HAL
open science

# Méthodes hybrides parallèles pour la résolution de problèmes d'optimisation combinatoire : application au clustering sous contraintes

Abdelkader Ouali

▶ **To cite this version:**

Abdelkader Ouali. Méthodes hybrides parallèles pour la résolution de problèmes d'optimisation combinatoire : application au clustering sous contraintes. Analyse numérique [cs.NA]. Normandie Université; Université d'Oran, 2017. Français. NNT : 2017NORMC215 . tel-01609160

**HAL Id: tel-01609160**

**https://theses.hal.science/tel-01609160**

Submitted on 3 Oct 2017

**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.

Normandie Université

# THÈSE

**Pour obtenir le diplôme de doctorat**

**Spécialité Informatique**

**Préparée au sein de l'Université de Caen Normandie**

**En partenariat international avec l'Université d'Oran 1 Ahmed Ben Bella, Algérie**

## Méthodes hybrides parallèles pour la résolution de problèmes d'optimisation combinatoire : application au clustering sous contraintes

**Présentée et soutenue par**
**Abdelkader OUALI**

| Thèse soutenue publiquement le 03 Juillet 2017 devant le jury composé de | | |
|---|---|---|
| Mme. Christine SOLNON | Professeur / INSA de Lyon | Rapporteur |
| Mme. Christel VRAIN | Professeur / Université d'Orléans | Rapporteur |
| M. Simon DE GIVRY | CR1, HdR / Chargé de recherche INRA / MIA de Toulouse | Examinateur |
| M. Amédéo NAPOLI | DR / Directeur de Recherche CNRS / LORIA Nancy | Examinateur |
| M. Lakhdar LOUKIL | MC A / Université d'Oran 1 | Examinateur |
| M. Patrice BOIZUMAULT | Professeur / Université de Caen Normandie | Directeur de thèse |
| M. Samir LOUDNI | MCF HdR / Université de Caen Normandie | Examinateur |
| M. Yahia LEBBAH | Professeur / Université d'Oran 1 | Examinateur |

**Thèse dirigée par Patrice BOIZUMAULT, laboratoire GREYC.**

UNIVERSITÉ CAEN NORMANDIE

GREYC

# Remerciements

Cette thèse a été partiellement soutenue par le programme d'excellence EIFFEL du ministère français des affaires étrangères et du développement international, REFERENCE : UR / EIFFEL-DOCTORAT 2015 / 840868H.

Tout d'abord, je tiens à remercier vivement Yahia Lebbah qui m'a fourni de bonnes conditions de travail, ainsi de le partage de ses idées et de son expérience à travers les nombreuses discussions, qui parfois prenaient un temps énorme ! Je lui en suis reconnaissant d'avoir initié la collaboration et la codirection de la thèse entre les deux universités, cette expérience a été très enrichissante pour moi.

Je remercie très chaleureusement Samir Loudni de ses efforts, de son énergie, de sa motivation et de sa disponibilité qui ont fortement contribué à l'aboutissement de ma thèse, c'est un grand plaisir de travailler avec lui.

Je tiens aussi à remercier Patrice Boizumault qui, malgré ses nombreuses occupations, a accepté la direction de la thèse ; vos remarques constructives et vos conseils précieux m'ont beaucoup aidé à améliorer mes travaux. Je remercie aussi Lakhdar Loukil pour sa disponibilité et son soutien tout au long de ma thèse ; je vous remercie pour le temps et la patience que vous m'avez accordé durant ces années.

Je souhaite remercier Albrecht Zimmermann pour l'intérêt qu'il a porté à ma thèse à travers les réunions et les groupes de travail de l'équipe CoDaG, mais aussi pour ses compétences, sa gentillesse et son humilité. Je lui adresse mes sentiments les plus respectueux.

J'adresse mes sincères remerciements à Mme Christine Solnon et à Mme Christel Vrain d'avoir accepté de rapporter cette thèse et de leur intérêt pour mon travail. Je remercie aussi tous les membres du jury pour l'intérêt qu'ils ont porté à mes travaux, avec un remerciement particulier à Simon de Givry pour son analyse soignée de mes travaux et ses suggestions qui m'ont beaucoup appris.

Je remercie tous les membres de l'équipe CoDaG du laboratoire GREYC et tous les membres de l'équipe PCO du laboratoire LITIO pour leur sympathie et leur aide durant ces années.

Je tiens à remercier l'université d'Oran 1 d'avoir mis à la disposition de ses chercheurs l'unité de calcul intensif, ainsi que le Centre de Recherche sur l'Information Scientifique et Technique (Alger) de nous avoir donné accès à la plateforme de Calcul Haute Performance IBNBADIS. Ces deux unités de calcul nous ont permis de réaliser nos expérimentations dans de très bonnes conditions.

Enfin, je remercie tous les membres de ma famille, ma mère, mon père, mon frère Mohammed et mes deux sœurs Rania et Safaa qui ont toujours été présents malgré l'éloignement.

# Table des matières

**Remerciements** **iii**

**Introduction** **xi**

1 Contexte . . . . . xi
1.1 Méthodes hybrides parallèles . . . . . xi
1.2 Extraction des ensembles de motifs . . . . . xii
2 Objectifs . . . . . xii
2.1 Méthodes hybrides parallèles exploitant la structure d'un problème . . . . . xii
2.2 Extraction d'ensembles de motifs . . . . . xii
3 Contributions . . . . . xiii
3.1 Méthodes hybrides parallèles . . . . . xiii
3.2 Extraction des ensembles de motifs à l'aide de la PLNE . . . . . xiv
4 Plan du mémoire . . . . . xv
4.1 État de l'art . . . . . xv
4.2 Contributions . . . . . xvi

## I État de l'art

**1 Outils de modélisation et de résolution classiques** **3**
1.1 Problèmes d'optimisation combinatoire . . . . . 3
1.2 Programmation linéaire . . . . . 4
1.2.1 Définitions . . . . . 4
1.2.2 Principaux algorithmes . . . . . 7
1.2.3 Génération de colonnes . . . . . 8
1.3 Programmation par contraintes . . . . . 9
1.3.1 Formalisme CSP . . . . . 9
1.3.2 Méthodes de recherche arborescente des CSP . . . . . 10
1.3.3 Méthodes de filtrage des CSP . . . . . 12
1.4 Réseaux de fonction de coût . . . . . 13
1.4.1 Formalisme WCSP . . . . . 13
1.4.2 Recherche par séparation et évaluation en profondeur d'abord ( DFBB) . . . . . 15
1.4.3 Filtrage par cohérences locales souples . . . . . 16

1.5 Exploiter la structure du problème . . . 19
1.5.1 Décomposition arborescente . . . 20
1.5.2 Méthodes de décomposition arborescente . . . 21
1.5.3 Raffinement des décompositions arborescentes . . . 22
1.6 Méthodes complètes exploitant la décomposition arborescente pour les WCSP . . . 24
1.6.1 Cluster Tree Elimination CTE . . . 24
1.6.2 Backtrack Bounded By Tree-Decomposition BTD . . . 24
1.6.3 AND/OR tree search . . . 24
1.7 Conclusion . . . 25

**2 Méthodes de recherche à voisinage variable 27**
2.1 Recherche locale . . . 28
2.2 Métaheuristiques . . . 30
2.3 Recherche à voisinage variable . . . 32
2.3.1 Schéma de base . . . 32
2.3.2 Recherche à voisinage variable réduite . . . 33
2.3.3 Recherche à voisinage variable biaisée . . . 34
2.3.4 Recherche à voisinage variable avec décomposition . . . 34
2.3.5 Recherche à voisinage variable primale-duale . . . 35
2.3.6 Recherche à formulation variable . . . 35
2.3.7 Recherche à voisinage large . . . 36
2.4 Extensions de VNS pour la résolution des WCSP . . . 37
2.4.1 VNS/LDS+CP . . . 37
2.4.2 VNS guidée par la décomposition arborescente (DGVNS) . . . 38
2.5 Stratégies parallèles pour VNS . . . 39
2.5.1 Parallélisme . . . 39
2.5.2 Parallélisation de VNS . . . 40
2.5.3 Recherche VNS parallèle synchrone . . . 40
2.5.4 Recherche VNS parallèle répliquée (RPVNS) . . . 40
2.5.5 Recherche VNS parallèle avec phase de shaking répliquée (RSVNS) . . . 40
2.5.6 Recherche VNS parallèle à voisinages coopératifs (CNVNS) . . . 41
2.6 Évaluation des métaheuristiques . . . 41
2.6.1 Temps CPU et taux de réussite . . . 42
2.6.2 Profil de performance . . . 43
2.6.3 Profils de rapport de performance . . . 43
2.7 Conclusion sur les métaheuristiques . . . 44

**3 Motifs ensemblistes et ensembles de motifs 45**
3.1 Théorie d'une base transactionnelle . . . 46
3.1.1 Contexte transactionnel . . . 46
3.1.2 Extraction sous contraintes . . . 47
3.2 Extraction de motifs sous contraintes locales . . . 47

3.2.1 Contraintes de base . . . . . 47
3.2.2 Élagage de l'espace de recherche . . . . . 49
3.2.3 Représentations condensées . . . . . 50
3.2.4 LCM : Extraction des motifs fermés en temps linéaire . . . . . 54
3.3 Extraction d'ensembles de motifs . . . . . 55
3.3.1 Définitions de base . . . . . 55
3.3.2 Contraintes n-aires . . . . . 56
3.4 Problèmes de l'extraction d'ensembles de motifs . . . . . 58
3.4.1 Clustering conceptuel . . . . . 58
3.4.2 Problème du tuilage . . . . . 59
3.4.3 Algorithmes d'extraction . . . . . 60
3.5 Bilan . . . . . 61

**4 Jeu de test** **63**
4.1 Problème d'allocation de fréquences à des liens radio . . . . . 64
4.1.1 Description du problème . . . . . 64
4.1.2 Modélisation sous forme d'un WCSP . . . . . 64
4.1.3 Description des instances . . . . . 64
4.2 Instances générées par GRAPH . . . . . 67
4.3 Problème de planification d'un satellite de prise de vue . . . . . 67
4.3.1 Description du problème . . . . . 67
4.3.2 Modélisation sous forme d'un WCSP . . . . . 68
4.3.3 Présentation des instances . . . . . 68
4.4 Problème de sélection de TagSNP . . . . . 68
4.4.1 Description du problème . . . . . 69
4.4.2 Modélisation sous forme d'un WCSP . . . . . 69
4.4.3 Présentation des instances . . . . . 70
4.5 Bases de données de l'UCI . . . . . 70
4.5.1 Descriptions des bases de données . . . . . 70
4.6 Conclusion . . . . . 71

**II Contributions**

**5 Méthodes parallèles exploitant la décomposition arborescente** **75**
5.1 Exploitation parallèle des clusters de la décomposition arborescente . . . . . 76
5.1.1 Modèle maître-travailleurs . . . . . 77
5.1.2 Modes de communication . . . . . 77
5.2 VNS coopératif parallèle exploitant la décomposition arborescente . . . . . 78
5.2.1 Algorithme du processus maître . . . . . 78
5.2.2 Algorithme du processus travailleur . . . . . 80
5.3 Stratégies de réplication parallèles pour DGVNS . . . . . 80

5.3.1 Réplication parallèle asynchrone de DGVNS . . . 81
5.3.2 Réplication parallèle synchrone de DGVNS . . . 82
5.3.3 Algorithme du processus travailleur . . . 82
5.4 Intensification et diversification . . . 82
5.4.1 Stratégies d'intensification . . . 82
5.4.2 Stratégies de diversification . . . 83
5.5 Résultats expérimentaux . . . 84
5.5.1 Apports de la parallélisation . . . 84
5.6 Bilan . . . 88

**6 Découverte d'ensembles de motifs utilisant la PLNE 91**
6.1 Démarche en deux étapes . . . 92
6.1.1 Aperçu général . . . 92
6.1.2 Catégorisation des contraintes . . . 93
6.1.3 Résolution des contraintes locales . . . 94
6.1.4 PLNE et contrainte n-aires . . . 94
6.2 Reformulation linéaire des contraintes n-aires . . . 95
6.2.1 Contrainte de taille ($C_{taille}^{\theta_{taille},\lesseqgtr}$) . . . 95
6.2.2 Contrainte de couverture ($C_{cov}^{\theta_{freq},\lesseqgtr}$) . . . 96
6.2.3 Contrainte d'aire ($C_{aire}^{\theta_{aire},\lesseqgtr}$) . . . 97
6.2.4 Contrainte de redondance ($C_{redd}^{\theta,\lesseqgtr}$) . . . 98
6.2.5 Contrainte de généralisation ($C_{gen}^{\Psi}$) . . . 99
6.2.6 Contrainte de spécialisation ($C_{spe}^{\Psi}$) . . . 100
6.2.7 Contrainte de représentativité ($C_{rep}^{\theta_{rep},\lesseqgtr,\mathcal{T}_i}$) . . . 101
6.3 Contraintes d'agrégation . . . 101
6.3.1 Contrainte d'agrégation moyenne ($C_{avg}^{\theta_{avg},\lesseqgtr}$) . . . 101
6.3.2 Contrainte d'agrégation somme ($C_{sum}^{\theta_{sum},\lesseqgtr}$) . . . 102
6.3.3 Contrainte d'agrégation maximum et minimum ($C_{max}^{\theta_{max}}, C_{min}^{\theta_{min}}$) . . . 102
6.4 Approximations linéaires des contraintes n-aires difficiles . . . 102
6.5 Conclusion . . . 103

**7 Programmation linéaire pour l'extraction des ensembles de motifs 105**
7.1 Applications de l'extraction d'ensembles de motifs . . . 106
7.1.1 Clustering conceptuel et ses variantes . . . 106
7.1.2 Problème du tuilage . . . 107
7.2 Extraction des ensembles de motifs en PLNE . . . 108
7.2.1 Encodage PLNE des problèmes de clustering . . . 108
7.2.2 Clustering conceptuel dur . . . 108
7.2.3 Co-clustering . . . 109
7.2.4 Soft clustering et soft co-clustering . . . 110
7.2.5 Problème du tuilage . . . 110
7.2.6 Modélisation d'autres types de requêtes . . . 111

7.3 Évaluation des modèles PLNE . . . . . 112
7.3.1 Protocole expérimental . . . . . 112
7.3.2 Passage à l'échelle . . . . . 113
7.3.3 Relaxation du nombre $k$ de clusters . . . . . 115
7.3.4 Qualité des résultats . . . . . 116
7.4 Conclusion . . . . . 121

**8 Conclusion et perspectives** **125**
8.1 Conclusions . . . . . 125
8.1.1 Méthode parallèle coopérative exploitant la décomposition arborescente dans VNS . . 126
8.1.2 Stratégies parallèles répliquées pour DGVNS . . . . . 126
8.1.3 Clustering Conceptuel en PLNE . . . . . 126
8.1.4 PLNE pour l'extraction des ensembles de motifs . . . . . 127
8.2 Perspectives . . . . . 127

**Bibliographie** **129**

# Introduction

## 1 Contexte

Les problèmes d'optimisation combinatoire sont devenus la cible de nombreuses recherches scientifiques pour leur importance dans la résolution de problèmes académiques et de problèmes réels rencontrés dans le domaine de l'ingénierie industrielle. La résolution de ces problèmes par des méthodes exactes telles que les méthodes *branch-and-X (X=bound, price, cut)*, la programmation dynamique, la programmation par contraintes, etc. ne peut être envisagée à cause des délais de traitement souvent exorbitants que nécessiteraient ces méthodes pour atteindre la (les) solution(s) optimale(s). Pour appréhender de tels problèmes, on utilise des méthodes de recherche locale ou des méta-heuristiques hybrides parallèles.

Dans cette thèse, nous nous sommes intéressés au contexte algorithmique de résolution des problèmes combinatoires, et au contexte de modélisation de ces problèmes.

— Au niveau algorithmique, nous avons appréhendé les méthodes hybrides qui excellent par leur capacité à faire coopérer les méthodes exactes et les méthodes approchées afin de produire rapidement des solutions.
— Au niveau modélisation, nous avons travaillé sur la spécification et la résolution exacte de problématiques complexes de l'extraction d'ensembles de motifs.

### 1.1 Méthodes hybrides parallèles

Les méthodes de recherche locale, ou les méta-heuristiques en général, [Talbi, 2009] permettent d'atteindre des solutions approchées avec des délais d'exécution raisonnables comparativement aux méthodes exactes, tout en évitant d'être piégées dans un minimum local. L'hybridation consiste à concevoir des méthodes d'optimisation évoluées et robustes en combinant des méthodes de résolution possédant des caractéristiques complémentaires. Le parallélisme permet d'accélérer le processus d'optimisation et de pouvoir traiter des problèmes de grande taille.

La décomposition arborescente introduite par Robertson et Seymour [Robertson et Seymour, 1986] permet de découper un problème en un ensemble de sous-problèmes (clusters) constituant un graphe acyclique. De nombreuses approches exploitant la décomposition arborescente ont été proposées dans divers domaines : pour vérifier la satisfiabilité (SAT) [Rish et Dechter, 2000], pour résoudre des problèmes en programmation par contraintes [Dechter et Pearl, 1989], dans les réseaux bayésiens ou probabilistes [Marinescu et Dechter, 2009], dans les bases de données relationnelles [Gottlob *et al.*, 2012], pour l'optimisation sous contraintes (`BTD` [Jégou et Terrioux, 2003], `+Lc-BTD` [de Givry *et al.*, 2006], `RDS-BTD` [Sánchez *et al.*, 2009], `DB` [Kitching et Bacchus, 2009]).

Récemment, Fontaine et al. [Fontaine *et al.*, 2013] ont introduit `DGVNS` (Decomposition Guided `VNS`) [1] qui utilise le graphe des clusters fourni par une décomposition arborescente afin de guider l'exploration de grands voisinages vers des solutions plus pertinentes.

1. VNS *(Variable Neighborhood Search)* est une méthode de recherche à voisinage variable introduite dans [Mladenovic et Hansen, 1997].

### 1.2 Extraction des ensembles de motifs

La fouille des motifs locaux est une tâche fondamentale dans le processus de découverte de connaissances. Beaucoup de travaux ont été menés dans ce contexte au cours des dernières décennies. Cependant, le traitement des motifs locaux obtenus est souvent une tâche difficile pour l'expert, en raison de l'absence de structure qui les relie et de leur nombre souvent très conséquent *(pattern flooding)*. Une solution souvent proposée consiste à réduire le nombre de motifs extraits en ajustant les seuils des contraintes sur les motifs locaux. L'inconvénient d'une telle approche est que l'utilisateur risque de perdre des connaissances intéressantes. En effet, les effets de changement des seuils sur la qualité des résultats sont souvent mal maîtrisés par l'utilisateur.

L'extraction d'ensembles de motifs (ou *Pattern Set Mining* [Raedt et Zimmermann, 2007] en anglais) est l'une des tâches importantes d'exploration de données, telles que le clustering et la classification, offrant une solution aux défis rencontrés dans l'extraction des motifs locaux. Cette tâche présente cependant des défis majeurs, tant sur le plan conceptuel qu'algorithmique. La plupart des approches proposées utilisent des stratégies heuristiques pour trouver des solutions en temps raisonnable. D'autres approches traitent le problème dans le cadre de la programmation par contraintes (PPC) [Guns *et al.*, 2013], et exploitent son aspect déclaratif pour modéliser et résoudre différentes tâches d'extraction d'ensembles de motifs.

