit.*

Étant donnée une partition en stables des sommets du graphe, une coloration dominante est obtenue en ordonnant les stables par taille décroissante et en leur affectant les couleurs dans l'ordre croissant des poids.

**Définition 6** *[18] La force d'un graphe est le nombre minimum de couleurs nécessaire à la construction d'une solution optimale du problème de somme coloration.*

## 3 Techniques de résolution

### 3.1 Méthodes incomplètes

La majorité des méthodes de résolution reposent sur une approche incomplète qui propose des bornes inférieures, supérieures voire les deux [9]. Elles parviennent à proposer des bornes supérieures de grande qualité. C'est moins le cas pour les bornes inférieures. De ce fait, la combinaison des résultats obtenus dans la littérature suffit uniquement pour faire la preuve d'optimalité (la meilleure borne supérieure est égale à la meilleure borne inférieure) de 15 instances sur le benchmark classique pour ce problème qui en contient 126.

### 3.2 Méthodes complètes

Il existe peu de méthodes complètes pour la résolution du problème de somme coloration. Elles sont basées

essentiellement sur la technique du branch and bound [17], la programmation par contraintes [17] et la programmation linéaire en nombres entiers [26]. La PLNE demeure de loin l'approche la plus performante avec la résolution à l'optimum de 61 instances du benchmark. Toutefois, cette approche requiert un espace mémoire très important qui peut le rendre totalement inopérant sur des graphes de grande taille. Pour y remédier, nous avons proposé plusieurs améliorations sur ce modèle, ainsi que sur le modèle de PPC [20]. Certes, ce dernier requiert moins de mémoire, mais il peine à réaliser des preuves d'optimalité.

### 3.2.1 Améliorations du modèle PPC

Le premier modèle PPC a été défini dans [17]. Il associe à chaque sommet du graphe $u \in V$, une variable $x_u$ dont le domaine est $D(x_u) = [1, \Delta(G) + 1]$. La valeur de $x_u$ correspond à la couleur affectée au sommet $u$ (plus précisément, $x_u$ correspond au poids de la couleur affectée au sommet $u$ sachant que chaque couleur est identifiée par son poids dans notre contexte). Il inclut également une contrainte de différence pour toute arête du graphe, *i.e.*, $\forall \{u, v\} \in E, x_u \neq x_v$. La fonction objectif à minimiser est la somme des variables, *i.e.*, $\sum_{u \in V} x_u$.

La première amélioration a consisté à réduire la taille des domaines en utilisant la propriété suivante.

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

Ainsi, le domaine de chaque variable $x_u$ peut être limité à $D(x_u) = [1, \deg(u) + 1]$.

Le modèle initial se limite à la propagation de contraintes de différence alors que dans les cliques du graphe, une propagation de la contrainte globale Tous Différents (AllDifferent) peut s'avérer bien plus efficace. De ce fait, nous avons adapté le modèle en y intégrant cette contrainte globale. Pour cela, nous calculons de manière gloutonne des cliques maximales du graphe et remplaçons les contraintes de différence qu'elles induisent par des contraintes globales AllDifferent.

### 3.2.2 Améliorations du modèle PLNE

Le modèle proposé par [26] associe une variable binaire $x_{uk}$ à chaque paire $(u, k) \in V \times [1, \Delta(G) + 1]$, tel que $x_{uk}$ a pour valeur 1 ssi le sommet $u$ a pour couleur $k$. L'objectif demeure de minimiser la somme des poids associés aux couleurs des sommets, *i.e.* $\sum_{u=1}^{|V|} \sum_{k=1}^{\Delta(G)+1} k \cdot x_{uk}$ tel que chaque sommet $u \in V$ n'a qu'une couleur, *i.e.*, $\sum_{k=1}^{\Delta(G)+1} x_{uk} = 1$, et pour chaque arête $\{u, v\} \in E$, $u$ et $v$ ont des couleurs différentes, *i.e.*, $x_{uk} + x_{vk} \leq 1, \forall k \in [1, \Delta(G) + 1]$.

Nous avons intégré à ce modèle les deux améliorations du modèle PPC. Tout d'abord, la réduction des domaines qui permet ici de réduire le nombre de variables binaires. On a une variable $x_{uk}$ uniquement pour une paire $(u, k) \in V \times [1, \deg(u) + 1]$. Nous avons également intégré une décomposition en cliques qui permet d'avoir une vision plus globale en précisant que toutes les variables appartenant à la même clique doivent avoir des couleurs différentes.

## 3.3 BFD : combinaison PPC et PLNE

Le modèle PLNE requiert un espace mémoire trop important pour la résolution d'instances de grande taille. Nous avons donc défini un cadre général de résolution basé sur une décomposition arborescente du graphe afin de cantonner ce modèle à la résolution de sous-problèmes de taille réduite. Cette stratégie est similaire à la méthode BTD (Backtracking on Tree Decomposition) [8, 4]. Cependant, elle repose sur une décomposition arborescente à 2 niveaux, la racine et les feuilles.