## 2 Objectifs

Cette thèse comporte deux volets :

— Au niveau algorithmique, la conception et la mise en œuvre des méthodes hybrides parallèles qui exploitent la structure du problème afin de produire rapidement des solutions.
— La modélisation de problématiques complexes de fouille des ensembles de motifs, en étudiant tout particulièrement le passage à l'échelle sur des bases de données de grande taille.

### 2.1 Méthodes hybrides parallèles exploitant la structure d'un problème

Les calculateurs actuels sont naturellement multi-coeurs et offrent d'immenses ressources de calcul aux utilisateurs. L'exploitation de ces ressources pour la mise en œuvre de méthodes d'optimisation hybrides parallèles offre la possibilité de résoudre des problèmes combinatoires complexes de grande taille dans différents domaines d'application avec une qualité de solution souvent meilleure que celle fournie par les méthodes séquentielles. D'autre part, la structure d'un problème n'est généralement pas prise en considération dans le processus de résolution des problèmes d'optimisation sous contraintes. Or, il existe de nombreux problèmes réels fortement structurés dont la décomposition arborescente peut s'avérer bénéfique. Les travaux menés jusqu'à présent exploitent les décompositions arborescentes uniquement dans le cadre des méthodes de recherche séquentielles.

**Le premier objectif** de cette thèse est de proposer de nouvelles méthodes hybrides parallèles qui exploitent la décomposition arborescente du problème afin d'accélérer le processus de résolution Au niveau applicatif, nous considérerons plusieurs instances de problèmes difficiles issuse des jeux tests de la communauté optimisation.

### 2.2 Extraction d'ensembles de motifs

Récemment, la PPC a ouvert de nouvelles perspectives en procurant un cadre déclaratif pour l'extraction de motifs en fouille de données. Plusieurs travaux récents portant sur l'extraction de motifs locaux [Guns *et al.*, 2011] ou le clustering [Métivier *et al.*, 2012, Dao *et al.*, 2013b] ont montré l'intérêt et les apports de la PPC pour modéliser et résoudre de tels problèmes. Suite aux défis algorithmiques posés par ce problème, les approches déclaratives

proposées traitent un cas particulier du problème d'extraction des ensembles de motifs, qui est l'extraction des ensembles de taille $k$, appelés ensembles de $k$ motifs *($k$-pattern sets)* en limitant la taille de l'espace de recherche et le nombre de solutions possibles.

En effet, le point commun de ces travaux est de modéliser les problèmes d'extraction de motifs sous forme d'un problème de satisfaction de contraintes CSP (pour *Constraint Satisfaction Problem*). Une telle modélisation présente l'avantage d'être flexible en permettant d'exprimer diverses contraintes de natures différentes et de les traiter simultanément sans se préoccuper de leur résolution. Le clustering sous contraintes a été aussi traité dans d'autres cadres déclaratifs tels que SAT [Davidson *et al.*, 2010] ou encore la programmation linéaire en nombres entiers (PLNE) [Babaki *et al.*, 2014, Mueller et Kramer, 2010a].

Cependant, toutes ces approches se heurtent au problème de passage à l'échelle quand il s'agit de traiter des ensembles de données de taille de plus en plus grande. De plus, ils peuvent parfois être restrictifs en ne considérant que des tâches spécifiques, ou en exigeant que le nombre de clusters soit fixé à l'avance.

**Le second objectif de cette thèse** est de s'attaquer au problème de passage à l'échelle des approches déclaratives existantes pour le clustering sous contraintes et l'extraction d'ensembles de motifs, afin de traiter des problèmes de (très) grande taille.

# 3 Contributions

## 3.1 Méthodes hybrides parallèles

**Approche parallèle coopérative exploitant la décomposition arborescente dans VNS.** La méthode `DGVNS` (Decomposition Guided `VNS`), proposée dans la thèse de Mathieu Fontaine [Fontaine, 2013], exploite les clusters issus de la décomposition arborescente d'un graphe de contraintes afin de guider l'exploration de voisinages au sein de `VNS`. Bien que `DGVNS` soit efficace sur certaines instances de taille conséquente, elle ne passe pas à l'échelle quand ces dernières sont de grande taille, les performances de `DGVNS` décroissent significativement en raison du nombre important de clusters à considérer. Afin de remédier à ce problème, nous avons proposé une première stratégie de parallélisation de `DGVNS` appelée `CPDGVNS` (Cooperative Parallel `DGVNS`) [Ouali *et al.*, 2014a, Ouali *et al.*, 2014b] qui consiste à explorer, en parallèle, tous les clusters fournis par une décomposition arborescente. `CPDGVNS` repose sur une architecture *maître-travailleurs*, où le processus maître mémorise, met à jour, et communique la meilleure solution courante, alors que les processus travailleurs gèrent, individuellement, l'exploration des clusters. Les processus travailleurs coopèrent par une communication asynchrone en échangeant des informations sur la meilleure solution trouvée. Ce schéma de communication permet, d'une part, de garantir une gestion locale des clusters, et d'autre part, d'initialiser l'exploration de ces clusters avec des solutions variées, favorisant ainsi la diversification de la recherche. Les expérimentations ont été effectuées sur plusieurs instances de trois problèmes réels ; la méthode `CPDGVNS` a montré son efficacité par rapport à la méthode séquentielle.

**Stratégies parallèles répliquées exploitant la décomposition arborescente pour `VNS`.** Dans `CPDGVNS`, les clusters de la décomposition arborescente sont explorés en parallèle par chaque processus travailleur. Chaque processus travailleur effectue une recherche `DGVNS` dans le cluster qui lui est assigné. La principale limite de cette approche est que les travailleurs doivent effectuer un certain nombre d'itérations avant de partager leurs meilleures solutions avec le maître. Cela rend la coopération avec le maître moins fréquente, et limite la diversification de l'exploration de l'espace de recherche par les travailleurs. Pour améliorer la diversification dans `CPDGVNS`, nous avons proposé deux nouvelles stratégies de parallélisation (1) *Replicated Asynchronous* `DGVNS` (`RADGVNS`) et (2) *Replicated Synchronous* `DGVNS` (`RSDGVNS`) [Ouali *et al.*, 2015] permettant la production rapide de solutions intermédiaires pour alimenter l'échange d'informations entre les travailleurs :

— `RADGVNS` : Comme pour `CPDGVNS`, les processus travailleurs coopèrent de manière asynchrone en échangeant des informations sur la meilleure solution courante. Toutefois, et contrairement à `CPDGVNS`, chaque

processus travailleur exécute une seule itération (un seul mouvement), en complétant une solution partielle dans le cluster qui lui est associé.

— `RSDGVNS` : Pour étudier l'impact de la synchronisation, nous avons proposé une seconde version dont la communication entre le processus maître et les processus travailleurs est synchrone. Dans `RSDGVNS`, le processus maître attend la fin de tous les processus travailleurs afin d'évaluer le prochain mouvement.

## 3.2 Extraction des ensembles de motifs à l'aide de la PLNE

Trois critères importants ont guidé nos travaux : le passage à l'échelle notamment en découpant le processus d'extraction en plusieurs étapes, la déclarativité de l'approche proposée ainsi que sa complétude [2].

**Déclarativité.** Les activités en fouille de données peuvent être classifiées en différentes catégories, en fonction des tâches à réaliser et des objectifs visés. La plupart des algorithmes efficaces sont dédiés pour résoudre une tâche particulière et ne peuvent être utilisés pour résoudre d'autres tâches sans devoir développer de nouveaux algorithmes de résolution, entraînant des coûts de conception et de développement très élevés. La déclarativité est un aspect important pour l'extraction des motifs locaux et les ensembles de motifs. En effet, elle permet d'exprimer des propriétés différentes sans pour autant changer d'algorithme de résolution. Ainsi, vouloir traiter une variété de contraintes complexifie la résolution puisque les contraintes peuvent ne pas vérifier les mêmes propriétés mathématiques (e.g. monotonie/anti-monotonie). Trouver un algorithme générique offrant un compromis entre efficacité et déclarativité est un axe de recherche important.

**Passage à l'échelle.** L'amélioration des technologies de collecte et de stockage des informations au fil des dernières décennies a permis de mémoriser d'énormes quantités de données dans différents domaines d'applications, notamment dans la communauté scientifique et médicale, l'administration publique, ainsi que dans le monde des affaires. La plupart des algorithmes liés à l'analyse de ces grandes bases de données, souvent pour extraire des connaissances utiles, sont inutilisables vu la complexité croissante et la diversité de l'information de ces bases de données. L'une des méthodes conventionnelles en fouille de données, notamment pour contourner ces problèmes d'efficacité, est le découpage du processus d'extraction en plusieurs étapes successives. Dans une telle démarche, chaque étape de résolution sera traitée par des algorithmes dédiés et efficaces, donnant ainsi des informations pertinentes en temps raisonnable pour les étapes suivantes. Cette séparation permet, dans certains cas, de "casser" la complexité du problème en le divisant en différents sous-problèmes qui seront résolus plus efficacement. Cette démarche pourrait permettre le passage à l'échelle par rapport à une méthode utilisant une et une seule étape de résolution.

**Preuve d'optimalité.** En raison des problèmes liés au passage à l'échelle, très peu de tentatives basées sur des approches complètes existent pour l'extraction des ensembles de motifs (voir section 3.3). Ceci peut s'expliquer par la taille importante de l'espace de recherche et le manque de techniques d'élagage efficaces pour ce type de problèmes. Extraire le meilleur ensemble de motifs relativement à une mesure donnée permet de mieux cibler le processus d'extraction vers les meilleurs motifs mais rend la tâche encore plus ardue. La plupart des approches existantes (souvent heuristiques) sacrifient la preuve d'optimalité au détriment de solutions approchées. Toutefois, la qualité de solutions obtenue par ces approches reste très variable. Il est donc nécessaire de disposer de méthodes qui offrent la garantie de trouver une solution optimale et d'en prouver l'optimalité.

Nous proposons **une approche hybride combinant à la fois la PLNE et la fouille de concepts** [Ouali *et al.*, 2016, Ouali *et al.*, 2017]. Notre approche est complète et tire profit du cadre général de la PLNE (en procurant un haut niveau de flexibilité et d'expressivité) et des heuristiques spécialisées pour l'exploration et l'extraction de données (pour améliorer les temps de calcul). Outre le cadre général de l'extraction des ensembles de motifs, nous avons étudié plus particulièrement deux problèmes : **le clustering conceptuel et le problème de tuilage** *(tiling)*. Les expérimentations menées ont montré l'apport de notre proposition par rapport aux approches à base de contraintes et aux heuristiques spécialisées.

2. Calcul de la solution optimale et preuve de son optimalité

# 4 Plan du mémoire

Le mémoire est classiquement organisé en deux parties : la première se focalise sur l'état de l'art et la seconde présente nos contributions.

Toutefois, *le lecteur plus particulièrement intéressé par les méthodes hybrides parallèles* pourra lire successivement les chapitres 1 et 2 (état de l'art), puis le chapitre 4 (jeux de données utilisés) et enfin le chapitre 5 pour les contributions. De manière analogue, *le lecteur plus intéressé par l'extraction d'ensembles de motifs en PLNE* pourra lire successivement les chapitres 1 et 3 (état de l'art), puis le chapitre 4 (jeux de données utilisés) et enfin les chapitres 6 et 7 pour les contributions.

## 4.1 État de l'art

Dans la première partie, nous présentons et définissons les différents concepts et terminologies associés à chaque contexte, nous dressons un panorama des différentes approches et méthodes existantes pourla résolution les problèmes considérés.

**Le chapitre 1** présente les modèles et les méthodes de résolution, tels que la programmation linéaire, les réseaux de fonctions de coût et les méthodes de décomposition arborescente. Nous présentons d'abord le formalisme utilisé pour décrire les différents problèmes rencontrés dans cette thèse. Nous résumons les principales notions liées à la programmation linéaire, la PLNE, et la génération de colonnes. Nous introduisons ensuite les réseaux de fonctions de coût (ou WCSP) en rappelant les notions de base liées à la PPC et à son extension au cas pondéré. Enfin, nous dressons pour chaque cadre les méthodes de résolution courantes les plus connues. Enfin, nous explorons les notions de la décomposition arborescente basées sur la triangulation afin de les exploiter sur les WCSP.

**Le chapitre 2** décrit les méthodes de recherche à voisinage variable. Nous introduisons les notions de base liées aux méthodes de recherche locale, les métaheuristiques, et le calcul parallèle. Nous nous focalisons ensuite sur la recherche à voisinage variable (VNS), nous citerons quelques variantes de VNS et présenterons quelques schémas de parallélisation de cette métaheuristique. Enfin, nous présentons différentes mesures pour évaluer nos contributions par rapport aux méthodes séquentielles.

**Le chapitre 3** s'intéresse à l'extraction des motifs locaux sous contraintes et à l'extraction des ensembles de motifs. Tout d'abord, nous présentons les différents concepts inhérents à la fouille de motifs, et nous explorons les différents types de motifs proposés dans la littérature. Puis, nous rappelons, les propriétés intéressantes des contraintes permettant le développement d'algorithmes dédiés très efficaces pour l'extraction des motifs locaux. Nous détaillons aussi, dans ce chapitre, les notions de base liées à l'extraction des ensembles de motifs et ses applications. Enfin, nous dressons un état de l'art sur les approches proposées pour le problème du clustering conceptuel et le problème de tuilage.

**Le chapitre 4** présente les différents problèmes (RLFAP, GRAPH, SPOT5 et TagSNP, ainsi que les bases de transactions *(datasets)* de l'UCI [3]) utilisées dans nos expérimentations. Afin d'évaluer précisément nos contributions sur les méthodes parallèles exploitant la décomposition arborescente, nous reprenons les mêmes instances et les mêmes décompositions que celles utilisées dans la thèse de Mathieu Fontaine [Fontaine, 2013]. Pour l'extraction des ensembles de motifs, nous évaluons notre approche PLNE sur les mêmes bases de transactions que celles utilisées dans [Guns *et al.*, 2013].

3. Dépôt d'apprentissage automatique en ligne, `https://archive.ics.uci.edu/ml/datasets.html`

## 4.2 Contributions

**Le chapitre 5** commence par décrire notre première stratégie parallèle, CPDGVNS (Cooperative Parallel DGVNS), où la recherche repose sur une exploration locale des clusters, qui est gérée individuellement par les différents processus travailleurs. Les différents processus coopèrent par un échange asynchrone des meilleures solutions trouvées par chacun. Nous décrivons ensuite RSDGVNS et RADGVNS (Replicated Synchronous and Asynchronous DGVNS). Les deux stratégies utilisent des processus travailleurs qui effectuent une itération de perturbation sur la solution (c'est-à-dire que $k$ variables choisies aléatoirement de la solution sont détruites, puis reconstruites d'une meilleure façon), rendant ainsi les deux méthodes plus coopératives que CPDGVNS. Enfin, nous étudions et discutons l'influence des différentes méthodes de décomposition arborescente sur nos stratégies parallèles. Nous montrons l'intérêt de fusionner les clusters qui se chevauchent fortement, afin de construire des structures de voisinage plus pertinentes qui améliorent la diversification des stratégies proposées.

Les expérimentations réalisées sur des instances aléatoires (GRAPH) et sur diverses instances difficiles des problèmes présentés dans le chapitre 4, montrent l'apport de l'approche parallèle par rapport à la version séquentielle. Les résultats montrent l'importance de retirer les clusters de petite taille pour mieux raffiner la décomposition arborescente et améliorer les performances de RADGVNS et de RSDGVNS.

**Le chapitre 6** décrit notre approche générique pour l'extraction des ensembles de motifs. Notre approche procède en deux étapes :

— une étape de pré-traitement qui permet d'extraire des représentations condensées sur une base de données, qui seront des candidats pour former les ensembles de motifs ;
— une étape de résolution qui utilise les techniques de la PLNE pour former le meilleur ensemble des motifs relativement à une mesure donnée.

Nous définissons et modélisons un nombre important de contraintes et de mesures avec des expressions linéaires pour définir des propriétés sur l'ensemble des motifs recherchés. Ces contraintes et mesures seront utilisées pour formuler différentes requêtes. Nous distinguons les contraintes locales sur les motifs des contraintes n-aires sur les ensembles de motifs, qui seront résolues en deux étapes principales. Chacune des étapes fait intervenir des algorithmes de résolution dédiés et efficaces.

**Le chapitre 7** est consacré à l'étude de deux problèmes : le problème du clustering conceptuel et le problème du tuilage. Nous définissons les concepts liés à chacun des problèmes, et citons quelques variantes des deux problèmes. Nous détaillons ensuite les modèles PLNE utilisés pour chaque problème et pour chacune des variantes. Nous terminons le chapitre par une évaluation et une discussion des résultats obtenus par notre approche et par les approches existantes sur la base des trois aspects évoqués précédemment : déclarativité, passage à l'échelle et qualité des résultats.

Enfin, nous concluons en dressant, **au chapitre 8**, un bilan des travaux menés durant cette thèse, puis nous présentons et discutons différentes perspectives.

# Première partie

# État de l’art

# Chapitre 1

# Outils de modélisation et de résolution classiques

## Sommaire


**1.1 Problèmes d'optimisation combinatoire . . . 3**
**1.2 Programmation linéaire . . . 4**
1.2.1 Définitions . . . 4
1.2.2 Principaux algorithmes . . . 7
1.2.3 Génération de colonnes . . . 8
**1.3 Programmation par contraintes . . . 9**
1.3.1 Formalisme CSP . . . 9
1.3.2 Méthodes de recherche arborescente des CSP . . . 10
1.3.3 Méthodes de filtrage des CSP . . . 12
**1.4 Réseaux de fonction de coût . . . 13**
1.4.1 Formalisme WCSP . . . 13
1.4.2 Recherche par séparation et évaluation en profondeur d'abord ( DFBB) . . . 15
1.4.3 Filtrage par cohérences locales souples . . . 16
**1.5 Exploiter la structure du problème . . . 19**
1.5.1 Décomposition arborescente . . . 20
1.5.2 Méthodes de décomposition arborescente . . . 21
1.5.3 Raffinement des décompositions arborescentes . . . 22
**1.6 Méthodes complètes exploitant la décomposition arborescente pour les WCSP . . . 24**
1.6.1 Cluster Tree Elimination CTE . . . 24
1.6.2 Backtrack Bounded By Tree-Decomposition BTD . . . 24
1.6.3 AND/OR tree search . . . 24
**1.7 Conclusion . . . 25**


## 1.1 Problèmes d'optimisation combinatoire

Dans cette section, nous présentons des notions de base sur les problèmes d'optimisation combinatoire. Une description plus détaillée peut être trouvée dans [Papadimitriou et Steiglitz, 1982, Nemhauser et Wolsey, 1988]. Dans cette thèse, nous nous intéressons aux problèmes d'optimisation et aux problèmes de satisfaction de contraintes.

Nous nous concentrons sur les problèmes d'optimisation, sachant qu'un problème de satisfaction peut se ramener à un problème d'optimisation.

**Définition 1.1. PROBLÈME COMBINATOIRE**
Un problème combinatoire est défini par un couple $(S, C)$ avec une requête $R$, où $S$ est l'ensemble des solutions candidates, $C$ est un ensemble fini de propriétés à satisfaire, et $R$ une requête déterminant l'objectif de la résolution.

L'ensemble $S$ représente les différentes configurations envisageables du problème, ici appelées solutions candidates. L'ensemble $C$ permet de définir, de façon implicite ou explicite, l'ensemble des propriétés que doit satisfaire une solution candidate, une telle solution sera appelée solution *faisable*[4]. La requête $R$ peut prendre plusieurs formes permettant de définir plusieurs catégories de problèmes. Par exemple, une requête pourrait être de type "existe-il une solution faisable ?", la réponse à cette requête est soit *oui*, soit *non*, on parle alors d'un problème de *décision*. Un second type de requête concerne la recherche de solutions d'un problème satisfaisant toutes les propriétés de $C$. On parle, dans ce cas, de problème de *satisfaction*. Enfin, une requête peut porter sur la recherche d'une solution qui optimise (minimise ou maximise) une certaine mesure, la réponse est une solution faisable optimale relativement à la mesure considérée. Ce type de problème est appelé problème d'*optimisation*. D'autres formes de requêtes peuvent être énoncées, notamment des requêtes associées aux problèmes *multi-objectifs*.

**Définition 1.2. PROBLÈME D'OPTIMISATION COMBINATOIRE**
Un problème d'optimisation combinatoire est défini par un quadruplet $(S, C, \xi, f)$ et une requête $R$, où $S$ est l'ensemble des solutions candidates, $C$ est un ensemble fini de propriétés à satisfaire, $\xi$ est un ensemble ordonné, $f : S \to \xi$ est la fonction objectif qui associe à chaque élément de $S$ un élément de $\xi$ et $R$ une requête déterminant l'objectif de la résolution (minimisation ou maximisation). $f(s)$ est appelée le coût (ou le gain) de la solution $s$.

La définition précédente permet de caractériser un problème de satisfaction au sein du problème d'optimisation. Résoudre un problème d'optimisation combinatoire revient à trouver la meilleure solution de l'ensemble $S$ au sens du critère $f$, cette solution sera appelée solution optimale.

**Définition 1.3. SOLUTION OPTIMALE**
Une solution optimale $s^* \in S$ est une solution faisable telle que : $\forall s \in S, f(s^*) \leq$ (resp. $\geq$) $f(s)$ si $R$ est une requête de minimisation (resp. maximisation).

## 1.2 Programmation linéaire

### 1.2.1 Définitions

La programmation linéaire (PL) est un outil très puissant en recherche opérationnelle qui repose sur des techniques de résolution et de modélisation développées depuis des décennies afin de résoudre des problèmes d'optimisation dans différents domaines d'application. Dans cette section, nous présentons les différentes définitions de base et les terminologies liées à la programmation linéaire.

#### 1.2.1a Programme linéaire

Un programme linéaire est défini par une fonction objectif linéaire sur des variables mathématiques dites variables de décisions. Ces variables sont soumises à un ensemble de contraintes exprimées sous forme d'équations et/ou d'inéquations linéaires. La requête dans un programme linéaire consiste à trouver une solution minimisant (resp. maximisant) la fonction objectif. Un programme linéaire peut être formulé comme suit :

4. Appelée aussi solution réalisable ou solution admissible.

$$(P)\left\{\begin{array}{lll} \texttt{optimiser} & (1) & z = \sum_{j=1}^{n} c_j \,.\, x_j \\ \texttt{s.c.} & (2) & \sum_{j=1}^{n} A_{i,j} \,.\, x_j \ \{\leq, =, \geq\}\ b_i, \forall i \in \{1, \ldots, m\} \\ & (3) & x_j \geq 0, \forall j \in \{1, \ldots, n\} \end{array}\right.$$

où $(P)$ est un programme linéaire de $n$ variables de décision et de $m$ contraintes, avec :

- $(1)$ représente la fonction objectif à optimiser[5],
- $(2)$ représente l'ensemble des $m$ contraintes (équations et/ou inéquations) linéaires que doivent satisfaire les variables de décision $x_j$, $j \in \{1, \ldots, n\}$, et
- $(3)$ représente les contraintes d'intégralité, également appelées contraintes implicites, sur les variables de décision $x_j$.

Notons que ces variables peuvent prendre une valeur réelle quelconque en remplaçant toute variable $x_j$ par la différence $(x_j^+ - x_j^-)$. Les contraintes d'intégralité classifient les programmes linéaires en trois catégories :

1. programme linéaire classique dont les variables sont toutes définies dans $\mathbb{R}$,
2. programme linéaire dont les variables sont toutes définies dans $\mathbb{Z}$, on parle alors d'un programme linéaire en *nombres entiers*,
3. programme linéaire dont seulement une partie des variables est définie dans $\mathbb{Z}$, dans ce cas, il est dit programme linéaire *mixte*.

#### 1.2.1b Formes d'un programme linéaire

Un programme linéaire où il s'agit de minimiser la fonction objectif et dont toutes les contraintes sont des inéquations de type $(\geq)$ et toutes les variables de décision sont non-négatives est dit sous forme *canonique*. Un programme linéaire dont toutes les contraintes sont des équations et toutes les variables de décision sont non-négatives est dit sous forme *standard*.

Tout programme linéaire peut être ramené à sa forme canonique ou standard moyennant des règles de transformation simples. Par exemple, $Max(z)$ est équivalent à $Min(-z)$. Une inéquation de type $(\leq)$ est transformée en une inéquation de type $(\geq)$ en multipliant les deux cotés de l'inéquation par $(-1)$. Enfin, une équation peut être remplacée par deux inéquations ($\leq$ et $\geq$), et une inéquation peut être transformée en équation par l'ajout des *variables d'écart*.

#### 1.2.1c Solutions d'un programme linéaire

Considérons un programme linéaire sous forme standard noté $(P_0)$ :

$$(P_0)\left\{\begin{array}{ll} \texttt{min}\ z = & \sum_{j=1}^{n} c_j \,.\, x_j \\ \texttt{s.c.} & \sum_{j=1}^{n} A_{i,j} \,.\, x_j = b_i, \forall i \in \{1, \ldots, m\} \\ & x_j \geq 0, \forall j = \{1, \ldots, n\} \end{array}\right.$$

Soit $A = (A_{i,j})$ la matrice des coefficients des contraintes de $(P_0)$. Soit $B$ une sous-matrice carrée de $A$ de taille $(m \times m)$ appelée une *base*, et $N$ une sous-matrice de $A$ telle que $A = (B|N)$. Cette équation peut être réécrite sous la forme :

$$A \,.\, x = b \Leftrightarrow (B|N) \,.\, \begin{pmatrix} x_B \\ x_N \end{pmatrix} = b$$

5. Minimiser ou maximiser.

où, $x_B$ sont les variables de base associées à la matrice $B$ de $(P_0)$, et $x_N$ sont les variables hors base associées à la matrice $N$ de $(P_0)$. En multipliant le système d'équations par $B^{-1}$ (on suppose que $B$ est inversible), nous obtenons :

$$B^{-1}\,.\,(B|N)\,.\,\begin{pmatrix} x_B \\ x_N \end{pmatrix} = B^{-1}\,.\,b \Leftrightarrow x_B + B^{-1}\,.\,N\,.\,x_N = B^{-1}\,.\,b$$

Nous pouvons remarquer que l'ensemble des solutions de l'équation $(A\,.\,x = b)$ peut être déterminé en fixant arbitrairement les valeurs de $x_N$ :

$$\begin{cases} x_N & = & v, \quad v \in \mathbb{R}^{n-m} \\ x_B & = & B^{-1}\,.\,(b - N\,.\,v) \end{cases}$$

Une *solution de base*, associée à la matrice $B$, est une solution où toutes les variables hors base $x_N$ associées à la matrice $N$ sont nulles ($x_B = B^{-1}\,.\,b$ et $x_N = 0$). Une solution de base est dite *dégénérée* si le vecteur $x_B$ a une variable nulle. L'ensemble des solutions faisables $\{x \mid A\,.\,x = b,\ x \geq 0\}$ de $(P_0)$ est appelé *polyèdre de solutions* dont les sommets correspondent aux solutions faisables.

#### 1.2.1d Dualité

Tout programme linéaire possède une autre version dite programme linéaire *dual*. Un programme linéaire dual est obtenu à partir d'une série de transformations sur le programme linéaire original, appelé *primal*.

$$\begin{cases} \texttt{min} & z = c^T x \\ \texttt{s.c.} & Ax \geq b \\ & x \geq 0. \end{cases} \quad \rightarrow \quad \begin{cases} \texttt{max} & w = b^T y \\ \texttt{s.c.} & A^T y \leq c \\ & y \geq 0. \end{cases}$$

$$\begin{cases} \texttt{min} & z = c^T x \\ \texttt{s.c.} & Ax = b \\ & x \geq 0. \end{cases} \quad \rightarrow \quad \begin{cases} \texttt{max} & w = b^T y \\ \texttt{s.c.} & A^T y \leq c \\ & y \ \texttt{non-restreintes}. \end{cases}$$

Un problème de minimisation dans le primal devient un problème de maximisation dans le dual. A toute contrainte du PL primal est associée une variable de décision $y$ dans le PL dual. Si le primal est consistant avec la forme canonique [6] (cf. section 1.2.1b), alors les variables de décision $y$ du dual sont non-négatives. S'il est l'inverse de la forme canonique, alors les variables $y$ sont des variables non-positives. Enfin, s'il est consistant avec la forme standard, alors les variables $y$ sont non-restreintes. La matrice $A^T$ des coefficients des contraintes du dual est la transposée de la matrice $A$ des coefficients des contraintes du primal. Les coefficients $c^T$ de la fonction objectif du primal deviennent le second membre des contraintes du dual, et le second membre $b$ des contraintes du primal deviennent les coefficients de la fonction objectif dans le dual. Les théorèmes de dualité ci-dessous donnent les liens entre les solutions du primal et du dual (pour plus de détails, voir [Griva *et al.*, 2009]) :

- *Dualité faible* : soit $x$ une solution faisable dans un PL primal sous forme standard et soit $y$ une solution faisable dans son PL dual, alors $z = c^T x \geq b^T y = w$.
- *Dualité forte* : considérons un programme linéaire primal et son dual. Si l'un des problèmes possède une solution optimale, alors son dual possède également une solution optimale, et les valeurs des fonctions objectif sont égales.
- *Écart complémentaire* : considérons un programme linéaire primal sous forme standard et son dual. Si $x$ est optimale pour le primal et $y$ est optimale pour le dual, alors $x^T(c - A^T y) = 0$. Si $x$ est faisable pour le primal, $y$ est faisable pour le dual, et $x^T(c - A^T y) = 0$, alors $x$ et $y$ sont optimales pour leurs problèmes respectifs.

6. Ayant le même type de contraintes, et le même sens de la fonction objectif.

### 1.2.2 Principaux algorithmes

#### 1.2.2a Programmation linéaire

La méthode du *simplexe* est la méthode la plus utilisée pour résoudre des programmes linéaires. Elle a été développée en 1947 par G. Dantzig en même temps que les modèles de programmation linéaire utilisés pour la planification économique et militaire. La méthode du simplexe est une méthode exacte et itérative. Elle consiste à démarrer d'une solution de base initiale réalisable (pas nécessairement optimale) et à l'améliorer itérativement par une solution adjacente qui améliore la fonction objectif, jusqu'à obtenir la solution optimale. La méthode du simplexe a été en mesure de s'adapter pour traiter des problèmes de plus en plus complexes et de tirer profit de la puissance des machines de calcul, devenant ainsi, la méthode de choix pour de nombreuses personnes. Cependant, Klee et Minty, dans [Klee et Minty, 1972], ont mis en évidence des exemples pour lesquels le simplexe nécessite un nombre d'opérations qui croit exponentiellement avec la taille du problème. Cette étude a ouvert la voie pour la recherche d'autres algorithmes polynomiaux pour résoudre les programmes linéaires. Dans [Khachiyan, 1979], un premier algorithme polynomial, appelé *ellipsoïde*, a été proposé pour résoudre des programmes linéaires. L'algorithme utilise une suite d'ellipsoïdes de volume décroissant et qui contiennent à chaque itération la solution optimale. Une illustration est donnée dans [Goldfarb et Todd, 1989]. Toutefois, l'algorithme ellipsoïde se heurte à des problèmes d'efficacité face au simplexe sur des problèmes réels autres que ceux de Klee et Minty. Un deuxième algorithme polynomial est proposé par Karmarkar [Karmarkar, 1984], appelé méthode de *points intérieurs*. L'idée est de chercher des points à l'intérieur du polyèdre des solutions, afin de converger plus rapidement vers le sommet optimal. Des dérivés de cette méthode concurrencent le simplexe sur certains problèmes de grande taille, en particulier la méthode de *barrière* [Goldfarb et Todd, 1989]. Une description plus détaillée du simplexe se trouve dans [Griva *et al.*, 2009].

#### 1.2.2b Programmation linéaire en nombres entiers

La résolution d'un programme linéaire en nombres entiers (PLNE) est difficile du fait que l'espace de recherche n'est plus *convexe*. Ainsi, trouver la solution optimale revient à énumérer chaque solution de l'espace de recherche. La PLNE fournit des outils de résolution et de modélisation très utiles pour de nombreuses approches de résolution. Cependant, les méthodes classiques de la programmation linéaire ne peuvent pas être utilisées directement pour résoudre un PLNE, puisqu'un sommet du polyèdre n'a pas forcément des coordonnées entières, comme illustré par la figure 1.1.

$$\begin{cases} \texttt{max} & z = 12x_1 + 7x_2 \\ \texttt{s.c.} & 6x_1 + 4x_2 \leq 13 \\ & x_1, x_2 \in \mathbb{Z}^+ \end{cases}$$

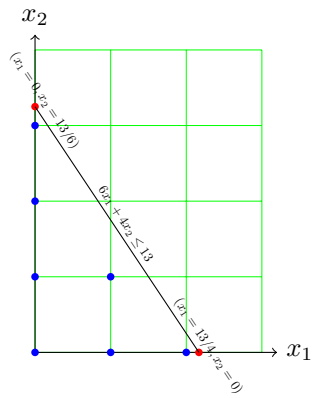



FIGURE 1.1 – Un PLNE dont les 2 sommets du polyèdre (points rouges) n'ont pas de coordonnées entières. Les points bleus représentent les solutions entières faisables du PLNE.

Il faut noter que pour certains problèmes, comme les problèmes de flot ou les problèmes dont la matrice du PLNE est totalement unimodulaire [7], la solution optimale de la relaxation linéaire du PLNE est une solution entière, ainsi les algorithmes classiques de la programmation linéaire peuvent être utilisés pour les résoudre. En général, la résolution de la relaxation s'avère très intéressante, puisqu'elle fournit une borne inférieure (resp. une borne supérieure) pour un problème de minimisation (resp. un problème de maximisation).

Différentes méthodes et stratégies ont été proposées pour la recherche des solutions entières optimales. Ce sont principalement des méthodes de recherche arborescente. La méthode la plus connue est la méthode *branch-and-bound* (méthode par *séparation* et *évaluation*, en français). La méthode branch-and-bound divise (sépare) l'espace de recherche en plusieurs sous-problèmes, évalue la borne inférieure (cas de minimisation) de la solution optimale du sous-problème en résolvant la relaxation linéaire (e.g. simplexe), et la compare au coût de la solution réalisable courante du problème, appelée référence. Ce processus est réitéré pour les sous-problèmes dont la borne inférieure est inférieure à la référence, c-à-d. les sous-problèmes susceptibles de contenir une meilleure solution. Les méthodes de coupes ont pour objectif de réduire la taille du polyèdre convexe qui enveloppe les solutions entières du PLNE. En effet, la résolution de la relaxation linéaire du PLNE réduit à cet ensemble donne une solution optimale entière. Les coupes sont réalisées en ajoutant à chaque itération des contraintes qui réduisent le polyèdre de solutions sans éliminer les solutions entières. Une description plus détaillée des différentes techniques standards pour la résolution des PLNE est donnée dans [Wolsey, 1998]. Les auteurs dans [Marchand *et al.*, 2002] explorent particulièrement les différentes méthodes de coupes, où les coupes de Gomory [Gomory, 1958] sont les plus connues. Les auteurs dans [Gamrath *et al.*, 2015] présentent les dernières techniques de prétraitement permettant de réduire l'espace de recherche avant l'étape de résolution.

### 1.2.3 Génération de colonnes

Soit à résoudre le programme linéaire de $n$ variables et $m$ contraintes suivant :

$$(P_1)\begin{cases} \texttt{min}\ z = & \sum_{j\in J} c_j \,.\, x_j \\ \texttt{s.c.} & \sum_{j\in J} A_{i,j} \,.\, x_j \geq b_i, \forall i \in \{1,\ldots,m\} \\ & x_j \geq 0, \forall j \in \{1,\ldots,n\}. \end{cases}$$

Soit $J$ l'ensemble des indices des $n$ variables de décision du programme linéaire $(P_1)$. $(P_1)$ est appelé problème maître.

Dans de nombreuses applications, le nombre $n$ de variables est largement supérieur au nombre $m$ de contraintes ($n \gg m$). L'utilisation de la méthode du simplexe dans ce cas n'est pas envisageable en raison du nombre d'itérations important qu'elle nécessite pour atteindre la solution optimale. Pour cela, nous considérons un problème maître restreint (PMR) contenant un sous-ensemble $J' \subseteq J$ de variables. Une solution optimale $x^*$ du (PMR) n'est pas forcément optimale pour le problème maître. Notons $y^*$ la solution duale optimale du (PMR).

L'étape du simplexe consiste à faire entrer une variable hors-base ayant un coût réduit négatif [8] dans l'ensemble des variables de base. La variable entrante correspond à une contrainte duale violée par la solution duale $c_j - y^* a_j < 0$. Afin d'accomplir cela avec la génération de colonnes, nous considérons un autre problème, appelé sous-problème, défini comme suit :

$$(SP) : min\{c_j - y^* a_j | j \in J\}$$

Si (SP) fournit un coût négatif, alors la variable $x_j$ correspondant au coût réduit $\{c_j - y^* a_j\}$ minimal, son coefficient $c_j$ dans la fonction objectif et son vecteur de coefficients $(a_{i,j})$ seront ajoutés aux variables du (PMR).

7. Une matrice unimodulaire est une matrice carrée à coefficients entiers dont le déterminant vaut +1 ou –1. Une matrice totalement unimodulaire est une matrice (non nécessairement carrée) à coefficients entiers dont chaque sous-matrice carrée de déterminant non nul est unimodulaire.

8. Pour un problème de minimisation.

Ce processus est réitéré jusqu'à ne plus trouver de variable améliorante (toutes les variables de (SP) ont un coût réduit positif). Dans ce cas, la solution optimale $x^*$ est aussi optimale pour le problème maître $(P_1)$.

L'apport de la génération de colonnes vient du fait que le sous-problème (ensemble de variables $J$) peut être résolu efficacement avec un autre algorithme. Par conséquent, au lieu d'énumérer explicitement toutes les variables hors base pour trouver la variable entrante, nous pouvons résoudre un problème typique bien structuré avec une énumération implicite, rendant ainsi le processus de résolution plus efficace que la méthode du simplexe.