**Définition 7** *[23] Une* décomposition arborescente *d'un graphe $(V, E)$ est un couple $(K, T)$ où $T = (I, F)$ est un arbre, et $K : I \to \mathcal{P}(V)$ est une fonction qui associe à un sous-ensemble de sommets $K_i \subseteq V$ (appelé cluster) à chaque nœud $i \in I$ tel que les propriétés suivantes sont garanties :*
(i) $\cup_{i \in I} K_i = V$ ;
(ii) *pour chaque arête $(v_j, v_k) \in E$, il existe un nœud $i \in I$ tel que $\{v_j, v_k\} \subseteq K_i$ ;*
(iii) *pour tout $i, j, k \in I$, si $k$ est sur le chemin entre $i$ et $j$ dans $T$, alors $K_i \cap K_j \subseteq K_k$.*

La principale propriété à retenir d'une décomposition arborescente est qu'après l'affectation des variables à la racine, ses clusters fils définissent des sous-problèmes totalement indépendants qu'il est possible de résoudre séparément. Par ailleurs, la solution d'un sous-problème peut être sauvegardée afin d'éviter de devoir le résoudre à nouveau.

**Définition 8** *Une* décomposition en fleur *d'un graphe $G = (V, E)$ est une décomposition arborescente de $G$ qui contient uniquement un cluster racine et ses clusters fils. Ces derniers sont des feuilles de l'arbre de la décomposition car ils n'ont pas de descendance.*

Le cadre, nommé BFD (Backtracking on flower decomposition) et présenté dans l'algorithme 1, intègre deux fonctions capables de résoudre un problème d'optimisation sous contraintes : SolveurPPC, un solveur PPC, et SolveurPLNE, un solveur PLNE. Elle prend en entrée un modèle PPC du problème de somme coloration, une décomposition en fleur du graphe et

**Algorithme 1 :** BFD$\big((X, D, C, f),\ (\mathcal{C}, \mathcal{T}),\ r,$ *maxcost*$\big)$

**Entrées :** $(X, D, C, f)$ un modèle PPC ;
$(\mathcal{C}, \mathcal{T})$ une décomposition arborescente ;
$r$ la racine de $\mathcal{T}$ ;
*maxcost* un coût maximum acceptable.

**Output :** La valeur optimale de $(X, D, C, f)$ si elle est inférieure à *maxcost* ;
$maxcost + 1$, sinon.

1 Soit $\mathcal{P}$ le CSP $(X, D, C)$ réduit aux variables de $\mathcal{C}_r$;
2 $best \leftarrow maxcost + 1$;
3 Soit $Ch$ les clusters fils de $r$ dans $\mathcal{T}$;
4 **pour chaque** *solution $S$ de $\mathcal{P}$ calculées par* SolveurPPC **faire**
5  **pour chaque** *fils $i \in Ch$* **faire**
6  $\text{LB}_i \leftarrow init\text{LB}_i$;
7  $\text{UB}_i \leftarrow init\text{UB}_i$;
8  Soit $A_i = S[\mathcal{C}_r \cap \mathcal{C}_i]$;
9  **si** *il existe un good valué* $(A_i, \text{LB}, \text{UB})$ **alors**
10  $\text{LB}_i \leftarrow \text{LB}$ de $(A_i, \text{LB}, \text{UB})$;
11  $\text{UB}_i \leftarrow \text{UB}$ de $(A_i, \text{LB}, \text{UB})$;
12  **si** $f(S) + \sum_{i \in Ch} \text{LB}_i < best$ **alors**
13  **tant que** $f(S) + \sum_{i \in Ch} \text{LB}_i < best$ **et** *il existe un fils $i \in Ch$ tq $A_i = S[\mathcal{C}_r \cap \mathcal{C}_i]$ n'est pas un good valué optimal* **faire**
14  Soit $i$ un cluster fils de $r$ tq $\text{LB}_i < \text{UB}_i$ ;
15  Soit $\mathcal{T}_i$ le sous-arbre de $\mathcal{T}$ enraciné en $i$;
16  Soit *newD* les domaines courants;
17  $(X', C', f') \leftarrow$ ConvertirPpcPlne$\big((X, newD, C, f), i\big)$;
18  $maxcost_i \leftarrow \min\{best - 1 - f(S) - \sum_{j \in Ch, i \neq j} \text{LB}_j, \text{UB}_i - 1\}$;
19  $best_i \leftarrow$ SolveurPLNE$\big((X', C', f'), maxcost_i\big)$;
20  **si** $best_i \leq maxcost_i$ **alors**
21  Enregistrer le good valué $(A_i, best_i, best_i)$;
22  **sinon**
23  Enregistrer le good valué $(A_i, best_i, \text{UB}_i)$;
24  $best_r \leftarrow f(S) + \sum_{i \in Ch} \text{LB}_i$;
25  **si** $best_r < best$ **alors**
26  $best \leftarrow best_r$;
27 **retourner** *best*;