**Bornes duales**
La résolution de la relaxation du (PMR) par le simplexe donne accès à la solution duale, par conséquent, nous pouvons calculer la borne duale du problème maître et arrêter la résolution quand on atteint une qualité désirée. Notons par $v(PMR)$ la solution optimale du (PMR). Si $\sum_{j \in J} x_j \leq \kappa$ pour une solution optimale du problème maître $v(PM)$, alors nous ne pouvons pas améliorer $v(PMR)$ de plus de $\kappa$ fois le coût réduit le plus petit $v(SP)$ :

$$v(PMR) + \kappa \,.\, v(SP) \leq v(PM).$$

Si une contrainte de convexité est présente dans le modèle $\kappa = 1$, cette borne est très serrée puisque $v(SP) = 0$ lorsque la génération de colonnes se termine. Il faut noter que si le sous-problème est résolu par une méthode heuristique, la solution $v(SP)$ ne sera plus disponible pour calculer la borne. D'autres propositions existent dans [Farley, 1990] et [de Carvalho, 2002].

## 1.3 Programmation par contraintes

La programmation par contraintes (PPC) [Rossi *et al.*, 2006] est un paradigme de programmation déclaratif offrant un formalisme, appelé CSP (*Constraint Satisfaction Problem*), afin de modéliser différents problèmes combinatoires en intelligence artificielle et en recherche opérationnelle résolus en faisant appel à un ensemble de méthodes génériques. Le formalisme CSP peut être vu comme une extension du modèle `SAT` afin de décrire des propriétés que doit satisfaire une solution, sous forme de *variables* et de *contraintes* plus générales. Cette représentation du problème permet de séparer l'étape de résolution de la modélisation, donnant lieu à des méthodes de résolution génériques.

Nous introduisons, dans cette section, les notions de base ainsi que les méthodes de recherche arborescente associées au cadre CSP. Nous nous focalisons en particulier sur la cohérence et les mécanismes de filtrage.

### 1.3.1 Formalisme CSP

Un problème de satisfaction de contraintes s'appuie sur la définition d'un réseau de contraintes fini.

**Définition 1.4. Réseau de contraintes fini**
Un réseau de contraintes fini est défini par un triplet $(X, D, C)$, où
- $X$ est un ensemble fini de variables ;
- $D$ est un ensemble de domaines finis $D_i$ associé à chaque variable $x_i$ de $X$ ;
- $C$ est un ensemble de contraintes, chaque contrainte est définie sur un sous-ensemble de variables $X_c \subseteq X$ et une relation $R_c$ entre les variables $X_c$ : $R_c \subset \Pi_{x_i \in X_c} D_i$.

Un réseau de contraintes permet d'exprimer l'ensemble des solutions faisables $S$ d'un problème combinatoire. Une solution faisable est appelée une *affectation complète*.

**Définition 1.5. Affectation**
Une affectation des variables est l'application qui associe à chaque variable $x_i \in X$ une valeur dans son domaine $a_i \in D_i$, notée par $(x_i, a_i)$. Soit $\mathcal{A}$ l'ensemble des variables affectées.

— $\mathcal{A}$ est une affectation partielle si $\mathcal{A} \subset X$.
— $\mathcal{A}$ est une affectation complète si $\mathcal{A} = X$.
— $\mathcal{A}[X']$ est la projection des valeurs de $\mathcal{A}$ sur le sous-ensemble de variables $X' \subseteq X$.
— $\mathcal{A}'$ est une extension de l'affectation $\mathcal{A}$ si $\mathcal{A} \subseteq \mathcal{A}'$ et si toutes les variables affectées par $\mathcal{A}$ ont les mêmes valeurs dans $\mathcal{A}'$. Si $\mathcal{A}'$ est une affectation complète, alors elle est appelée une extension complète de $\mathcal{A}$.

Dans de nombreuses applications, le nombre de variables peut être très grand, potentiellement, avec des domaines de grande taille. Par conséquent, le nombre d'affectations complètes devient rapidement très important. En effet, un réseau de contraintes de $n$ variables, où chaque variable a un domaine de $d$ valeurs, possède $d^n$ affectations complètes. Néanmoins, seule une partie des affectations complètes peut être acceptée. La faisabilité des solutions est vérifiée par des relations entre les valeurs que peuvent prendre les variables du problème, appelées contraintes.

**Définition 1.6. CONTRAINTE**
Soit $c = (X_c, R_c)$ une contrainte, où $X_c$ est un sous-ensemble de variables de l'ensemble $X$, appelé la portée de la contrainte $c$, et $R_c$ est une relation sur les variables de $X_c$ déterminant les affectations autorisées des affectations interdites.
— On appelle arité d'une contrainte le cardinal de l'ensemble $X_c$.
— La dureté de la contrainte $c$ est définie par le ratio entre le nombre d'affectations interdites et le nombre d'affectations possibles.
— Le taux de satisfiabilité est défini par le ratio entre le nombre d'affectations autorisées et le nombre d'affectations possibles.
— Une contrainte $c$ est complètement affectée par l'affectation $\mathcal{A}$ si $X_c \subseteq \mathcal{A}$.
— Une contrainte $c$ est satisfaite par l'affectation $\mathcal{A}$ si elle est complètement affectée par $\mathcal{A}$ et si les valeurs associées par $A(X_c)$ satisfont la relation $R_c$.

Le problème de satisfaction de contraintes est un problème de décision construit avec un réseau de contraintes dont l'objectif est de chercher une affectation complète qui satisfait toutes les contraintes du problème.

**Définition 1.7. CSP**
Un problème de satisfaction de contraintes est défini par :
— Un réseau de contraintes $(X, D, C)$,
— Une requête $R$ spécifiant l'objectif de la résolution.

### 1.3.2 Méthodes de recherche arborescente des CSP

La résolution d'un CSP est un problème *NP-complet* dans le cas général. Cependant, de nombreuses méthodes ont été développées ces deux dernières décennies afin d'améliorer l'efficacité des méthodes de résolution sur des problèmes réels de grande taille. Parmi ces méthodes, nous distinguons les méthodes complètes dont l'espace de recherche est organisé sous forme d'un arbre : à chaque *nœud* de l'arbre correspond une variable (ou point de choix) et à chaque *branche* une des différentes affectations possibles des variables. Chaque feuille de l'arbre donne une affectation complète des variables du problème qui est considérée comme une solution, les affectations qui ne satisfont pas les contraintes, le long de l'arbre, sont considérées comme un échec.

Une des principales caractéristiques des méthodes de recherche arborescente pour la résolution des CSP est leur complétude assurant un parcours complet de l'espace des solutions. Ces méthodes sont au cœur des systèmes de programmation par contraintes.

#### 1.3.2a Algorithme BackTrack (BT)

L'algorithme de base pour résoudre les CSP est l'algorithme du BackTrack chronologique (BT) [Golomb et Baumert, 1965]. C'est une procédure d'énumération récursive en profondeur d'abord. Il commence par une affec-

tation vide, et à chaque itération, une valeur d'une variable courante est choisie. Le seul contrôle effectué consiste à vérifier que l'affectation résultante est satisfaite par l'ensemble des contraintes. Dans l'affirmative, il continue avec cette nouvelle affectation partielle en ajoutant une nouvelle variable non encore affectée. Sinon, il essaie une autre valeur de la variable courante. S'il n'y a plus de valeur inexplorée, il se trouve dans une voie sans issue, par conséquent, il désinstancie la variable courante (il effectue un backtrack). Dans le pire des cas, l'arbre de recherche est exploré en entier avec une complexité temporelle de l'ordre de $O(d^n)$. Cet algorithme peut être amélioré en pratique par l'ajout d'un filtrage des domaines (en prétraitement ou au cours de la recherche après chaque affectation de variable) [Sabin et Freuder, 1994, Bessière et Régin, 1996, Debruyne et Bessière, 2001]. Ce filtrage consiste à supprimer les valeurs qui ne peuvent pas participer à une solution du CSP (cf. section 1.3.3).

Une autre piste d'amélioration concerne le retour-arrière lui-même. Alors que le Backtrack chronologique revient, en cas d'échec, sur le dernier choix effectué, d'autres mécanismes analysent les causes de l'échec soit pour choisir un meilleur point de retour (*Backjumping* [Gaschnig, 1979], *Graph-Based Backjumping* [Dechter, 1990], *Conflict-Directed Backjumping* [Prosser, 1993]), soit pour identifier les instanciations partielles qui ne participent à aucune solution et ainsi éviter des explorations inutiles (*Local Change* [Schiex et Verfaillie, 1994], *Dynamic Backtracking* [Ginsberg, 1993] et `mac-dbt` [Jussien *et al.*, 2000]). Une description détaillée de ces algorithmes est donnée dans [Dechter, 2003].

#### 1.3.2b Heuristiques sur l'ordre de choix des variables et des valeurs

Ordonner les variables et les valeurs avant ou pendant la résolution d'un CSP est connu comme une autre voie importante d'amélioration pour l'algorithme de BT. On distingue généralement deux types d'ordre : *statique* dont les choix sont effectués avant la recherche et sont figés ; *dynamique* dont les choix dépendent du sous-arbre à explorer (par exemple, nombre de variables affectées, taille des domaines, etc.) et sont effectués à chaque nœud au cours de la recherche. Le choix de variable permet de réduire la taille de l'espace de recherche, généralement en provoquant le plus rapidement possible un échec. De nombreuses heuristiques de choix de variable ont été proposées, on distingue notamment l'heuristique *domaine minimum* (dom) [Haralick et Elliott, 1980] favorisant les variables avec les domaines les plus petits, l'heuristique *degré maximum* [Dechter et Meiri, 1989] favorise les variables dont les degrés sont les plus élevés, l'heuristique *domaine/degré* (dom/deg) [Bessière et Régin, 1996] propose de tirer parti des informations des deux heuristiques précédentes en favorisant les variables dont le ratio de la taille de leur domaine par rapport à leur degré est le plus faible, et l'heuristique *domaine/degré futur* (dom/fdeg) [Smith et Grant, 1998], qui est une variante de dom/deg, où le degré futur d'une variable est utilisé à la place du degré. Le degré futur d'une variable correspond au nombre de contraintes portant sur une variable et dont au moins une variable n'est pas affectée. D'autre part, les heuristiques de choix de valeur cherchent à aiguiller la recherche vers les branches les plus prometteuses de l'arbre de recherche. Parmi ces heuristiques, on retrouve l'heuristique lexicographique (lex) dont les valeurs sont simplement ordonnées selon l'ordre de leur domaine, l'heuristique *basée « look-ahead »* [Frost et Dechter, 1995] qui classe les valeurs selon l'ordre croissant du nombre de retraits provoqués par l'affectation de celle-ci, l'heuristique *basée sur les explications* [Cambazard, 2006] qui utilise les informations liées au retrait d'une valeur (c'est-à-dire aux affectations justifiant ce retrait), et l'heuristique *basée sur les impacts* [Refalo, 2004] qui ordonne les valeurs par ordre croissant d'impact. L'impact associé à une valeur mesure l'influence de son affectation sur la taille de l'espace de recherche.

#### 1.3.2c Recherche arborescente rendue incomplète

Les délais que nécessiteraient les algorithmes complets pour atteindre les solutions sont souvent exorbitants. L'idée consiste à diriger la recherche dans des espaces de l'arbre jugés prometteurs. C'est le principe des recherches guidées par une heuristique de choix de valeur qui produisent un parcours limitant les divergences par rapport aux choix proposés par l'heuristique. Citons la recherche à divergence limitée (LDS) [Harvey et Ginsberg, 1995] qui repose sur l'existence d'une heuristique de choix de valeur $h$ permettant de guider la recherche vers des feuilles de l'arbre les plus prometteuses. Le mot divergence désigne les nœuds intermédiaires de l'arbre de recherche dans

lesquelles l'heuristique de choix de valeur n'a pas été respectée. Le principe de LDS est de suivre l'heuristique de choix $h$, en autorisant un petit écart $\delta$ aux choix de l'heuristique, qui peut proposer de meilleurs choix quand l'heuristique $h$ échoue. La version complète de l'algorithme consiste à faire une recherche avec au plus $\delta_{max}$ divergences par rapport au choix de l'heuristique $h$, puis, si aucune solution n'est trouvée, on recommence la recherche en incrémentant la valeur de $\delta_{max}$. La figure 1.2 illustre le nombre de divergences accumulées $\delta_{max} = 3$ afin d'explorer tout l'arbre de recherche.

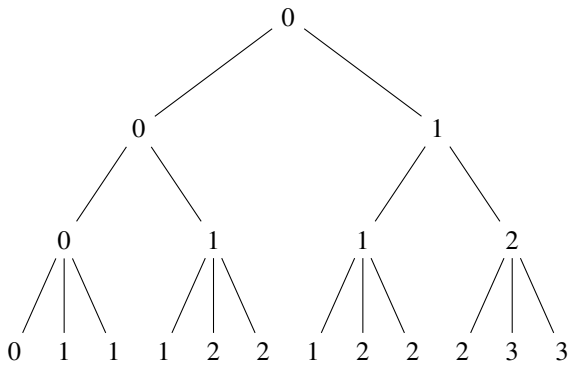



FIGURE 1.2 – Nombre de divergences accumulées $\delta_{max}$ pour explorer tout l'arbre de recherche

Une amélioration de LDS a été proposée dans [Korf, 1996], appelée *Improved LDS* ( ILDS), afin d'éviter la redondance en terme de génération de nœuds. La méthode permet d'éviter de se diriger vers les feuilles de profondeur $n$ dont la divergence est inférieure à celle de l'itération courante.

### 1.3.3 Méthodes de filtrage des CSP

Le principal défi des méthodes arborescentes est de réduire la taille de l'arbre de recherche exploré en filtrant les domaines des variables au fur et à mesure, c-à-d en retirant les valeurs (ou combinaisons de valeurs) qui ne peuvent mener à des affectations non-satisfaites par le réseau de contraintes. Plusieurs niveaux de cohérence et mécanismes de filtrage ont été proposés permettant de maintenir un certain niveau de cohérence souhaité à chaque nœud de l'arbre de recherche.

Pour un réseau de contraintes binaires, le filtrage le plus souvent utilisé est la *cohérence d'arc*. Il consiste à considérer chaque contrainte séparément et à supprimer les valeurs qui ne peuvent pas satisfaire la contrainte (i.e. qui n'appartiennent à aucune combinaison autorisée par la contrainte).

**Définition 1.8. COHÉRENCE D'ARC D'UN RÉSEAU DE CONTRAINTES N-AIRES**
Une contrainte est arc-cohérente s'il existe pour chaque affectation d'une de ses variables, une affectation des autres variables (appelée support) qui satisfait la contrainte. Un CSP n-aire est arc-cohérent (arc-consistent AC) si toutes ses contraintes sont arc-cohérentes.

De nombreux algorithmes de renforcement d'arc-cohérence dans le cadre des contraintes binaires ont été proposés. AC-3 est l'une des premières méthodes proposées [Mackworth, 1977]. Sa complexité en temps est en $O(e \cdot d^3)$, où $e$ est le nombre de contraintes et $d$ la taille du plus grand domaine. Plusieurs améliorations de cet algorithme ont été proposées. On peut notamment citer :

— AC-4 [Mohr et Henderson, 1986] qui est la première méthode permettant de rendre arc-cohérent un réseau de contraintes en temps optimal dans le pire des cas, c'est-à-dire en $O(e \cdot d^2)$. Cette méthode propose de calculer dans un premier temps les supports de toutes les valeurs en $O(e \cdot d^2)$ et ensuite d'effectuer le filtrage en $O(e \cdot d^2)$.

— AC-6 [Bessière et Cordier, 1993] qui permet de diminuer la complexité en moyenne en introduisant une nouvelle notion : celle de support unique. En effet, au lieu de calculer l'ensemble de tous les supports comme AC-4, AC-6 s'assure seulement qu'il en existe au moins un. Et à l'inverse d'AC-3, lorsqu'un support est retiré AC-6 ne recherche pas à nouveau un support à partir de la première valeur du domaine mais depuis celle venant d'être retirée.
— AC2000 [Bessière et Régin, 2001] qui utilise une structure de données additionnelle permettant de stocker le dernier support trouvé. Lors du prochain test de cohérence, on testera en premier si ce support est toujours présent dans le domaine et si tel est le cas, aucune recherche de support n'est effectuée. Cet algorithme a une complexité temporelle et spatiale polynomiale (respectivement en $O(e \cdot d^2)$ et $O(e \cdot d)$).

Une extension de l'arc-cohérence pour les réseaux de contraintes n-aires, appelée *arc-cohérence généralisée* (GAC) ou encore *cohérence globale*, est également proposée dans [Bessière et Régin, 1997].

## 1.4 Réseaux de fonction de coût

Comme présenté dans la section précédente, les CSP permettent d'exprimer des contraintes au sens strict (contraintes dures) entre des variables afin de modéliser les propriétés d'une solution d'un problème donné, qui rend ce cadre trop restrictif pour exprimer, par exemple, des préférences entre solutions. Par ailleurs, les algorithmes associés au formalisme CSP sont incapables de fournir des solutions dans le cas d'un problème sur-contraint. En effet, une solution est requise même si cette dernière ne satisfait pas certaines contraintes considérées comme peu importantes. Par conséquent, différents formalismes ont été proposés afin d'étendre le cadre CSP en modélisant des préférences sur les tuples des contraintes, ou encore des coûts de violation (notion de contrainte souple). Parmi ces extensions, nous pouvons citer [Fargier et Lang, 1993] permettant de modéliser les problèmes ayant des données incomplètes, [Rosenfeld *et al.*, 1976, Schiex, 2013, Ruttkay, 1994] permettant de définir des niveaux de préférences sur les tuples des contraintes, ou encore [Shapiro et Haralick, 1981, Freuder et Wallace, 1992] permettant de traiter les problèmes sur-contraints.

Le cadre général VCSP (*Valued Constraint Satisfaction Problem*) [Schiex *et al.*, 1995] englobe les différentes extensions citées précédemment. Nous nous intéressons, en particulier, à une instance de ce modèle, appelée réseaux de fonctions de coût (*Cost Function Network* (CFN)) ou Weighted CSP (WCSP) [Larrosa, 2002].

Nous introduisons, tout d'abord, le formalisme WCSP. Puis, nous présentons les méthodes de recherche arborescente des WCSP ainsi que les techniques de filtrage associées. Enfin, nous présentons les notions liées à la décomposition arborescente ainsi que les méthodes exploitant la décomposition arborescente pour les WCSP.

### 1.4.1 Formalisme WCSP

Un WCSP est un cas particulier du VCSP où les contraintes valuées sont des fonctions de coût représentant le poids de violation de chaque contrainte. Ces fonctions de coût sont définies à l'aide d'une structure de valuation qui associe à chaque contrainte une valeur désignant le poids de violation de cette contrainte.