éventuellement un coût maximum à ne pas dépasser. Ce coût maximum peut être initialisé à l'aide d'une borne supérieure de la somme coloration du graphe. La fonction SolveurPPC va énumérer toutes les solutions sur la racine. Il s'agit de colorations propres sur les sommets de la racine dont la borne inférieure sur une extension à tous les sommets du graphe laisse la possibilité d'améliorer la meilleure solution courante *best*. Cette borne inférieure repose sur le coût minimal de la coloration de chaque cluster fils calculé grâce à des bornes théoriques. Pour chaque solution trouvée par SolveurPPC, l'affectation des variables du séparateur entre la racine et un cluster fils (il s'agit des variables présentes à la fois dans la racine et le cluster fils) définit un nouveau sous-problème sur ce cluster fils. Pour chaque sous-problème ainsi défini, nous commençons par calculer les bornes inférieure et supérieure de la somme coloration du sous-graphe induit par les sommets du cluster. Soit le problème a déjà été résolu (pas nécessairement à l'optimum) et nous récupérons les bornes sauvegardées dans le good valué associé. Un good valué d'un cluster est formé par une affectation des variables du séparateur avec son cluster père et les bornes supérieure et inférieure de la somme coloration du sous-graphe induit par les sommets du cluster. S'il s'agit de la première génération de l'affectation sur le séparateur, alors on calcule des bornes théoriques. Si la somme du coût de la solution sur la racine et des bornes inférieures sur les sous-problèmes induits par les clusters fils atteint ou dépasse le coût de la meilleure solution, alors on passe à la solution suivante à la racine. En effet, nous avons la certitude qu'il est impossible d'améliorer la meilleure solution courante. Dans le cas contraire, on considère un sous-problème dont la solution optimale n'est pas encore connue (sa borne inférieure est strictement plus petite que sa borne supérieure), on le modélise en PLNE grâce à la fonction ConvertirPpcPlne qui permet donc de passer du modèle courant en PPC à un modèle en PLNE. Puis, on résout le sous-problème avec la fonction SolveurPLNE. On détermine le coût maximum acceptable pour ce sous-problème $maxcost_i$ qui laisse la possibilité d'améliorer la meilleure solution courante. La fonction SolveurPLNE nous renvoie la valeur $maxcost_i + 1$ si le sous-problème n'admet pas de solution de coût inférieur ou égal à $maxcost_i$ ou une valeur inférieure ou égale à $maxcost_i$ correspondant au coût de la solution optimale du sous-problème. Si le problème est résolu à l'optimum, on enregistre le good valué associé qui nous évitera de résoudre le problème à nouveau. Sinon, on enregistre un good valué dont la borne inférieure est à $maxcost_i + 1$ et la borne supérieure reste à la valeur initiale. Ce good valué non optimal permettra à la prochaine résolution du sous-

problème de partir d'une borne inférieure de meilleure qualité que la borne théorique. Si après la résolution d'un sous-problème, la nouvelle borne inférieure ne laisse plus la possibilité d'améliorer la valeur *best*, alors on revient à la racine pour passer à la solution suivante. Si on parvient à résoudre tous les sous-problèmes en restant en dessous de la valeur *best* alors on vient de trouver une nouvelle meilleure solution et on met à jour cette valeur.

Ce cadre général peut être instancié de différentes manières avec plusieurs paramétrages possibles pour les fonctions SolveurPPC et SolveurPLNE. On peut également définir plusieurs stratégies pour calculer la décomposition en fleur.

## 4 Une instanciation du cadre BFD

### 4.1 Calcul d'une décomposition en fleur

Nous définissons ici une méthode de décomposition qui prend en paramètre la taille maximale à ne pas dépasser pour la taille de la racine *maxrac*. Tout d'abord, nous utilisons une méthode classique de calcul de décomposition arborescente [7]. Elle commence par trianguler le graphe à l'aide de l'heuristique Minfill [10]. Ensuite, elle détermine les cliques maximales du graphe triangulé qui vont constituer les clusters de la décomposition. Enfin, elle calcule un arbre garantissant la propriété (iii) d'une décomposition arborescente.

À partir de cette décomposition arborescente, nous allons choisir les sommets qui vont apparaître dans la racine de la décomposition en fleur. Cette racine est composée d'un sous-ensemble de l'ensemble des séparateurs de la décomposition arborescente initiale. L'objectif est de construire une racine de taille réduite permettant malgré tout d'avoir des clusters feuilles de taille réduite. Afin de calculer ce sous-ensemble de séparateurs, nous modélisons notre problème en PPC avec une variable binaire associée à chaque séparateur. Nous intégrons à ce modèle la contrainte sur la taille maximale autorisée pour la racine *maxrac*. La fonction objectif est de minimiser la taille maximale des clusters fils, plus précisément le nombre maximal de sommets appartenant à un clusters fils et pas à la racine.

Si ce problème n'admet pas de solution (il n'existe aucun sous-ensemble de séparateurs de taille inférieure à *maxrac*), alors nous construisons une solution gloutonne. Elle consiste à choisir *maxrac* sommets du graphe dans l'ordre décroissant de leur degré pour former la racine de la décomposition, puis on en déduit les clusters fils à partir des composantes connexes induites par le retrait des sommets de la racine.

Cette approche permet de décomposer tous les graphes y compris ceux qui ne sont pas structurés en calculant dans le pire des cas, une décomposition avec deux clusters, la racine et son fils. Il est même possible que l'intersection entre les deux contienne tous les sommets de la racine rendant inutile la sauvegarde de good valué. Cela peut demeurer intéressant dans le cadre BFD de disposer de ce type de décomposition qui réduit malgré tout la taille des sous-problèmes à résoudre avec la fonction SolveurPLNE. Dans le cas général, on peut limiter la taille des goods valués en les enregistrant uniquement sur des séparateurs de taille bornée par un paramètre.

Afin que le cadre BFD puisse capturer, en plus des combinaisons PPC et PLNE, la résolution du problème de somme coloration par PPC ou PLNE uniquement, nous avons choisi d'étendre la notion de décomposition en fleur. Si $maxrac = 0$, nous construisons une décomposition avec deux clusters, la racine et son fils, et la racine est un cluster vide. Ainsi, la résolution se limite à un appel à la fonction SolveurPLNE. L'autre cas extrême intervient quand $maxrac \geq |V|$. Nous obtenons également une décomposition avec deux clusters, mais cette fois-ci, le cluster fils est vide. Ainsi, la résolution consiste énumérer les solutions de la racine qui recouvre tout le problème jusqu'à trouver une solution optimale.

### 4.2 Paramétrage de SolveurPPC

Les solveurs de contraintes utilisent généralement la technique du Branch and Bound pour résoudre les problèmes d'optimisation. Son efficacité repose, entre autres, sur la qualité de la borne inférieure évaluée à chaque nœud de l'arbre de recherche, les heuristiques de choix de variables et de valeurs et la stratégie d'exploration de l'espace de recherche.

#### 4.2.1 Borne inférieure [20]

Nous avons proposé une technique d'évaluation d'une borne inférieure plus performante que la stratégie par défaut qui consiste à faire la somme des valeurs minimales des domaines courants de toutes les variables. Elle repose sur une décomposition en cliques du graphe pour mieux prendre en compte les interactions entre les variables. La borne inférieure globale est constituée de la borne inférieure de chaque clique de la décomposition. Pour chaque clique de taille $k$, il suffit de faire l'union des domaines courants des variables associées et de faire la somme des $k$ plus petites valeurs de cet ensemble. On obtient ainsi une borne de bonne qualité à un coût assez réduit. Cette qualité est équivalente à celle obtenue en utilisant la combinaison des contraintes globales AllDifferent et Somme définie dans [1]. En effet, la similarité entre les domaines des variables (les domaines contiennent beaucoup de valeurs identiques) dans notre

contexte minore l'efficacité de cette approche. Elle ne parvient pas à fournir de meilleurs bornes alors que sa durée d'exécution est plus importante.

#### 4.2.2 Heuristiques [20]

Nous avons défini une heuristique de choix de variables adaptée au problème qui donne de meilleurs résultats que les heuristiques génériques les plus performantes telles que wdeg [2] et activity [19]. Elle utilise le fait que le meilleur choix possible pour la valeur à affecter à une variable est la plus petite valeur de son domaine afin de ne pas faire augmenter trop brusquement la somme des variables. De ce fait, elle propose de choisir comme prochaine variable à affecter celle dont le domaine contient la plus petite valeur. En cas d'égalités, on choisit la variable avec le moins de variables voisines dont le domaine contient cette valeur. Ainsi, l'affectation de cette variable à sa plus petite valeur va retirer cette valeur du minimum de domaines des variables voisines. S'il subsiste des égalités, on choisit de manière aléatoire. Ce dernier choix aléatoire permet d'intégrer une politique de restarts [5] afin de mieux explorer l'espace de recherche. On évite donc une exploration exhaustive prématurée de zones de l'espace de recherche ne contenant pas nécessairement des solutions de qualité.

#### 4.2.3 Colorations dominantes [18]

Nous avons intégré à la résolution la technique du swapping qui permet de transformer toute nouvelle coloration calculée en coloration dominante en échangeant les couleurs des différents ensembles stables ainsi calculés. Ainsi, toute décomposition en stables induite par une coloration ne sera calculée qu'une fois nous permettant d'éviter la génération de nombreuses solutions symétriques correspondant à des colorations dominées.

#### 4.2.4 Force d'un graphe [18]

Nous utilisons l'algorithme défini dans [18] qui permet de borner éventuellement la force d'un graphe à chaque nouvelle solution trouvée. On peut ainsi rajouter des contraintes limitant la taille des domaines au fur et à mesure de l'amélioration de la qualité des solutions calculées.

#### 4.2.5 Stratégie de recherche

La recherche en profondeur d'abord, DFS, constitue la stratégie par défaut pour la résolution complète des problèmes en PPC. Pour assurer la complétude, DFS peut considérer des solutions de piètre qualité au niveau de la racine avant d'autres plus prometteuses. La stratégie LDS (Limited Decrepancy Search) [6] souffre moins de ce problème. En effet, LDS prend en paramètre un entier $k$ qui limite le nombre de valeurs différentes du choix initial de l'heuristique de choix de valeurs. Elle permet dans notre contexte de considérer dans un premier temps des solutions de la racine utilisant des couleurs de poids faibles et donc de coût moindre. Pour rétablir la complétude, il est nécessaire de faire croître la valeur de $k$ jusqu'à l'exploration de la totalité de l'espace de recherche. Nous avons choisi d'initialiser $k = 1$, puis de lui faire suivre une croissance géométrique en multipliant sa valeur par 2 à chaque fois que toutes les solutions de la racine pour une valeur donnée de $k$ ont été considérées. Pour éviter une trop lente marche vers la complétude, nous avons choisi de borner la valeur de $k$ à $2 * |V|$. Au-delà, elle est fixée directement à une valeur permettant une exploration exhaustive de l'espace de recherche du type de DFS.

### 4.3 Paramétrage de SolveurPLNE

Notre paramétrage de la fonction SolveurPLNE dépend essentiellement des options offertes par le solveur sous-jacent. L'objectif est de limiter au minimum l'espace mémoire requis afin de mieux passer à l'échelle sur les graphes de grande taille.