**Définition 1.9.** STRUCTURE DE VALUATION
Une structure de valuation, notée $\mathcal{S}$, est définie par un triplet $(\xi, \succ, \oplus)$ où :
— $\xi$ est l'ensemble des valuations associées aux tuples des contraintes, muni :
  — d'un élément minimum $\perp$ correspondant à une insatisfaction maximale,
  — d'un élément maximum $\top$ correspondant à une satisfaction maximale.
— $\succ$ est un ordre total permettant de comparer deux valuations de l'ensemble $\xi$.
— $\oplus$ est un opérateur d'agrégation $\xi \times \xi \rightarrow \xi$, permettant d'agréger ces valuations et respecte les propriétés suivantes :

— commutativité et associativité.
— monotonicité relativement à l'ordre $\succ$ : $\forall a, b, c \in \xi, (a \succ b) \Rightarrow ((a \oplus c) \succ (b \oplus c))$.
— $\perp$ est l'identité : $\forall a \in \xi,\ (a \oplus \perp) = a$.
— $\top$ est l'élément absorbant : $\forall a \in \xi,\ (a \oplus \top) = \top$.

**Définition 1.10. STRUCTURE DE VALUATION D'UN WCSP**
Une structure de valuation d'un WCSP est définie par un triplet $\mathcal{S} = (E^+, \leq, \oplus)$, où $E^+ = [0, 1, \ldots, \top]$ avec $\top \in \mathbb{N} - \{0\}$ représentant la valuation maximale associée à un tuple, l'opérateur $\oplus$ est une somme bornée dans $\mathbb{N}$ telle que :

$$\forall a, b \in E^+,\ a \oplus b = \min\{a + b, \top\}.$$

**Définition 1.11. FONCTION DE COÛT**
Une fonction de coût $w_{X_i}$ est une fonction définie sur un ensemble de variables $X_i \subset X$ qui associe à chaque affectation des variables de $X_i$ un coût dans l'ensemble $E^+$ :

$$w_{X_i} : (\prod_{x_k \in X_i} D_k) \rightarrow E^+.$$

On notera également par $D^{X_i}$ l'ensemble des combinaisons de valeurs possibles des variables de $X_i$. On appelle arité de $w_{X_i}$ le cardinal de l'ensemble de variables $X_i$ dans la portée de $w_{X_i}$. La fonction de coût $w_\emptyset$ d'arité nulle est une constante qui peut être ajoutée à toute affectation et qui sert d'un minorant pour l'algorithme DFBB (voir la section 1.4.2).

**Définition 1.12. WCSP**
Un WCSP est défini par un quadruplet $(X, D, W, \mathcal{S})$ où :
— $X = \{x_1, \ldots, x_n\}$ est l'ensemble des variables du problème,
— $D = \{D_1, \ldots, D_n\}$ est l'ensemble des domaines finis associés aux variables du problème,
— $W = \{w_{X_1}, \ldots, w_{X_e}\}$ est l'ensemble des fonctions de coût,
— $\mathcal{S}$ est une structure de valuation.

Un WCSP est dit binaire si toutes les fonctions de coût $w_{X_i}$ sont des fonctions de coût binaires (i.e. $|X_i| = 2$).

**Définition 1.13. VALUATION D'UNE AFFECTATION COMPLÈTE**
Soit $\mathcal{A}_c$ une affectation complète, la valuation de $\mathcal{A}_c$ est l'agrégation par l'opérateur $\oplus$ des valuations associées aux contraintes non satisfaites par l'affectation $\mathcal{A}_c$. Soit $\widehat{C} \subseteq C$ l'ensemble des contraintes non satisfaites par $\mathcal{A}_c$, la valuation $\varphi(\mathcal{A}_c)$ de l'affectation $\mathcal{A}_c$ est définie par :

$$\varphi(\mathcal{A}_c) = \underset{c \in \widehat{C}}{\oplus} \varphi(c).$$

**Définition 1.14. VALUATION LOCALE D'UNE AFFECTATION PARTIELLE**
Soit $\mathcal{A}_p$ une affectation partielle, la valuation de $\mathcal{A}_p$ est l'agrégation par l'opérateur $\oplus$ des valuations des contraintes complètement affectées et non satisfaites par $\mathcal{A}_p$. Soit $\widetilde{C} \subseteq C$ l'ensemble des contraintes complètement affectées et non satisfaites par $\mathcal{A}_p$, la valuation $\varphi(\mathcal{A}_p)$ de l'affectation $\mathcal{A}_p$ est définie par :

$$\varphi(\mathcal{A}_p) = \underset{c \in \widetilde{C}}{\oplus} \varphi(c).$$

**Définition 1.15. VALUATION GLOBALE D'UNE AFFECTATION PARTIELLE**
Étant donné une affectation partielle $\mathcal{A}_p$, la valuation minimale des extensions complètes est appelée une valuation

globale, notée par $\varphi_g(\mathcal{A}_p)$. Notons $ext_c(\mathcal{A}_p)$ l'ensemble des extensions complètes de $\mathcal{A}_p$, une valuation globale de l'affectation partielle $\mathcal{A}_p$ est définie par :

$$\varphi_g(\mathcal{A}_p) = \bigoplus_{\mathcal{A}' \in ext_c(\mathcal{A}_p)} \varphi(\mathcal{A}').$$

**Définition 1.16. COÛT D'UNE AFFECTATION DANS UN WCSP**
Soit $\mathcal{A}$ une affectation complète, le coût d'une affectation complète est donné par :

$$f(\mathcal{A}) = w_\emptyset \bigoplus_{i=1}^{e} w_{X_i}(\mathcal{A}[X_i]).$$

### 1.4.2 Recherche par séparation et évaluation en profondeur d'abord ( DFBB)

L'objectif dans la résolution d'un WCSP est de trouver une solution optimale minimisant la somme des fonctions de coût. Les algorithmes exacts pour les WCSP les plus efficaces correspondent à des approches arborescentes de type séparation et évaluation en profondeur d'abord. L'algorithme DFBB pour *Depth First Branch and Bound*, qui est une recherche complète en profondeur d'abord utilisée dans différentes approches [Larrosa et Schiex, 2003a, Larrosa et Schiex, 2004, de Givry *et al.*, 2005a], exploite deux bornes pour parcourir implicitement l'arbre de recherche, la première est une borne supérieure $\mathcal{UB}$, appelée aussi un majorant, qui correspond au coût de la meilleure solution trouvée, et qui est mise à jour durant la recherche. La deuxième borne est une borne inférieure $\mathcal{LB}$, appelée aussi un minorant, obtenue par évaluation de chaque nœud de l'arbre de recherche, elle désigne le coût minimal de n'importe quelle solution du sous-problème associé au nœud.

**Algorithme 1 :** DFBB($Y, \mathcal{A}, \mathcal{UB}$)

```
1  LB ← f(A);
2  si LB < UB alors
3      si A est une affectation complète alors
4          retourner LB;
5      Choisir une variable x_i ∈ Y;
6      pour a_j ∈ D_i faire
7          c ← DFBB(Y\x_i, A ∪ (x_i ← a_j), UB);
8          si c < UB alors
9              UB ← c;
10 retourner UB;
```

L'algorithme 1 illustre le pseudo-code de DFBB. Soit $Y \subseteq X$ un sous-ensemble de variables, $\mathcal{A}$ une affectation courante, $\mathcal{UB}$ la borne supérieure courante. Au début, DFBB est appelé avec $(X, \emptyset, \top)$, il calcule ensuite la borne inférieure $\mathcal{LB}$ de la solution partielle courante $\mathcal{A}$ avec $f(\mathcal{A})$, si la borne $\mathcal{LB}$ est inférieure à $\mathcal{UB}$, DFBB vérifie si $\mathcal{A}$ est une affectation complète, auquel cas, $\mathcal{LB}$ est retournée. Dans le cas contraire, une variable $x_i \in Y$ est sélectionnée, puis, pour chaque valeur $a_j \in D_i$, DFBB est appelé récursivement afin de parcourir les affectations possibles des autres variables. Si durant la recherche $\mathcal{LB} \geq \mathcal{UB}$, la borne supérieure $\mathcal{UB}$ est retournée, elle correspond au plus petit coût trouvé sur les extensions de $\mathcal{A}$.

**Algorithme 2 :** Algorithme de projection.

1 procédure projection $(w_{X_i}, x_k, a_j, \alpha)$ ;
2 **début**
4 **pour tous les** $\mathcal{A} \in D^{X_i}$ *tels que* $\mathcal{A}[x_k] = a_j$ **faire**
5 $w_{X_i}(\mathcal{A}) \leftarrow w_{X_i}(\mathcal{A}) \ominus \alpha$;
7 $w_k(a_j) \leftarrow w_k(a_j) \oplus \alpha$ ;

### 1.4.3 Filtrage par cohérences locales souples

Afin de profiter des travaux menés dans le cadre CSP, plusieurs travaux ont adapté des algorithmes dédiés au formalisme CSP pour le formalisme WCSP, notamment pour déduire des affectations non satisfaites par l'ensemble des contraintes et fournir à la recherche un minorant le plus précis possible. Parmi ces travaux, on peut citer par exemple la cohérence de nœud ( NC) [Larrosa, 2002], la cohérence d'arc souple (AC) [Schiex, 2000]), la cohérence d'arc directionnelle ( DAC) [Cooper, 2003, Larrosa et Schiex, 2003b], la cohérence d'arc directionnelle complète ( FDAC) [Cooper, 2003, Larrosa et Schiex, 2003b], la cohérence d'arc existentielle et directionnelle ( EDAC) [de Givry *et al.*, 2005b] et l'arc cohérence virtuelle ( VAC) [Cooper *et al.*, 2008].

**Algorithme 3 :** Algorithme d'extension.

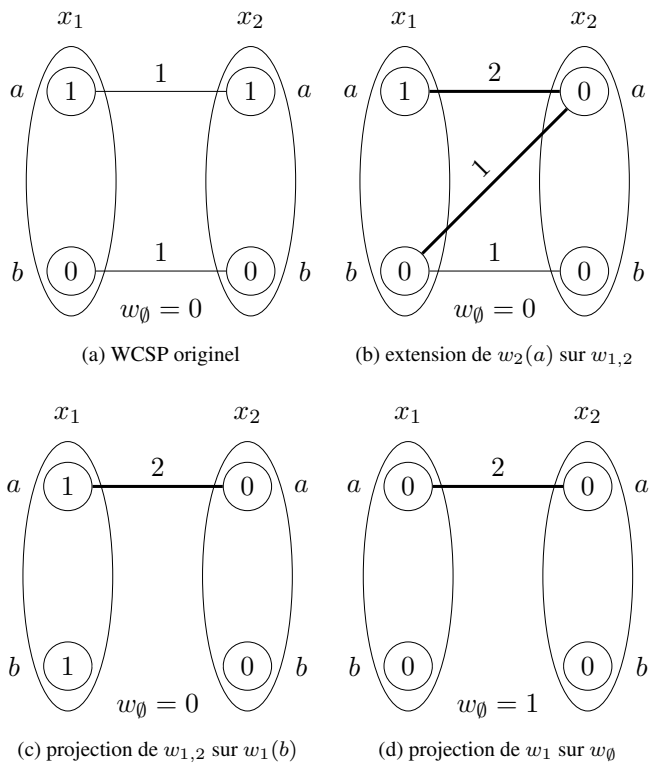
1 procédure extension $(w_{X_i}, x_k, a_j, \alpha)$ ;
2 **début**
4 **pour tous les** $\mathcal{A} \in D^{X_i}$ *tels que* $\mathcal{A}[x_k] = a_j$ **faire**
5 $w_{X_i}(\mathcal{A}) \leftarrow w_{X_i}(\mathcal{A}) \oplus \alpha$;
7 $w_k(a_j) \leftarrow w_k(a_j) \ominus \alpha$ ;

Les algorithmes évoqués ci-dessus utilisent des opérations de transfert de coûts, appelées transformations préservant l'équivalence (EPTs). L'idée principale est de rechercher des *reformulations* en temps polynomial d'un WCSP de manière à faire apparaître explicitement un bon minorant de l'optimum du problème traité. Ces reformulations transforment le réseau en un réseau équivalent, c'est à dire qu'elles ne changent pas le coût des affectations complètes. Elles agissent en déplaçant des coûts entre les fonctions, avec pour objectif de transférer les coûts vers la fonction d'arité nulle $w_\emptyset$.

Nous supposons par la suite l'existence d'une fonction de coût unaire, notée par $w_k$, portant sur chaque variable $x_k$ du problème. Les EPTs classiques utilisées dans les algorithmes de filtrage par cohérences locales souples sont les suivantes :

— opération de *projection* (cf. Algorithme 2) qui déplace un coût $\alpha$ d'une fonction $w_{X_i}$ vers une fonction unaire $w_k(a_j)$, avec $x_k \in X_i$ et $a_j \in D_k$.
— opération *d'extension* (cf. Algorithme 3), qui est l'opération inverse de la projection, et vise à déplacer un coût d'une fonction unaire vers une autre fonction de coût.
— opération de *projection unaire* qui déplace un coût $\alpha$ d'une fonction unaire $w_k$ vers $w_\emptyset$.

**Exemple 1.1.**
Soit le réseau à deux variables $x_1$ et $x_2$ à domaine $\{a, b\}$ de la figure 1.3a ayant 3 fonctions de coût : deux fonctions de coûts unaires $w_1$ et $w_2$ portant respectivement sur $x_1$ et $x_2$ et une fonction de coût binaire $w_{1,2}$. Les coûts unaires non nuls sont représentés dans des cercles et les coûts binaires non-nuls sur des arêtes. L'extension de la fonction de coût unaire de $x_2$ vers $w_{1,2}$ permet de déplacer le coût lié à $(x_2, a)$ sur $w_{1,2}$ (figure 1.3b). Puis, la projection sur $w_1(b)$ de $w_{1,2}$ sur la figure 1.3c, permet d'incrémenter $w_1(b)$ de 1. Enfin, le minorant peut être augmenté de 1 en déplaçant le coût minimal de $w_1$ vers $w_\emptyset$ (figure 1.3d).




(a) WCSP originel (b) extension de $w_2(a)$ sur $w_{1,2}$

(c) projection de $w_{1,2}$ sur $w_1(b)$ (d) projection de $w_1$ sur $w_\emptyset$

FIGURE 1.3 – Opérations d'extension et de projection.

#### 1.4.3a Cohérences pour les WCSP

En se basant sur les opérations de projection et d'extension citées précédemment, différentes notions de cohérence ont été proposées, nous nous focalisons sur les notions qui ont un lien avec nos travaux.

**Définition 1.17. COHÉRENCE DE NŒUD (NC, NODE CONSISTENCY) [LARROSA, 2002]**
Une variable $x_i$ est NC si :

1. $\forall\, a \in D_i, w_i(a) \oplus w_\emptyset < \top$,
2. $\exists\, a \in D_i$ tel que $w_i(a) = 0$.

Un WCSP est nœud cohérent si toutes les variables sont nœud cohérentes.

Dans le cas où $w_i(a) \oplus w_\emptyset = \top$, la valeur $a$ est impossible, et donc elle peut être éliminée du domaine de $D_i$. Tout WCSP $(\mathcal{P})$ peut être rendu nœud cohérent en projetant itérativement les valuations minimales des fonctions de coût unaires sur $w_\emptyset$. Puis, toute valeur qui n'est pas nœud cohérente est filtrée. Enfin, si aucun domaine n'est vide, $(\mathcal{P})$ est alors NC. L'établissement de la nœud cohérence a une complexité temporelle de $O(nd)$.

La cohérence d'arc ( AC, Arc Consistency) dans les CSP est plus forte que la cohérence de nœud car elle fait intervenir le voisinage des variables. La cohérence d'arc souple (SAC, Soft Arc Consistency) est une extension d' AC pour les WCSP.

**Définition 1.18. VOISINAGE D'UNE VARIABLE**
Le voisinage d'une variable $x_i$, noté $V(x_i)$, est défini par l'ensemble des variables $x_j$ telles qu'il existe une fonction de coût portant sur $x_i$ et $x_j$.

**Définition 1.19. COHÉRENCE D'ARC SOUPLE (SAC, SOFT ARC CONSISTENCY) [SCHIEX, 2000]**
Une variable $x_i$ est arc-cohérente souple si :

1. $x_i$ est NC, et
2. $\forall\, x_j \in V(x_i), \forall\, a \in D_i, \exists\, b \in D_j$ tel que $w_{i,j}(a,b) = 0$,
3. $\forall\, x_j \in V(x_i), \forall\, a \in D_i, \forall\, b \in D_j$, $w_{i,j}(a,b) = \top$ si $w_{i,j}(a,b) \geq \top \ominus (w_i(a) \oplus w_j(b) \oplus w_\emptyset)$.

Un WCSP est arc-cohérent souple si toutes les variables sont arc-cohérentes souples.

**Définition 1.20. COHÉRENCE D'ARC TOTALE (FAC, FULL ARC CONSISTENCY)**
Une variable $x_i$ est totalement arc-cohérente si :

1. $x_i$ est NC, et
2. $\forall\, x_j \in V(x_i), \forall\, a \in D_i, \exists\, b \in D_j$ telle que $w_{i,j}(a,b) \oplus w_j(b) = 0$,
3. $\forall\, x_j \in V(x_i), \forall\, a \in D_i, \forall\, b \in D_j$, $w_{i,j}(a,b) = \top$ si $w_{i,j}(a,b) \geq \top \ominus (w_i(a) \oplus w_j(b) \oplus w_\emptyset)$.

Étant donné que la plupart des problèmes n'ont pas de fermeture par FAC, cette cohérence n'est pas utilisée en pratique. Nous allons cependant présenter quelques techniques qui ont réutilisé partiellement FAC avec succès. Nous considérons un ordre arbitraire sur l'affectation des variables du problème tel que la variable $x_i$ est affectée avant $x_j$ si, et seulement si, $i < j$, noté par $x_i \prec x_j$.

**Définition 1.21. VOISINAGE INFÉRIEUR ET SUPÉRIEUR D'UNE VARIABLE**
Le voisinage inférieur (resp. supérieur) d'une variable $x_i$ est l'ensemble : $V^-(x_i) = V(x_i) \cap \{x_j \in X \mid x_j \prec x_i\}$ (resp. $V^+(x_i) = V(x_i) \cap \{x_j \in X \mid x_i \prec x_j\}$).

**Définition 1.22. COHÉRENCE D'ARC DIRECTIONNELLE (DAC, DIRECTIONAL ARC CONSISTENCY)**
[Cooper, 2003, Larrosa et Schiex, 2003b] Une variable $x_i$ est DAC si :

1. $x_i$ est NC, et
2. $\forall\, x_j \in V^+(x_i), \forall\, a \in D_i, \exists b \in D_j$ telle que $w_{i,j}(a,b) \oplus w_j(b) = 0$,
3. $\forall\, x_j \in V(x_i), \forall\, a \in D_i, \forall\, b \in D_j$, $w_{i,j}(a,b) = \top$ si $w_{i,j}(a,b) \geq \top \ominus (w_i(a) \oplus w_j(b) \oplus w_\emptyset)$.

Un WCSP est DAC si toutes les variables sont DAC.

**Définition 1.23. COHÉRENCE D'ARC DIRECTIONNELLE TOTALE (FDAC, FULL DAC)**
Une variable est FDAC si elle est arc-cohérente directionnelle DAC et arc-cohérente souple SAC. Un WCSP est FDAC si toutes les variables sont FDAC.