## 5 Étude expérimentale

Nous avons choisi le solveur Gecode 6.0.1 [24] pour implémenter la fonction SolveurPPC. Nous avons utilisé la cohérence de borne des propagateurs par défaut des contraintes. La fonction SolveurPLNE a été implémentée grâce au solveur ILOG CPLEX 12.6.2 [3]. Nous avons choisi un paramétrage de CPLEX optimisant l'espace mémoire requis (la stratégie DFS et cuts factor à 1.5) sans que cela ne nuise à l'efficacité du solveur.

Nos expérimentations ont été menées sur des machines équipées d'un processeur Intel Xeon E5-2670 0 à 2,60 Ghz, disposant de 20 480 KB de mémoire cache et de 4 GB de RAM. Les programmes sont écrits en C/C++. Nous avons fixé une durée limite d'exécution à 24h au regard de la difficulté du problème.

Nous considérons les 126 instances du benchmark couramment utilisé pour le problème de somme coloration [26, 9]. Certaines ont été ajoutées par COLOR02/03/04 [1], mais la plupart sont simplement les instances DIMACS conçues pour le problème de coloration classique [2]. Le nombre de sommets des graphes varie de 11 à 4146, alors que le nombre d'arêtes passe

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

de 20 à 307350, définissant ainsi des tailles et des structures variées pour ces instances. La littérature fournit des bornes inférieures et supérieures pour 92 instances [26, 9]. Ces valeurs constituent les bornes de références. Pour les autres, nous utilisons comme valeur de référence la meilleure borne supérieure fournie par les modèles PPC de [17] et PLNE [26]. Les tableaux de cette section intégrent ces valeurs de référence et positionnent souvent nos résultats par rapport à celles-ci à travers la notion de distance. La distance entre une borne calculée $b$ et une borne de référence $r$ est donnée par le ratio $(b - r)/r$.

Nous présentons dans cette section plusieurs tableaux résumant les résultats des expérimentations et permettant d'avoir une vision globale du comportement des différentes méthodes. Ces tableaux ont une partie commune qui présente les distances minimales (Min), moyennes (Moy) et maximales (Max) par rapport aux valeurs de référence. Les lignes "Preuves" donnent le nombre de preuves d'optimalité réalisées, "Mem. out" le nombre d'instances dont la résolution a échoué par manque de mémoire, "Ref." le nombre d'instances pour lesquelles la solution trouvée est de qualité au moins équivalente à la solution de référence et "Best" le nombre d'instances pour lesquelles une méthode est meilleure que les autres méthodes considérées. Une méthode est meilleure qu'une autre sur une instance si elle parvient à faire la preuve plus rapidement, ou à défaut si elle obtient une solution de meilleure qualité, ou si elle obtient cette solution plus rapidement.

### 5.1 DFS vs LDS

Nous commençons par une comparaison des stratégies d'exploration LDS et DFS (utilisée par la stratégie BAB dans Gecode). Les résultats résumés dans la table 1 montrent un meilleur comportement de LDS. Comme on pouvait s'y attendre, le fait d'explorer en priorité des solutions prometteuses à la racine permet de trouver plus rapidement de meilleures solutions au problème. Cela se traduit par une distance moyenne aux solutions de référence bien inférieure pour LDS (14% de différence quand la taille de la racine est bornée à 150). C'est également le cas pour la ligne Ref. où la stratégie LDS se révèle toujours plus efficace à produire des solutions de référence. Les résultats peuvent paraître un peu plus surprenant concernant le nombre de preuves où LDS fait toujours aussi bien voire mieux que DFS qui est plus orienté vers une exploration exhaustive directe de l'espace de recherche. Sur la colonne 10, il parvient à réaliser 4 preuves de plus que DFS. En trouvant une meilleure solution plus rapidement, LDS permet d'améliorer la borne supérieure et de filtrer d'avantage de zones de l'espace de recherche. Cela aide à réaliser les preuves plus rapidement. Pour finir, une comparaison instance par instance des deux stratégies (ligne Best) montre une domination nette de LDS qui n'est battue que pour une taille de racine limitée à 2 où son effet est moins perceptible.

### 5.2 BFD : différentes tailles de racine

Nous avons évalué l'impact de la taille de la racine sur les résultats de BFD et résumé les résultats dans le tableau 2. Quand la taille de la racine est bornée par la valeur 0, BFD capture la méthode plne+ qui utilise le modèle amélioré présenté dans la section 3.2.2. On note donc l'efficacité de ce modèle qui permet de faire le plus grand nombre de preuves et de calculer le plus de solutions de référence. Son principal défaut réside dans le nombre d'échecs dus à un espace mémoire requis trop important. Malgré tout, on note que sur les 103 instances dont la résolution ne soulève pas de problème de mémoire, elle ne parvient pas à résoudre à l'optimum 38 instances en moins de 24h. À l'autre extrémité, nous avons bfd+$\infty$ qui correspond à ppc+. Ce dernier requiert un espace mémoire très limité et ne souffre pas d'échecs de cette nature. Il parvient de ce fait à calculer des solutions de qualité acceptable pour toutes les instances y compris celles de très grande taille. Il constitue la meilleure approche pour 30 instances du benchmark et parvient à calculer 46 solutions de référence. La distance moyenne aux solutions de référence des solutions qu'il produit est limitée à 5.46%. Sa grande faiblesse se situe au niveau du nombre de preuves réalisées qui reste à 8 sur 126 instances. Nous avons considéré plusieurs autres tailles possibles pour la racine allant de 2 à 950. Plus, la valeur est proche de 0, plus le comportement de BFD est proche de celui de plne+. À l'inverse, plus cette valeur est grande, plus le comportement est proche de celui de ppc+. Nous avons gardé dans le tableau 2, les valeurs qui ont une influence sensible sur le vbs (virtual best solver). Le vbs est un solveur virtuel qui choisit le meilleur paramétrage parmi les paramétrages de notre ensemble pour chaque instance. Cela permet donc d'avoir un ensemble de paramétrages complémentaires sur le benchmark. On ne choisit pas uniquement les meilleurs paramétrages, mais ceux qui couvrent efficacement le benchmark en donnant toujours de très bons résultats sur toute instance considérée. Ce sous-ensemble complémentaire est présenté dans ce tableau, ainsi que le vbs associé. Ce vbs réalise 66 preuves, obtient 83 solutions de référence et une distance moyenne aux solutions de référence à 4.18%. Il améliore la solution de référence de la littérature pour 9 instances du benchmark (voir tableau 4). Le vbs se montre d'une qualité indéniable. Toutefois, trouver le bon paramétrage pour une nouvelle instance peut s'avérer une tâche délicate.

Tableau 1 – Résumé des résultats de BFD pour les stratégies DFS et LDS et plusieurs tailles de racine de la décomposition. La première ligne donne les tailles maximales de la racine de la décomposition en fleur. Pour chaque taille de racine, nous avons une colonne qui donne les résultats de DFS et une autre ceux de LDS.

| | | 2 | | 5 | | 10 | | 150 | | 250 | | 400 | | 950 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | dfs | lds | dfs | lds | dfs | lds | dfs | lds | dfs | lds | dfs | lds | dfs | lds |
| Dist | Min | **−5,99** | **−5,99** | **−5,76** | **−5,76** | **−5,20** | **−5,20** | **0,00** | **0,00** | **−0,16** | **−0,16** | **−0,16** | **−0,16** | **−0,16** | **−0,16** |
| | Moy | 62,54 | **52,89** | 64,40 | **55,56** | 71,18 | **66,71** | 59,35 | **45,25** | 57,53 | **51,62** | 36,69 | **35,20** | 15,11 | **6,09** |
| | Max | 1 119,50 | **864,70** | 1 119,50 | **864,70** | **1 119,50** | **1 119,50** | 1 119,50 | **864,70** | **1 119,50** | **1 119,50** | **864,70** | **864,70** | 1 119,50 | **72,33** |
| Preuves | | **61** | **61** | **36** | **36** | 15 | **19** | **7** | **7** | **7** | **7** | 6 | **7** | 6 | **7** |
| Mem. out | | **20** | **20** | **20** | **20** | **19** | 21 | **10** | 11 | **10** | **10** | **7** | **7** | **1** | **1** |
| Ref. | | 71 | **73** | 45 | **51** | 21 | **50** | 26 | **32** | 31 | **34** | 31 | **36** | 41 | **42** |
| Best | | **92** | 69 | 59 | **103** | 46 | **114** | 47 | **89** | 57 | **82** | 48 | **88** | 55 | **72** |

Tableau 2 – Résumé des résultats de BFD avec la stratégie LDS et différentes tailles de racine de la décomposition. plne+ correspond à bfd+0, ppc+ à bfd+∞, vbs au virtual best solver et sélecteur est le résultat de la sélection automatique d'algorithmes.

| | | plne+ | bfd+5 | bfd+10 | bfd+150 | bfd+250 | bfd+950 | ppc+ | vbs | sélecteur |
|---|---|---|---|---|---|---|---|---|---|---|
| Dist | Min | −2,55 | **−5,76** | −5,20 | 0,00 | −0,16 | −0,16 | −0,16 | **−5,76** | −2,55 |
| | Moy | 63,80 | 55,56 | 66,71 | 45,25 | 51,62 | 6,09 | 5,46 | **4,18** | 6,20 |
| | Max | 1 119,50 | 864,70 | 1 119,50 | 864,70 | 1 119,50 | **72,33** | 72,58 | **72,33** | 101,64 |
| Preuves | | 65 | 36 | 19 | 7 | 7 | 7 | 8 | **66** | **66** |
| Mem. out | | 23 | 20 | 21 | 11 | 10 | 1 | **0** | **0** | 1 |
| Ref. | | 74 | 51 | 50 | 32 | 34 | 42 | 46 | **83** | 77 |
| Best | | 67 | 10 | 4 | 5 | 3 | 7 | 30 | **126** | 0 |

### 5.3 Sélection de paramétrage

Le but de cette section est de présenter une approche permettant de sélectionner un paramétrage de notre sous-ensemble de paramétrages pour résoudre une nouvelle instance en tenant compte de ses caractéristiques [22]. Cette approche a été utilisée avec un grand succès dans la littérature [27, 21, 13, 14, 25, 12]. Elle se décompose en deux étapes. La première étape consiste à extraire les caractéristiques de la nouvelle instance à résoudre. La seconde étape se résume à choisir un paramétrage de BFD pour résoudre l'instance.