**Définition 1.24. COHÉRENCE D'ARC EXISTENTIELLE ( EAC, EXISTENTIAL ARC-CONSISTENCY)**
[de Givry *et al.*, 2005b] Une variable $x_i$ est EAC si :

1. $x_i$ est NC, et
2. $\exists\, a \in D_i, w_i(a) = 0 \wedge \forall\, x_j \in V(x_i),\ \exists\, b \in D_j$ telle que $w_{i,j}(a,b) \oplus w_j(b) = 0$,
3. $\forall\, x_j \in V(x_i), \forall\, a \in D_i, \forall\, b \in D_j$ $w_{i,j}(a,b) = \top$ si $w_{i,j}(a,b) \geq \top \ominus (w_i(a) \oplus w_j(b) \oplus w_\emptyset)$.

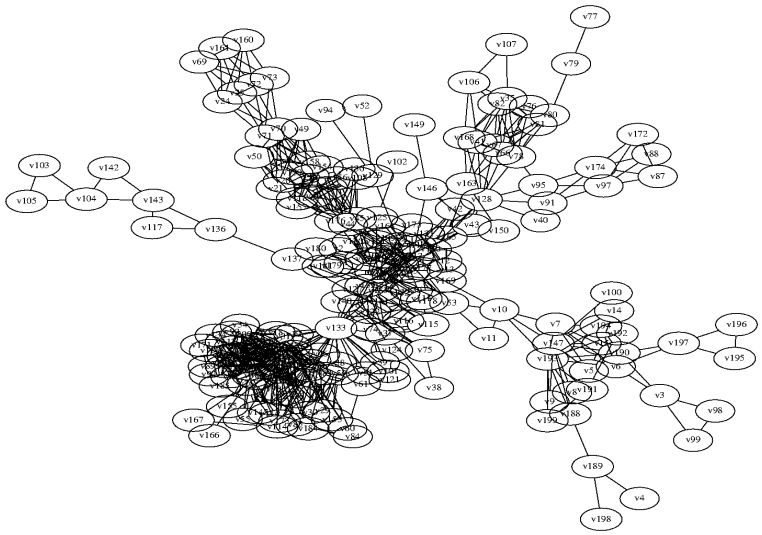



FIGURE 1.4 – Graphe de contraintes de l'instance Scen07.

Un WCSP est EAC si toutes les variables sont EAC.

La complexité de EAC est de $\mathcal{O}(nd + ed^2) = \mathcal{O}(ed^2)$ en temps et $\mathcal{O}(ed)$ en espace.

**Définition 1.25. COHÉRENCE D'ARC EXISTENTIELLE ET DIRECTIONNELLE (EDAC, EXISTENTIAL DAC)** Une variable est EDAC, si elle est arc cohérente existentielle EAC, et arc cohérente directionnelle totale FDAC. Un WCSP est un EDAC si toutes les variables sont EDAC.

La complexité de EDAC est de $\mathcal{O}(ed^2 \times \max nd, m + ed^3 + \min m, nd \times nd) = \mathcal{O}(ed^2 \times \max nd, m)$ en temps et de $O(ed)$ en espace [de Givry *et al.*, 2005b]. Il faut noter que DAC et EDAC ont été étendues aux fonctions de coût ternaires [Sánchez *et al.*, 2008] et à certaines fonctions de coût globales [Lee et Leung, 2009], mais ces extensions se heurtent à une complexité exponentielle selon l'arité de la fonction de coût considérée.

## 1.5 Exploiter la structure du problème

De nombreux problèmes réels tels que le problème d'allocation de fréquence à des liens radio (RLFAP) [Cabon *et al.*, 1999], le problème de sélection des prises de vue d'un satellite d'observation terrestre (SPOT5) [Bensana *et al.*, 1999], ou encore le problème de la sélection des marqueurs représentatifs d'une population d'individus en génétique (TagSNP) [Hirschhorn et Daly, 2005] sont très importants mais difficiles à résoudre. Cependant, la plupart d'entre eux présentent un graphe de contraintes fortement structuré comme indiqué par la figure 1.4. Exploiter cette structure peut s'avérer profitable pour surmonter la difficulté du problème, donnant ainsi, une nouvelle direction pour développer des méthodes de résolution plus efficaces.

L'une des méthodes utilisées pour révéler une telle structure est la décomposition arborescente introduite dans [Robertson et Seymour, 1986]. À l'aide de la décomposition arborescente, les algorithmes de recherche peuvent capturer des informations d'indépendance structurelle, qui seront utilisées pour obtenir des bornes de complexité en temps et en espace.

Dans cette section, nous introduisons les notions et les concepts de base liés à la décomposition arborescente. Puis, nous présentons principalement les méthodes de décomposition MCS et MINFILL.

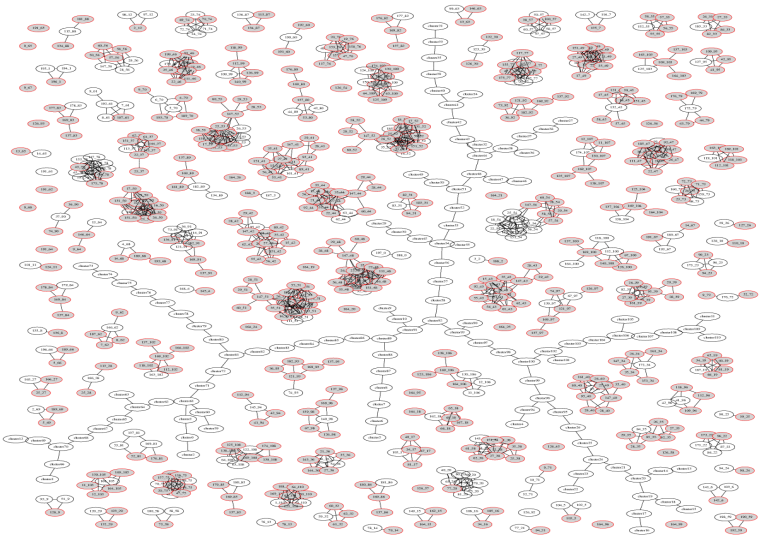


FIGURE 1.5 – La décomposition arborescente MCS de l'instance Scen07.

### 1.5.1 Décomposition arborescente

Le graphe de contraintes d'un réseau de fonction de coût $(X, D, W, S)$ est un graphe $G = (X, E)$ composé d'un sommet par variable et il existe une arête $\{u, v\} \in E$ si, et seulement si, $\exists\, w_{X_i} \in W \wedge \{u, v\} \in X_i$.

**Définition 1.26.**
Une décomposition arborescente [Robertson et Seymour, 1986] de $G = (X, E)$ est un couple $(C_T, T)$ où : $T = (I, A)$ est un arbre avec pour ensemble de nœuds $I$ et pour ensemble d'arêtes $A$, $C_T = \{C_i \mid i \in I\}$ est une famille de sous-ensembles de $X$ (appelés *clusters*) telle que :

— $\cup_{i \in I} C_i = X$,
— $\forall\, (u, v) \in E, \exists\, C_i \in C_T$ t.q. $u, v \in C_i$,
— $\forall\, i, j, k \in I$, si $j$ est sur le chemin de $i$ à $k$ dans $T$, alors $C_i \cap C_k \subseteq C_j$.

La figure 1.5 illustre la décomposition arborescente de largeur 23 en utilisant la méthode MCS (voir sec. 1.5.2a) sur l'instance `CELAR Scen07`. Cette décomposition est constituée de 110 clusters et la taille de son plus grand séparateur vaut 18. Comme on peut le remarquer, la plupart des clusters ont plus de 90% de leurs variables partagées (cf. ellipses [9] en rouge). L'arbre de la décomposition est éparpillé sur toute la figure.

**Définition 1.27. LARGEUR D'ARBRE D'UN GRAPHE $G$**
On définit par $max_{i \in I}(|C_i| - 1)$ la largeur d'une décomposition. La largeur d'arbre d'un graphe $G$ est la largeur minimale sur toutes ses décompositions arborescentes.

**Définition 1.28. SÉPARATEUR**
L'ensemble des variables partagées entre deux clusters $C_i$ et $C_j$, noté $sep(C_i, C_j)$, est appelé séparateur :

$$sep(C_i, C_j) = C_i \cap C_j.$$

**Définition 1.29. VOISINAGE, DEGRÉ**
Soit $C_i$ un cluster de $\mathcal{C}_T$. On appelle voisinage de $C_i$, noté $vois(C_i)$, l'ensemble des clusters $C_j$ qui partagent des

9. Chaque ellipse contient un numéro de la forme $x_y$, avec $x$ le numéro de la variable et $y$ son numéro du cluster.

variables avec $C_i$ :

$$vois(C_i) = \{C_j | sep(C_i, C_j) \neq \emptyset\}.$$

Le degré d'un cluster $C_i$ est le cardinal de $vois(C_i)$.

**Définition 1.30. VARIABLE PROPRE**
On appelle variables propres de $C_i$ l'ensemble $v_p(C_i)$ des variables qui n'appartiennent qu'au cluster $C_i$ :

$$v_p(C_i) = C_i \backslash \bigcup_{C_j \in vois(C_i)} sep(C_i, C_j).$$

### 1.5.2 Méthodes de décomposition arborescente

Dans cette section, nous décrivons quelques heuristiques populaires pour calculer une limite supérieure sur la largeur de l'arbre. Ces heuristiques fonctionnent en temps polynomial par rapport à la taille du graphe et constituent donc des candidats importants qui pourraient être utilisés pour placer de bonnes bornes supérieures aux méthodes de résolution (voir la section 1.6). Toutes les heuristiques décrites ci-dessous sont utilisées pour construire un ordre d'élimination parfait du graphe G. En effet, calculer la largeur de décomposition d'un graphe (c'est-à-dire trouver une décomposition optimale parmi un grand nombre de décompositions arborescentes possibles) est un problème NP-difficile. Cependant, ce problème peut être résolu en un temps polynomial si le graphe à décomposer est *triangulé* (i.e. tous ses cycles de taille supérieure à quatre ont une corde). On peut donc construire une décomposition arborescente d'un graphe non triangulé en produisant une triangulation de ce graphe.

#### 1.5.2a MCS

La méthode de décomposition MCS (*Maximum Cardinality Search*) est une méthode qui produit un ordre d'élimination parfait lorsqu'un graphe triangulé est donné en entrée. MCS traite d'abord le sommet $v$ pour lequel $\alpha(v) = n$, où $\alpha$ est un ordre d'élimination du graphe $G$ qui numérote les sommets de G avec des nombres entiers de 1 à $n$, et continue à générer un ordre d'élimination en sens inverse. L'algorithme 4 décrit le pseudo-code de MCS. À chaque itération, le sommet (noté $v$) qui a le plus grand nombre de voisins déjà choisis est sélectionné. Ensuite, le poids de chaque sommet non numéroté voisin de $v$ est incrémenté de 1. L'algorithme s'arrête quand tous les sommets de $G$ ont été numérotés. La complexité de la triangulation est en $O(n + m)$, avec $n$ le nombre de sommets et $m$ le nombre d'arêtes. La procédure *jeu-elimination* (lignes 11-17) décrit le pseudo-code du jeu d'élimination. Il reçoit en entrée un graphe $G$ et un ordre d'élimination $\alpha$ et produit en sortie un graphe $G^*$, triangulation de $G$.

#### 1.5.2b MinFill

**Définition 1.31. SOMMET SIMPLICIAL**
Un sommet est simplicial si son voisinage est une clique (graphe complet).

**Définition 1.32. ORDRE D'ÉLIMINATION SIMPLICIAL**
$[v_1, \ldots, v_i, \ldots, v_n]$ est un ordre d'élimination simplicial si $v_i$ est simplicial dans le sous-graphe $G_i = G[\{v_i, \ldots, v_n\}]$.

MINFILL [Yannakakis, 1981] ordonne les sommets de 1 à n comme suit :

$(i)$ elle sélectionne d'abord un sommet $v$ qui ajoute le moins d'arêtes lorsqu'il est éliminé du graphe et elle le place à la première position. Le sommet $v$ sera éliminé du graphe en le rendant simplicial.

**Algorithme 4 :** `MCS`

**Données** : un graphe $G = (V, E)$
**Résultat** : un ordre minimal $\alpha$ et $G^*_\alpha$

1 **début**
2 $F = \emptyset$;
3 Pour tout sommet $v$ de $V$, initialiser son poids $w(v) = 0$;
4 **pour** $i = n$ **a** *1* **faire**
5 Choisir le sommet $v$ de poids $(w(v))$ maximal;
6 **pour tous les** *sommets non numérotés $u$ de $V$* **faire**
7 **si** $\{u, v\} \in E$ **alors**
8 incrémenter $w(u)$;
9 $\alpha(v) = i$ ;
10 jeu-elimination $(G, \alpha)$ ;
11 **Procedure** *jeu-elimination $(G, \alpha)$*
12 $G^0 = G$;
13 **pour** *i = 1* **a** *n* **faire**
14 soit $v$ le sommet tel que $\alpha(v) = i$;
15 soit $F^i$ l'ensemble des arêtes nécessaires pour faire de $N_{G^{i-1}}(v)$ une clique dans $G^{i-1}$;
16 $G^i = G^{i-1}(V \setminus \{v\}, E \cup F^i)$;
17 $G^*_\alpha = (V, E \bigcup F^i)$ ;

$(ii)$ elle sélectionne n'importe quel sommet qui ajoute le moins de bords quand il est éliminé et elle le place à la position suivante dans l'ordre.

$(iii)$ le processus de rupture des liens sera répété arbitrairement.

La figure 1.6 illustre la décomposition arborescente de largeur 17 en utilisant la méthode MINFILL sur l'instance `CELAR Scen07`. Cette décomposition est constituée de 123 clusters et la taille de son plus grand séparateur vaut 17.

### 1.5.3 Raffinement des décompositions arborescentes

Dans sa thèse, M. Fontaine [Fontaine, 2013] a constaté que la décomposition arborescente, telle qu'illustrée dans la section 1.5.1, présente quelques inconvénients pour la mise en oeuvre de leur méthode DGVNS (voir le chapitre 2), ceci est dû à la redondance des variables entre clusters. Pour renforcer la qualité des structures de voisinage explorées dans DGVNS, il a proposé de fusionner les clusters qui sont redondants dans la décomposition arborescente. L'idée sous-jacente est d'augmenter la proportion de variables propres dans les clusters. À cet effet, nous reprenons le critère appelé *absorption*.

**Définition 1.33.** ABSORPTION
Soit $(\mathcal{C}_T, T)$ une décomposition arborescente, $C_i$ et $C_j$ deux clusters de $\mathcal{C}_T$. L'absorption de $C_i$ par $C_j$, notée $abs(C_i, C_j)$, est définie par :

$$abs(C_i, C_j) = \frac{|sep(C_i, C_j)|}{|C_i|}.$$

La figure 1.7 donne la nouvelle décomposition obtenue après fusion par absorption. Comme on peut le constater, la fusion par absorption permet de supprimer la plupart des clusters redondants et de générer une décomposition contenant au moins 50% de variables propres. Par ailleurs, la taille du plus grand cluster (resp. séparateur) est égale

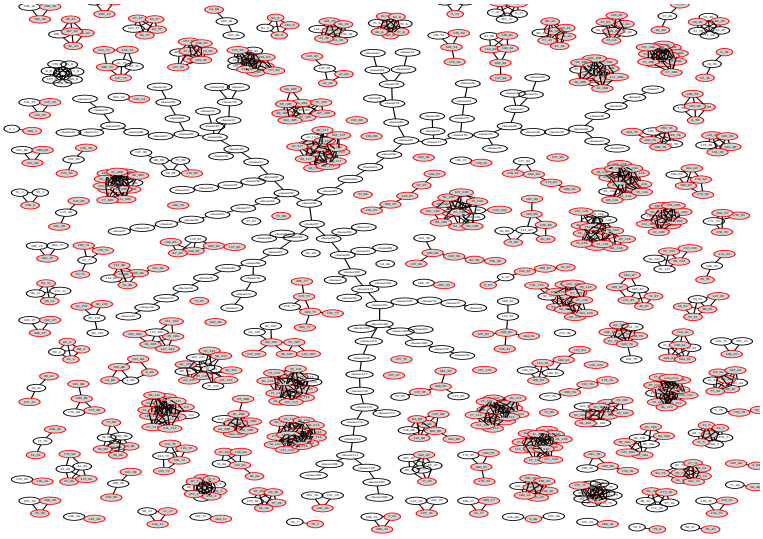


FIGURE 1.6 – La décomposition arborescente MINFILL de l'instance Scen07.

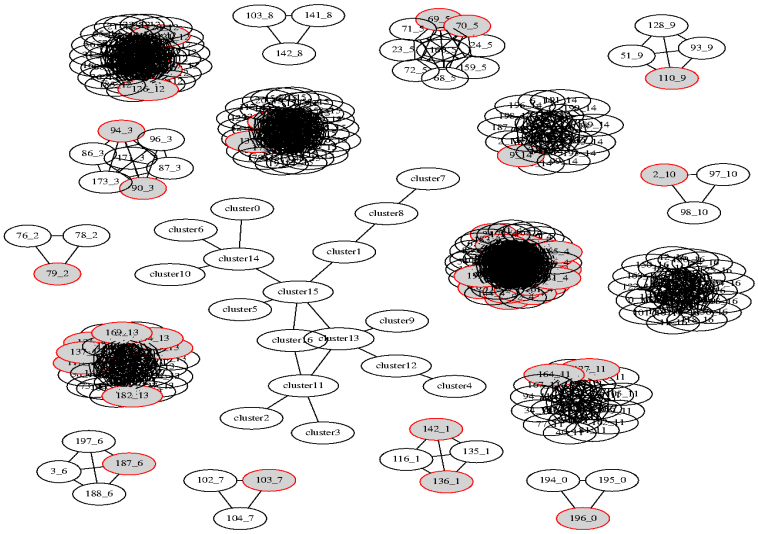



FIGURE 1.7 – Le raffinement de la décomposition arborescente MCS de l'instance Scen07.

à 49 (resp. 8). Ainsi, la fusion par absorption permet de produire des décompositions de bonne qualité : des clusters de tailles raisonnables avec une forte proportion de variables propres.

## 1.6 Méthodes complètes exploitant la décomposition arborescente pour les WCSP

Les algorithmes exploitant la décomposition arborescente proposés jusqu'à maintenant sont principalement des algorithmes exacts. Une première exploitation de la décomposition arborescente dans le cadre des méthodes de recherche locale a été proposée par M. Fontaine dans sa thèse [Fontaine, 2013].

### 1.6.1 Cluster Tree Elimination CTE

L'algorithme CTE propose d'exploiter la décomposition arborescente afin de supprimer les variables par groupes. Il utilise pour cela les clusters et se base sur l'arbre de clusters. En partant des feuilles et en remontant jusqu'à la racine, CTE réalise la suppression groupée des variables propres des clusters. L'efficacité de CTE repose donc à la fois sur la largeur de décomposition d'un point de vue temps de calcul, mais aussi sur la taille des séparateurs pour ce qui est de l'espace mémoire nécessaire. En effet, soit $e$ le nombre de contraintes, $n$ le nombre de variables, $deg$ le plus grand degré d'un cluster dans la décomposition, $d$ la taille du plus grand domaine, $w$ la largeur de la décomposition et $sep$ la taille du plus grand séparateur. La complexité temporelle de CTE est $O((e+n)deg.k^{w+1})$ et sa complexité spatiale est $O(n.k^{sep})$. En pratique, CTE n'est utilisable que pour des problèmes dont la décomposition possède une très faible largeur de décomposition.