#### 5.3.1 Caractéristiques

Étant donnée une nouvelle instance $G = (V, E)$, on calcule ses caractéristiques suivantes (si on dispose de plusieurs valeurs, on détermine les valeurs minimales, maximales, moyennes et la déviation standard) : $|V|$, $|E|$, les degrés des sommets, la taille des composantes connexes, le nombre de variables et de contraintes du modèle PLNE+, le nombre de contraintes AllDifferent (d'arité supérieure à 2) dans le modèle PPC+, l'arité des contraintes AllDifferent, la densité de la plus grande composante connexe, les bornes théoriques inférieure et supérieure de cette composante connexe, le nombre et la taille des clusters de la décomposition arborescente initiale et de la décomposition en fleur, la taille du produit cartésien des domaines des variables de la racine de la décomposition en fleur, la distance entre les bornes supérieure et inférieure des clusters, densité des clusters, la taille des séparateurs et leur densité, le nombre de variables et de contraintes des modèles PLNE+ des clusters feuilles.

#### 5.3.2 Modèle pour la sélection de paramétrages.

Nous avons choisi d'utiliser l'outil Llama [11] avec un paramétrage identique à [20]. Pour chaque instance du benchmark, nous entraînons notre modèle sur le reste du benchmark, puis nous utilisons le modèle appris pour choisir le paramétrage à utiliser pour résoudre l'instance. Le score associé à la performance d'un paramétrage sur une instance tient compte des critères suivants dans cet ordre : la capacité à faire la preuve, la durée de réalisation de cette preuve, la qualité de la borne supérieure sur la somme coloration, le temps de calcul de cette borne, l'absence d'échec dû à l'utilisation de la mémoire.

#### 5.3.3 Sélecteur.

La colonne sélecteur du tableau 2 donne les résultats obtenus en utilisant cette technique. Elle réussit à faire 66 preuves, autant que le vbs et donc une de plus que PLNE+, tout en gardant une distance moyenne aux solutions de référence à 6.20%. Elle parvient également à calculer 77 solutions de référence sur les 126 instances du benchmark dont 6 améliorent la solution de référence de la littérature. Sur la ligne Best, elle ne peut pas être le meilleur paramétrage sur une instance dans la mesure où il faut intégrer le temps d'extraction des caractéristiques (moins de 6mn en moyenne) et de choix du paramétrage (moins de 1 s). Le tableau 3 montre les paramétrages choisis par le sélecteur et le vbs. On note qu'il existe une marge de progression

Tableau 3 – Résumé des résultats de BFD avec la stratégie LDS et différentes tailles de racine de la décomposition. plne+ correspond à bfd+0, ppc+ à bfd+$\infty$, vbs au virtual best solver et sélecteur est le résultat de la sélection automatique d'algorithmes. les lignes vbs et sélecteur donnent le nombre de fois où chaque paramétrage a été choisi par la méthode.

| | plne+ | bfd(lds)+5 | bfd(lds)+10 | bfd(lds)+150 | bfd(lds)+250 | bfd(lds)+950 | ppc+ |
|---|---|---|---|---|---|---|---|
| vbs | 67 | 10 | 4 | 5 | 3 | 7 | 30 |
| sélecteur | 77 | 7 | 12 | 0 | 3 | 10 | 16 |

non négligeable pour le sélecteur qui fait un peu trop confiance à la PLNE et pas assez à la PPC. En outre, une technique d'apprentissage plus perfectionnée intégrant de la régression sur la taille de la racine de la décomposition en fleur pourrait nous permettre d'avoir une valeur encore plus adaptée que celles de notre portfolio de paramétrages et ainsi d'améliorer d'avantage les performances.

Tableau 4 – Résultats de vbs pour les instances du benchmark avec une nouvelle solution de référence. La colonne "Benchmark" contient le nom des instances, les bornes inférieure (LB) et supérieure (UB) de référence, les nombres de sommets $|V|$ et d'arêtes $|E|$. La colonne vbs donne la borne calculée (UB), sa distance à la solution de référence (Dist) et le paramétrage ayant trouvé cette solution.

| Benchmark | | | | | vbs | | |
|---|---|---|---|---|---|---|---|
| Instance | LB | UB | \|V\| | \|E\| | UB | Dist | bfd+? |
| DSJR500.1 | | 2 143 | 500 | 3 555 | **2 141** | **−0,09** | bfd+5 |
| ash331GPIA | | 1 432 | 662 | 4 181 | **1 419** | **−0,9** | bfd+5 |
| ash608GPIA | | 2 636 | 1 216 | 7 844 | **2 600** | **−1,36** | bfd+0 |
| will199GPIA | | 1 940 | 701 | 6 772 | **1 892** | **−2,47** | bfd+2 |
| le450_25b | | 3 350 | 450 | 8 263 | **3 349** | **−0,02** | bfd+0 |
| r125.5 | | 1 832 | 125 | 3 838 | **1 825** | **−0,38** | bfd+0 |
| r250.1c | | 5 951 | 250 | 30 227 | **5 941** | **−0,16** | bfd+950 |
| r250.5 | | 6 979 | 250 | 14 849 | **6 591** | **−5,55** | bfd+2 |
| r1000.1 | | 7 585 | 1 000 | 14 378 | **7 130** | **−5,99** | bfd+2 |

## 6 Conclusion

Nous avons proposé une nouvelle instanciation du cadre BFD qui a un comportement plus lisible que celle de [20] : comportement proche de celui de PLNE+ quand on donne plus de poids à la PLNE, comportement proche de celui de PPC+ quand on donne plus de poids à la PPC. Cela facilite un peu la tâche de notre méthode de sélection de paramétrage lui permettant de réaliser 66 preuves sur les 126 instances du benchmark standard et de calculer 77 solutions de référence. Il demeure possible d'améliorer ses performances avec une méthode d'apprentissage par régression capable de prédire efficacement la bonne taille de racine de la décomposition en fleur pour une nouvelle instance à résoudre.

## Références

[1] Nicolas Beldiceanu, Mats Carlsson, Thierry Petit et Jean-Charles Régin : An $O(nlog(n))$ bound consistency algorithm for the conjunction of an alldifferent and an inequality between a sum of variables and a constant, and its generalization. *In ECAI*, volume 12, pages 145–150, 2012.

[2] Frédéric Boussemart, Fred Hemery, Christophe Lecoutre et Lakhdar Sais : Boosting systematic search by weighting constraints. *In ECAI*, volume 16, page 146, 2004.

[3] ILOG CPLEX : High-performance software for mathematical programming and optimization, 2005.

[4] Simon De Givry, Thomas Schiex et Gerard Verfaillie : Exploiting tree decomposition and soft local consistency in weighted csp. *In AAAI*, volume 6, pages 1–6, 2006.

[5] Carla P Gomes, Bart Selman, Henry Kautz *et al.* : Boosting combinatorial search through randomization. *AAAI/IAAI*, 98:431–437, 1998.

[6] William D. Harvey et Matthew L. Ginsberg : Limited discrepancy search. *In In Proceedings IJCAI'95*, 1995.

[7] P. Jégou, S. N. Ndiaye et C. Terrioux : Computing and exploiting tree-decompositions for solving constraint networks. *In Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP-2005)*, pages 777–781, 2005.

[8] Philippe Jégou et Cyril Terrioux : Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, 146:43–75, 2003.

[9] Yan Jin, Jean-Philippe Hamiez et Jin-Kao Hao : Algorithms for the minimum sum coloring problem : a review. *Artificial Intelligence Review*, pages 1–28, 2016.

[10] U. Kjaerulff : Triangulation of Graphs - Algorithms Giving Small Total State Space. Rapport technique, Judex R.R. Aalborg., Denmark, 1990.

[11] Lars Kotthoff : LLAMA : leveraging learning to automatically manage algorithms. *CoRR*, abs/1306.1031, 2013.

[12] Lars Kotthoff : Algorithm selection for combinatorial search problems : A survey. *AI Magazine*, 35(3):48–60, 2014.

[13] Lars Kotthoff, Pascal Kerschke, Holger Hoos et Heike Trautmann : Improving the state of the art in inexact TSP solving using per-instance algorithm selection. *In LION 9*, 2015.

[14] Lars Kotthoff, Ciaran Mccreesh et Christine Solnon : Portfolios of subgraph isomorphism algorithms. *In Learning and Intelligent OptimizatioN Conference (LION 10)*. Springer, 2016.

[15] Marek Kubale : *Graph colorings*, volume 352. American Mathematical Soc., 2004.

[16] Ewa Kubicka et Allen J Schwenk : An introduction to chromatic sums. *In Proceedings of the 17th conference on ACM Annual Computer Science Conference*, pages 39–45. ACM, 1989.

[17] Clément Lecat, Chu-Min Li, Corinne Lucet et Yu Li : Exact methods for the minimum sum coloring problem. *In DPCP-2015*, pages 61–69, Cork, Ireland, Iran, 2015.

[18] Clément Lecat, Corinne Lucet et Chu-Min Li : Sum coloring : New upper bounds for the chromatic strength. *CoRR*, abs/1609.02726, 2016.

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

[20] Maël MINOT, Samba Ndojh Ndiaye et Christine Solnon : Combining CP and ILP in a tree decomposition of bounded height for the sum colouring problem. *In CPAIOR 2017*, Padova, Italy, juin 2017.

[21] Eoin O'Mahony, Emmanuel Hebrard, Alan Holland, Conor Nugent et Barry O'Sullivan : Using case-based reasoning in an algorithm portfolio for constraint solving. *In Proceedings of the 19th Irish Conference on Artificial Intelligence and Cognitive Science*, janvier 2008.

[22] John R. Rice : The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

[23] N. Robertson et P.D. Seymour : Graph minors II : Algorithmic aspects of tree-width. *Algorithms*, 7:309–322, 1986.

[24] Christian Schulte, Guido Tack et Mikael Z. Lagerkvist : Modeling. *In* Christian Schulte, Guido Tack et Mikael Z. Lagerkvist, éditeurs : *Modeling and Programming with Gecode*. 2018. Corresponds to Gecode 6.0.0.

[25] Jendrik Seipp, Manuel Braun, Johannes Garimort et Malte Helmert : Learning portfolios of automatically tuned planners. *In ICAPS*, 2012.

[26] Yang Wang, Jin-Kao Hao, Fred Glover et Zhipeng Lü : Solving the minimum sum coloring problem via binary quadratic programming. *Optimization*, 2012.

[27] Lin Xu, Frank Hutter, Holger H. Hoos et Kevin Leyton-Brown : SATzilla : Portfolio-based algorithm selection for SAT. *J. Artif. Intell. Res. (JAIR)*, 32:565–606, 2008.