### 1.6.2 Backtrack Bounded By Tree-Decomposition BTD

BTD [Jégou et Terrioux, 2003] est une méthode énumérative qui exploite la décomposition arborescente à la fois pour guider la recherche et pour améliorer le filtrage. Pour cela, BTD définit un ordre partiel des variables du problème dans lequel toutes les variables d'un cluster fils seront placées après les variables du cluster père. BTD effectue une recherche en profondeur d'abord dans laquelle les variables seront instanciées selon cet ordre. Ainsi, BTD affecte les variables du problème de la racine aux feuilles de la décomposition. Cela lui permet de décomposer le problème en sous-problèmes. La complexité temporelle de BTD est en $O(n.sep^2.e.log(d^s).d^{w+1})$ et sa complexité spatiale est en $O(n.sep.d^s)$ [Jégou et Terrioux, 2003], avec $e$ le nombre de contraintes, $n$ le nombre de variables, $deg$ le plus grand degré d'un cluster dans la décomposition, $d$ la taille du plus grand domaine, $w$ la largeur de la décomposition et $sep$ la taille du plus grand séparateur. Il est à noter qu'une extension, nommée $BTD_{val}$, a été proposée dans le cadre des WCSP. On constate une nouvelle fois l'importance de la largeur de décomposition et de la taille du plus grand séparateur, qui sont les deux facteurs clés de la complexité spatiale et temporelle de BTD.

### 1.6.3 AND/OR tree search

Une seconde méthode énumérative est basée sur une autre approche de l'exploration de l'espace de recherche, le parcours d'arbre AND/OR [Marinescu et Dechter, 2009]. Dans une recherche arborescente classique, on fait correspondre à chaque nœud de l'arbre de recherche une variable et à chaque arête une affectation. Dans un arbre AND/OR, on alterne successivement des nœuds AND et OR. Les nœud AND correspondent à un choix de variable $x_i$ et les nœuds OR à l'affectation de $x_i$ aux différentes valeurs de son domaine $D_i$. Une solution est définie par un sous-arbre $T$ dans lequel :

— la racine de l'arbre de recherche appartient à $T$ ;

— si un nœud $u \in T$, tous ses fils appartiennent à $T$ ;
— si un nœud OR $u \in T$, un seul de ses fils appartient à $T$ ;
— toutes ses feuilles correspondent à une affectation cohérente.

Plusieurs algorithmes de résolution basés sur les arbres AND/OR ont été proposés pour la résolution de WCSP. Par exemple, AOBB [Marinescu et Dechter, 2009] est basé sur un *branch-and-bound*. Il parcourt l'arbre AND/OR en profondeur d'abord en maintenant une borne inférieure de la solution courante. À chaque nœud est associé le coût du sous-arbre dont il est la racine. Lorsque celui-ci n'est pas connu, on calcule une borne inférieure grâce aux méthodes de cohérence (cf la section 1.4.3). Un nœud $u$ de l'arbre sera filtré si la borne inférieure dépasse la borne supérieure de la meilleure solution trouvée pour le sous-problème formé par les nœuds AND suivant $u$ dans l'arbre de recherche. L'algorithme alterne entre deux phases : l'extension de l'affectation courante et la révision de la borne supérieure des ancêtres pour la solution courante dans l'arbre de recherche. La complexité de cet algorithme est $O(nd^w)$. La complexité de l'algorithme repose donc sur la structure du graphe de contraintes, et plus précisément sur sa largeur de décomposition.

## 1.7 Conclusion

Nous avons introduit dans ce chapitre la programmation linéaire en nombres entiers, les réseaux de fonction de coût et les notions liées à la décomposition arborescente, qui vont nous servir pour élaborer nos contributions sur les méthodes parallèles exploitant la décomposition arborescente ainsi que notre approche basée sur la PLNE pour traiter le problème d'extraction des ensembles de motifs.

Nous présentons dans le chapitre suivant un état de l'art sur les méthodes de recherche à voisinage variable, et les travaux menés au sein de ces méthodes pour exploiter le parallélisme.

# Chapitre 2

# Méthodes de recherche à voisinage variable

## Sommaire


**2.1 Recherche locale** . . . . . **28**
**2.2 Métaheuristiques** . . . . . **30**
**2.3 Recherche à voisinage variable** . . . . . **32**
2.3.1 Schéma de base . . . . . 32
2.3.2 Recherche à voisinage variable réduite . . . . . 33
2.3.3 Recherche à voisinage variable biaisée . . . . . 34
2.3.4 Recherche à voisinage variable avec décomposition . . . . . 34
2.3.5 Recherche à voisinage variable primale-duale . . . . . 35
2.3.6 Recherche à formulation variable . . . . . 35
2.3.7 Recherche à voisinage large . . . . . 36
**2.4 Extensions de VNS pour la résolution des WCSP** . . . . . **37**
2.4.1 VNS/LDS+CP . . . . . 37
2.4.2 VNS guidée par la décomposition arborescente (DGVNS) . . . . . 38
**2.5 Stratégies parallèles pour VNS** . . . . . **39**
2.5.1 Parallélisme . . . . . 39
2.5.2 Parallélisation de VNS . . . . . 40
2.5.3 Recherche VNS parallèle synchrone . . . . . 40
2.5.4 Recherche VNS parallèle répliquée (RPVNS) . . . . . 40
2.5.5 Recherche VNS parallèle avec phase de shaking répliquée (RSVNS) . . . . . 40
2.5.6 Recherche VNS parallèle à voisinages coopératifs (CNVNS) . . . . . 41
**2.6 Évaluation des métaheuristiques** . . . . . **41**
2.6.1 Temps CPU et taux de réussite . . . . . 42
2.6.2 Profil de performance . . . . . 43
2.6.3 Profils de rapport de performance . . . . . 43
**2.7 Conclusion sur les métaheuristiques** . . . . . **44**


La résolution des problèmes d'optimisation combinatoire par des méthodes exactes comme le branch-and-bound ne peut être envisagée sur des instances de problèmes de grande taille en raison des délais d'exécution généralement exorbitants que nécessiteraient ces méthodes pour atteindre la (les) solution(s) optimale(s). Une alternative, dans ces cas, est de recourir aux méthodes approchées qui permettent de fournir des solutions acceptables en temps raisonnable sur des instances de grande taille.

On distingue deux classes de méthodes approchées, *les méthodes d'approximation* qui sont capables de donner une borne d'approximation sur la qualité des solutions et *les métaheuristiques* qui sont des méthodes génériques pouvant être utilisées pour optimiser une variété de problèmes. Le processus de résolution mis en oeuvre par une métaheuristique s'inspire généralement des phénomènes naturels ou physiques. Leur essence est d'éviter de se faire piéger dans les optimums locaux, tout en gardant un bon compromis entre diversification et intensification de la recherche pour converger vers des solutions de bonne qualité.

Les calculateurs de nos jours sont naturellement multi-coeurs. L'exploitation de la puissance de calcul disponible dans un noeud de calcul est essentielle pour, d'une part, accélérer le processus de résolution et permettre ainsi d'attaquer des problèmes de grande taille et, d'autre part, améliorer la qualité des solutions. L'hybridation (coopération) de plusieurs méthodes de résolution à caractéristiques complémentaires est une approche courante en optimisation. Les méthodes hybrides qui résultent de la coopération de plusieurs méthodes d'optimisation de base sont généralement plus efficaces et plus robustes que les méthodes d'optimisation de base.

**Plan du chapitre.** La section 2.1 est consacrée aux méthodes de recherche locale. Nous y définissons la terminologie liée à cette classe de métaheuristiques, nous présentons les étapes principales d'une recherche locale de base et donnons quelques variantes. Nous présentons, dans la section 2.2, une classification des métaheuristiques proposées dans la littérature. Nous nous focalisons, dans la section 2.3, sur la méthode à voisinage variable et ses variantes. Dans la section 2.4, nous présentons DGVNS (Decomposition Guided VNS), un algorithme qui exploite la décomposition arborescente d'un réseau de fonctions de coût (cf. Chapitre 1) pour guider la recherche VNS vers des voisinages potentiellement prometteurs. Dans la section 2.5, nous donnons quelques stratégies de parallélisation de VNS. Enfin, nous présentons dans la section 2.6 quelques critères que nous avons retenus pour l'évaluation des métaheuristiques.

## 2.1 Recherche locale

Soient $S$ l'ensemble des solutions d'un problème d'optimisation combinatoire, et $f$ une fonction réelle qui attribue à chaque solution $s \in S$ un coût $f(s)$.

Une méthode de recherche locale est une méthode d'optimisation qui consiste à démarrer d'une solution initiale $s_0 \in S$ (générée aléatoirement ou construite au moyen d'une heuristique) et à effectuer une succession de mouvements de la solution courante vers une solution de son voisinage jusqu'à atteindre un nombre maximal d'itérations ou épuiser un timeout. Une méthode de recherche locale est basée sur deux éléments essentiels :

- Une structure de *voisinage* permettant de définir un ensemble de solutions proches de la solution courante,
- Une *stratégie d'exploration* du voisinage.

**Définition 2.1. VOISINAGE D'UNE SOLUTION**
Soit $S$ l'ensemble des solutions réalisables d'un problème d'optimisation. On appelle **voisinage** l'application $N$ : $S \rightarrow 2^S$ associant à toute solution $s \in S$ un ensemble de solutions $N(s)$ appelé voisinage de $s$.

**Définition 2.2. MOUVEMENT**
On appelle **mouvement** l'opération qui consiste à passer d'une solution $s \in S$ à une solution voisine $s' \in N(s)$.

**Définition 2.3. MINIMUM LOCAL À UN VOISINAGE**
Une solution $s \in S$ est dite minimum localement au voisinage $N$ si et seulement si : $\forall s' \in N(s)$, $f(s) \leq f(s')$.

L'algorithme 5 illustre le pseudo-code générique d'une recherche locale, où $iter_{max}$ est le nombre maximal de mouvements autorisés sans amélioration de la qualité de la solution, et $TimeOut$ est le temps total alloué à la recherche. L'algorithme démarre d'une solution initiale $s$ (ligne 3), puis tente de l'améliorer en enchaînant une succession de mouvements locaux dans son voisinage. À chaque mouvement une solution $s'$, si possible de

**Algorithme 5 :** Pseudo-code générique d'une recherche locale.

1 `Fonction rechercheLocale(`$S, f, iter_{max}, N$`) ;`
2 **début**
3 $s \leftarrow$ `genSolInit(`$S$`) ;`
4 $iter \leftarrow 1$ ;
5 **tant que** $(iter < iter_{max}) \wedge (not\ TimeOut)$ **faire**
6 $s' \leftarrow$ `selectVoisin(`$s, N$`) ;`
7 **si** $(f(s') < f(s))$ **alors**
8 $s \leftarrow s'$ ;
9 $iter \leftarrow 1$ ;
10 **sinon**
11 $iter \leftarrow iter + 1$ ;
12 **retourner** $s$ ;

meilleure qualité que la solution courante $s$, est sélectionnée dans le *voisinage* $N(s)$ (ligne 6). Si la solution courante ne peut être améliorée pendant $iter_{max}$ mouvements, ou que le *TimeOut* est atteint (ligne 5), l'algorithme retourne la meilleure solution trouvée $s$ (ligne 12).

**Algorithme 6 :** Exploration du voisinage : *Première amélioration*.

`Fonction selectPremierVoisin(`$s, N$`) ;`
**début**
**répéter**
$s' \leftarrow s, i \leftarrow 0;$
**répéter**
$i \leftarrow i + 1;$
$s \leftarrow arg\min\{f(s), f(s_i)\}, s_i \in N(s)$ ;
**jusqu'à** $(f(s) < f(s_i)$ **ou** $i =\mid N(s) \mid);$
**jusqu'à** $f(s) \geq f(s');$
**retourner** $s'$ ;

Plusieurs stratégies d'exploration du voisinage peuvent être utilisées pour définir un mouvement. Elles peuvent être déterministes ou stochastiques. Une première stratégie (algorithme 6), dite **première amélioration**, consiste à énumérer les voisins de la solution courante $s$ jusqu'à en rencontrer un qui l'améliore, le prochain mouvement sera effectué sur ce voisin.

Une autre stratégie (algorithme 7), dite **meilleure amélioration**, consiste à explorer le voisinage $N(s)$ de manière complète afin de rechercher le meilleur voisin. Pour éviter des biais dus à des choix déterministes qui feraient explorer toujours les mêmes parties de l'espace de recherche, une part d'aléatoire est introduite dans ces algorithmes les rendant stochastiques. En particulier, sur le choix du voisin ou sur le choix de la solution initiale qui peut être complètement aléatoire.

**Minima locaux.** L'algorithme de *descente* (ou *Hill-Climbing*) est une recherche locale de base qui n'accepte que des voisins améliorant la solution courante. Cet algorithme est systématiquement piégé dans le premier minimum local rencontré lors de la recherche. De nombreux mécanismes ont alors été conçus pour contrôler les méthodes de recherche locale et les guider pour éviter qu'elles ne restent piégées dans un minimum local. On les appelle généralement des *métaheuristiques* [10].

10. Le lecteur intéressé par une synthèse des métaheuristiques pourra se référer à [Talbi, 2009], pour de plus amples détails.

**Algorithme 7 :** Exploration du voisinage : *Meilleure amélioration*.

```
1 Fonction selectMeilleurVoisin(s, N) ;
2 début
3     répéter
4         s' ← s ;
5         s ← arg min_{y∈N(s)} f(y);
6     jusqu'à (f(s) ≥ f(s'));
7     retourner s' ;
```

Nous présentons, dans ce document, une brève description des métaheuristiques en général, pour se focaliser ensuite sur les métaheuristiques faisant parties de la thèse et nos contributions.

## 2.2 Métaheuristiques

Au cours des dernières décennies, un nouveau type de méthodes d'optimisation approchées, appelées métaheuristiques, a émergé. Ce sont des méthodes génériques conçues pour explorer efficacement un espace de recherche et applicables à une variété de problèmes d'optimisation. Une métaheuristique englobe des stratégies guidant le processus d'exploration de l'espace de recherche pour trouver efficacement des solutions de meilleure qualité. Elle peut être constituée d'une simple recherche locale ou de processus d'apprentissage complexes, utilisant des mécanismes sophistiqués pour ne pas stagner sur des minima locaux. Dans ce qui suit, nous présentons une classification des différentes métaheuristiques classiques les plus répandues dans la littérature.

Selon les caractéristiques sélectionnées pour différencier les métaheuristiques, il est possible de répartir celles-ci en plusieurs classes, chacune d'elles étant le résultat d'un point de vue spécifique. La figure 2.1 présente une liste non exhaustive des métaheuristiques existantes, qui sont réparties en différentes classes suivant les mécanismes utilisés pour s'échapper ou éviter les minima locaux. Le lecteur intéressé par une synthèse des métaheuristiques classiques pourra se référer à [Aupetit et Slimane, 2016, Talbi, 2009] pour de plus amples détails. Nous retrouvons aussi deux autres classes de métaheuristiques connues par hybrides et parallèles qui profitent des métaheuristiques existantes pour combiner les avantages de chacune et booster le processus de résolution en utilisant des modèles de coopération entre plusieurs processus. Pour plus de détails sur ces deux classes, le lecteur pourra se référer à [Blum *et al.*, 2011, Alba *et al.*, 2013]. Nous résumons, ci-dessous les principales classes des métaheuristiques.

**Trajectoires vs. discontinues.** Une des distinctions importantes entre les différentes métaheuristiques est de savoir si celles-ci continuent la recherche sur une seule trajectoire, qui correspond à des mouvements sur des voisinages proches de la solution, ou des sauts plus grands sont autorisés sur les voisinages. La méthode de recherche taboue [Glover et Laguna, 1997] et le recuit simulé [Kirkpatrick *et al.*, 1983] sont des exemples typiques de métaheuristiques à trajectoire. Ces méthodes utilisent des mouvements sur des solutions non améliorantes pour s'échapper des minima locaux. Ainsi d'autres méthodes de recherche locale peuvent être dans la même classe telles que l'algorithme Variables Depth Search (`VDS`) [Lin et Kernighan, 1973] et les algorithmes de chaîne d'éjection [Glover, 1996]. Les métaheuristiques telles que la méthode de colonies de fourmis (ACO), la méthode de recherche itérative (ILS), les algorithmes génétiques (GA), génèrent souvent plusieurs points de départ pour les prochaines itérations. Par exemple, cela est réalisé en construisant une solution par les fourmis (dans ACO), de modifier des optima locaux déjà visités (dans ILS), l'application des opérations génétiques (dans GA), etc. La génération de nouvelles solutions de départ correspond à des sauts dans l'espace de recherche caractérisés par une suite de mouvements discontinus.

**Solution unique vs. population de solutions.** Une autre caractéristique qui distingue les métaheuristiques est le nombre de solutions utilisées à chaque itération. Il y a une forte corrélation entre ces deux classes et les classes à trajectoire et discontinue. Souvent, les méthodes à trajectoire utilisent une solution unique pour chaque itération

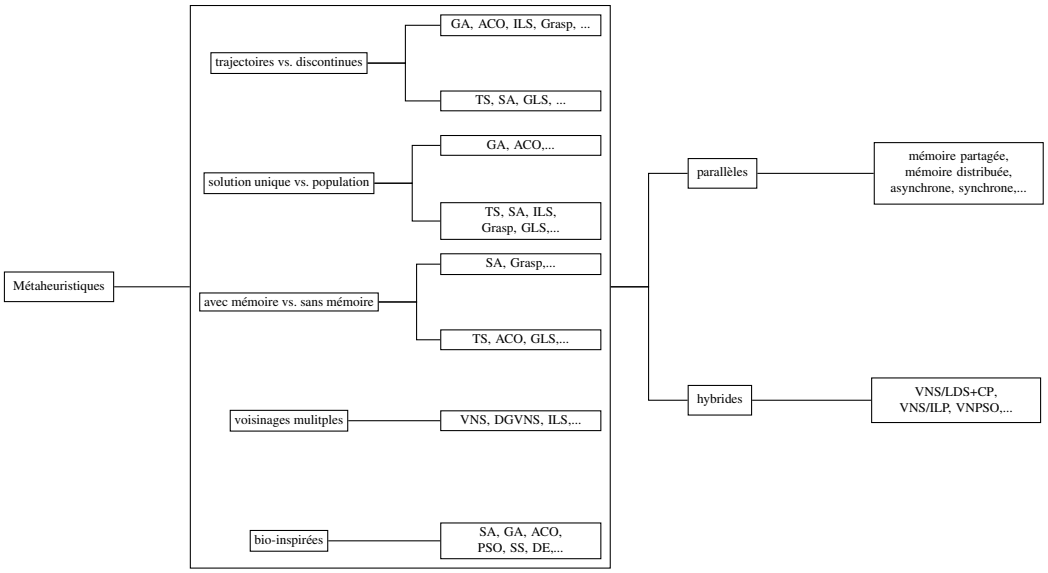



FIGURE 2.1 – Classification d'une liste non exhaustive des métaheuristiques classiques.

alors que les méthodes discontinues sont à population de solutions. La classe des métaheuristiques à solution unique englobe les méthodes de recherche locale telles que la méthode taboue, la recherche locale itérative ou les méthodes de recherche à voisinage variable. Cependant, les métaheuristiques à population de solutions telles que les algorithmes génétiques [Mitchell, 1998], les algorithmes de colonies de fourmis [Dorigo et Stützle, 2004], ou l'optimisation par essaims de particules [Kennedy et Eberhart, 1995], effectuent des processus de recherche qui décrivent l'évolution d'un ensemble de points dans l'espace de recherche.

**Avec mémoire vs. sans mémoire.** Un troisième élément qui caractérise les métaheuristiques est l'exploitation ou non de l'historique de la recherche pour décider des futurs mouvements à effectuer. L'exploitation de l'historique de recherche nécessite l'utilisation d'une mémoire pour sauvegarder certaines informations liées à l'amélioration de la recherche ou sa configuration. Les algorithmes sans mémoire exécutent un processus de Markov pour déterminer le prochain mouvement en se basant uniquement sur l'état actuel de la recherche. L'utilisation de la mémoire peut être différente d'une méthode à une autre. Souvent, nous différencions l'utilisation de la mémoire à court terme et à long terme. La mémoire à court terme mémorise les mouvements effectués récemment, les solutions visitées ou, en général, les décisions prises. La mémoire à long terme sert à mémoriser les paramètres synthétiques de la recherche. La méthode taboue est un exemple typique de ce genre de méthodes, qui utilise explicitement la mémoire pour s'échapper des optima locaux. L'utilisation de la mémoire est, aujourd'hui, reconnue comme l'un des éléments fondamentaux d'une bonne métaheuristique.

**voisinage unique vs. voisinages multiples.** La majorité des méthodes de recherche locale utilise une seule structure de voisinage qui définit un seul type de mouvement. En d'autres termes, le paysage de recherche ne change pas au cours de l'algorithme. C'est le cas de la méthode taboue ou de la recherche locale itérative qui, exceptionnellement, utilisent deux structures de voisinages ($N$ et $N'$), où le voisinage $N$ est utilisé pour améliorer la solution courante jusqu'à tomber sur un optimum local, le voisinage $N'$ est utilisé pour diversifier les mouvements sur la solution courante. D'autres métaheuristiques, telles que la recherche à voisinage variable, utilisent plusieurs structures de voisinages ($N_1 \ldots N_k$), généralement de tailles différentes, qui permettent de diversifier la recherche en changeant systématiquement de voisinage lors de l'exploration de l'espace des solutions.

**Bio-inspirée.** Un autre critère de classement des métaheuristiques est relatif à leurs origines. En effet, plusieurs métaheuristiques s'inspirent de phénomènes naturels ou physiques dans leur processus d'exploration de l'espace

de recherche. Les algorithmes génétiques, les colonies de fourmis ou encore le recuit simulé en font partie. L'optimisation par les colonies de fourmis, par exemple, reproduit les mécanismes utilisés par les colonies de fourmis pour chercher de la nourriture. Quand une fourmi trouve une source de nourriture, elle attire ses congénères sur ce chemin en traçant son passage par un phénomène chimique. Plus les fourmis emprunteront ce chemin, plus la quantité de phéromones augmentera, rendant le chemin de plus en plus attrayant [Colorni *et al.*, 1992]. D'autres métaheuristiques, par contre, sont non bio-inspirées. La recherche à voisinage variable, par exemple, est une métaheuristique de cette classe.

**Hybrides.** Une direction de recherche très prometteuse est l'hybridation des métaheuristiques. Beaucoup de travaux ont été menés dans cette direction dans l'objectif de combiner les avantages de différentes métaheuristiques. Différents schémas d'hybridation ont été envisagés. Un premier schéma consiste à inclure des composants d'une métaheuristique dans une autre. Un exemple typique de ce schéma d'hybridation est l'intégration d'un algorithme de recherche à solution unique dans un algorithme à population de solutions. Un deuxième schéma d'hybridation est une hybridation que nous qualifions d'hybridation par composition d'algorithmes. Ici les différentes recherches collaborent de manière indirecte, tout en gardant une certaine forme d'autonomie. Un troisième schéma est l'hybridation de méthodes complètes avec des méthodes approchées. Un exemple d'hybridation de ce type est l'utilisation d'un algorithme approché pour fournir un bon majorant de départ à un algorithme complet, afin que celui-ci réalise de bonnes coupes. Le lecteur intéressé pourra se référer à une synthèse des métaheuristiques hybrides dans [Talbi, 2013].

**Parallèles.** Bien que les métaheuristiques fournissent des solutions de bonne qualité en temps raisonnable comparativement aux méthodes exactes, elles restent cependant coûteuses en temps de calcul pour des instances de problèmes de grande taille. De plus, les calculateurs actuels sont naturellement parallèles. Par conséquent, la parallélisation des processus de recherche est impérative pour réduire le temps de résolution et améliorer la qualité des solutions fournies. Le lecteur intéressé pourra se référer à une synthèse des métaheuristiques parallèles dans [Blum *et al.*, 2011, Alba *et al.*, 2013].

Dans ce qui suit, nous présentons un état de l'art des différents travaux menés sur la recherche VNS, qui est au cœur de nos contributions.

## 2.3 Recherche à voisinage variable

### 2.3.1 Schéma de base

La recherche à voisinage variable (Variable Neighborhood Search, VNS), introduite dans [Mladenovic et Hansen, 1997], est une métaheuristique qui utilise systématiquement plusieurs structures de voisinages, dans le but d'échapper aux minima locaux. Elle comporte trois phases principales :

- *une phase stochastique* (dite phase de *shaking*) pour éviter des cycles infinis qui pourraient se produire si une règle déterministe est appliquée,
- *une phase de recherche locale* qui permet de converger vers un optimum local,
- *une phase de changement de voisinage* qui permet de *diversifier* l'exploration de l'espace de solutions afin d'accéder à un plus grand nombre de régions intéressantes.

Soit $L = \{N_1, \ldots, N_{k_{max}}\}$ une liste finie de *structures de voisinage*, ordonnée par ordre croissant de taille, où $N_k(s)$ $(k = 1 \ldots k_{max})$, désigne l'ensemble des solutions voisines de $s$ dans $N_k$. Dans la plupart des méthodes de recherche locale $k_{max} = 1$.

L'algorithme 8 donne le pseudo-code de VNS, avec $N_k$ $(k = 1, \ldots, k_{max})$ la structure de voisinage d'index $k$ et $t_{max}$ le temps maximal alloué à la recherche ($TimeOut$). L'algorithme démarre d'une solution initiale $s$ souvent générée aléatoirement. À chaque itération de la boucle des lignes (5-8), une solution $s'$ est sélectionnée

**Algorithme 8 :** Pseudo-code de la recherche VNS.

```
1  Fonction VNS(s, k_max, t_max);
2  début
3      répéter
4          k ← 1;
5          répéter
6              s' ← Shake(s,N_k) /* Phase de Perturbation */ ;
7              s'' ← RechercheLocale(s') /* Phase d'une recherche locale */ ;
8              NeighbourhoodChange(s, s'', k) /* Changement de voisinage */ ;
9          jusqu'à k = k_max;
10         t ← CpuTime();
11     jusqu'à t > t_max;
12     retourner s ;
13 Procédure NeighbourhoodChange(s, s', k);
14 début
15     si (f(s') < f(s)) alors
16         s ← s' ; k ← 1 /* Effectuer le mouvement et réinitialiser le voisinage */ ;
17     sinon k ← k + 1 /* Prochain voisinage */ ;
```

aléatoirement (phase de *shaking*) dans le voisinage $N_k$ de $s$ ($s' \in N_k(s)$) (ligne 6). Une recherche locale est appliquée à $s'$ et produit une nouvelle solution $s''$ (ligne 7). La procédure de changement de voisinage est ensuite appelée avec les paramètres $s$, $s''$ et $k$ (ligne 8). Si $s'$ est de meilleure qualité que $s$ (ligne 15), alors $s'$ devient la solution courante et l'index du voisinage est réinitialisé à 1 (ligne16). Sinon, l'index du voisinage est incrémenté pour passer au voisinage $N_{k+1}$ (ligne 19). L'algorithme s'arrête dès qu'on atteint le $TimeOut$.

La recherche VNS a été utilisée avec succès pour résoudre une variété de problèmes d'optimisation. L'un des premiers problèmes traités par cette approche est le problème du voyageur de commerce [Hansen et Mladenović, 1999, Hansen et Mladenović, 2006]. De nombreuses variantes de VNS ont été proposées dans la littérature. Nous passons en revue les plus connues dans les sections suivantes.

### 2.3.2 Recherche à voisinage variable réduite

Cette variante de VNS a été proposée et étudiée dans [Hansen et Mladenovic, 2001]. Elle consiste à explorer les différents voisinages de manière complètement aléatoire. L'algorithme `RVNS` reprend globalement le même schéma basique d'une recherche VNS à l'exception de la phase d'amélioration par une recherche locale (algorithme 8). A chaque itération, une nouvelle solution est choisie aléatoirement dans le voisinage courant $N_k$. La nouvelle solution est comparée à la solution courante. Si elle est de meilleure qualité, la solution courante est mise à jour par la nouvelle solution et la recherche continue.

Cette méthode peut être utilisée pour résoudre des problèmes de (très) grande taille pour lesquels l'utilisation d'une recherche locale peut s'avérer très coûteuse lors de la phase d'amélioration. Les auteurs dans [Sevkli et Sevilgen, 2008] ont proposé une méthode hybride intégrant une recherche VNS réduite dans un algorithme d'essaims de particules. Les auteurs dans [Xiao *et al.*, 2011] ont proposé une approche basée sur la recherche VNS réduite pour résoudre un problème connu sous la terminologie anglaise de *multi-level lot sizing problem*.

**Algorithme 9 :** Pseudo-code de la recherche VNS biaisée.

```
1  Fonction SVNS(s, k_max, t_max);
2  début
3      répéter
4          s* ← s;
5          k ← 1;
6          répéter
7              s' ← Shake(s,N_k) ;
8              s'' ← RechercheLocale(s') ;
9              keepBest(s*, s) ;
10             NeighbourhoodChange(s, s'', k)
11         jusqu'à k = k_max;
12         t ← CpuTime();
13     jusqu'à t > t_max;
14     retourner s ;
15 Procédure NeighbourhoodChange(s, s', k);
16 début
17     si (f(s') − α × d(s', s) < f(s)) alors
18         s ← s' ; k ← 1 ;
19     sinon k ← k + 1 ;
```

### 2.3.3 Recherche à voisinage variable biaisée

L'exploration et l'évaluation efficace des régions de l'espace de recherche contenant de bonnes solutions qui sont très éloignées [11] entre elles est l'un des défis majeurs des métaheuristiques en général et de la recherche VNS en particulier. La recherche VNS biaisée (connue sous la terminologie anglaise de Skewed VNS) a été proposée notamment pour adresser cette limitation. L'idée est de centrer la recherche autour d'une solution de bonne qualité, qui ne soit pas nécessairement meilleure que la solution courante. Cela est fait en intégrant dans la fonction d'évaluation une distance entre la meilleure solution courante et la solution à évaluer, notamment pour autoriser des mouvements qui ne soient pas nécessairement de meilleure qualité.

L'algorithme 9 illustre le pseudo-code de `SVNS`. La différence de cette variante par rapport au schéma basique de VNS est l'étape de mise à jour de la meilleure solution $s^*$ (ligne 9) qui est insérée entre la recherche locale et le changement de voisinage (pour ne pas perdre la meilleure solution). Le changement de la solution courante inclut la fonction de distance entre la nouvelle solution et la solution courante (ligne 17). Cette méthode a été utilisée avec succès dans [Brimberg *et al.*, 2006] pour résoudre le problème de k-cardinalité dans un graphe.

### 2.3.4 Recherche à voisinage variable avec décomposition

Les grands voisinages constituent une alternative très intéressante pour diversifier la recherche et assurer la convergence vers un optimum local de (très) bonne qualité. Toutefois, sur des problèmes de grande taille, l'exploration de ces voisinages peut s'avérer très coûteuse en temps et peut ralentir considérablement l'amélioration de la solution courante. Pour privilégier l'exploration de petits voisinages, la recherche à voisinage variable avec décomposition (connue sous la terminologie anglaise de Variable Neighborhood Decomposition Search VNDS) [Hansen et Mladenović, 2001] a été proposée, et vise à réduire l'espace de solutions parcouru par la recherche locale.

11. Nécessitant beaucoup de mouvements.

**Algorithme 10 :** Pseudo-code de la recherche VNS avec décomposition.

```
1  Fonction VNDS(s, k_max, t_max);
2  début
3      répéter
4          k ← 1;
5          répéter
6              s' ← Shake(s,N_k) ;
7              y ← s'\s ;
8              y' ← VNS(y, k, t_max) ;
9              s'' ← (s'\y) ∪ y' ;
10             NeighbourhoodChange(s, s'', k)
11         jusqu'à k = k_max;
12         t ← CpuTime();
13     jusqu'à t > t_max;
14     retourner s ;
```

L'algorithme 10 détaille le pseudo-code de la recherche VNS avec décomposition. À chaque itération, la recherche est effectuée uniquement sur un sous-problème déterminé par une heuristique. Le principe est de fixer une sous-partie du problème et de rechercher de nouvelles solutions en ne modifiant que les parties « libres » du problème. Cette recherche est effectuée par la recherche VNS appliquée au sous-problème déterminé précédemment.

Parmi les applications de cette variante, nous pouvons citer le travail original de [Hansen et Mladenović, 2001] pour la résolution du problème du *p-médiane*, ou encore le travail réalisé dans [Lazić *et al.*, 2010], où les auteurs proposent une heuristique hybride pour la résolution du problème en variables 0-1 mixtes en utilisant le principe de la recherche VNS avec décomposition.

### 2.3.5 Recherche à voisinage variable primale-duale

La recherche à voisinage variable primale-duale a été introduite dans [Hansen *et al.*, 2007] dont l'objectif est de donner une évaluation sur la qualité des solutions obtenues. Les métaheuristiques ne donnent aucune garantie sur l'optimalité de la solution ou une approximation de cette dernière. Généralement, il est difficile de donner avec précision l'écart entre la valeur de la solution finale retournée par l'approche heuristique et la valeur optimale. Pour cela, les auteurs appliquent, dans une première phase, un algorithme VNS pour obtenir une solution faisable du problème primal. Cette solution sera utilisée dans une deuxième phase pour déduire une solution du problème dual. Enfin, et selon le problème traité, les conditions des écarts de complémentarité peuvent être utilisées pour améliorer les bornes obtenues, sachant qu'il est possible d'utiliser d'autres variantes de VNS pour les deux phases.

Cette variante a été appliquée initialement dans [Hansen *et al.*, 2007] pour résoudre un problème de localisation. L'approche a permis d'obtenir des solutions approchées avec une erreur bornée à 4% sur des instances intéressantes.

### 2.3.6 Recherche à formulation variable

Dans de nombreux problèmes d'optimisation, il est possible d'avoir plusieurs formulations différentes, et il n'est pas toujours aisé de savoir si une formulation domine les autres sur toutes les instances du problème traité. La recherche à formulation variable a été introduite dans [Mladenović *et al.*, 2005], et repose sur l'idée d'utiliser plusieurs formulations disponibles, et de changer de formulation en fonction de l'état de la recherche. Cette mé-

**Algorithme 11 :** Fonction d'acceptation de mouvement avec plusieurs formulations.

```
1  Acceptation(s, s', n_f);
2  début
3      i ← 1;
4      tant que i ≤ n_f faire
5          si f_i(s') < f_i(s) alors retourner Vrai ;
6          ;
7          sinon si f_i(s') < f_i(s) alors retourner Faux ;
8          ;
9          i ← i + 1;
10     retourner Faux ;
```

**Algorithme 12 :** Pseudo-code de la recherche LNS.

```
1  LNS (s, t_max);
2  début
3      s_b ← s;
4      répéter
5          s_t ← r(d(s));
6          si accept(s_t, s) alors
7              s ← s_t;
8          si f(s_t) < c(s_b) alors
9              s_b ← s_t;
10     jusqu'à t > t_max;
11     retourner s_b ;
```

thode est intéressante si la recherche locale utilisée au sein de VNS se comporte différemment sur les différentes formulations et qu'il est facile de transformer une solution d'une formulation vers une autre.

Cette variante a été appliquée dans [Pardo *et al.*, 2013] pour résoudre un problème de découpage. La mise en place de cette méthode demande quelques ajustements de sorte à considérer les différentes formulations du problème. Les auteurs ont ajouté dans chacune des trois étapes principales de VNS (la phase de perturbation, la recherche locale et la fonction de changement de voisinage) l'utilisation de la fonction d'acceptation décrite par le pseudo-code de l'algorithme 11.

### 2.3.7 Recherche à voisinage large

La recherche à voisinage large (connue sous la terminologie anglaise de *Large Neighborhood Search* (LNS)) a été proposée par Shaw [Shaw, 1998]. Dans LNS, les voisinages sont définis implicitement par une destruction puis une reconstruction de la solution courante. Une fonction de destruction détruit une partie de la solution courante tandis qu'une méthode de reconstruction reconstruit la solution détruite. La fonction de destruction contient généralement de l'aléatoire pour que différentes parties de la solution soient détruites à chaque invocation de la fonction.

Le pseudo-code de la méthode LNS est illustré dans l'algorithme 12. L'algorithme maintient trois variables pour manipuler les solutions. La variable $s_b$ est utilisée pour maintenir la meilleure solution trouvée, la variable $s$ est utilisée pour maintenir la solution courante, tandis que $s_t$ est une solution temporaire qui peut être rejetée ou promue pour être une solution courante. La fonction $d()$ retourne une copie de $s$ partiellement détruite. Appliquer la fonction $r()$ sur la solution détruite retourne une solution faisable construite à partir de la solution détruite.

**Algorithme 13 :** Pseudo-code de l'algorithme VNS/LDS+CP.

```
1  8
2  Fonction VNS/LDS+CP (X, W, k_init, k_max, δ_max) ;
3  début
4      s ← genSolInit(X, W) ;
5      k ← k_init ;
6      tant que (k < k_max) ∧ (not TimeOut) faire
7          X_un ← Hneighborhood(N_k, X, W, s) ;
8          A ← s\{(x_i, a) | x_i ∈ X_un} ;
9          s' ← LDS + CP(A, X_un, δ_max, f(s), s) ;
10         NeighbourhoodChange(s, s'', k) ;
11     retourner s ;
```

Pour chaque itération, une fonction $accept()$ est appelée sur la solution temporaire $s_t$. Cette fonction peut être implémentée de différentes façons, la plus simple est d'accepter uniquement des solutions améliorées.

## 2.4 Extensions de VNS pour la résolution des WCSP

Nous présentons dans cette section les travaux menés autour de la recherche VNS pour la résolution des WCSP. L'utilisation du schéma VNS pour résoudre les WCSP a commencé avec VNS/LDS+CP. Cette approche est liée à LNS [Shaw, 1998], mais permet d'ajuster dynamiquement la taille du voisinage lorsque la recherche stagne dans un minimum local comme dans VNS.

### 2.4.1 VNS/LDS+CP

VNS/LDS+CP [Loudni et Boizumault, 2008] peut être considérée comme une extension de VNDS où la phase de shaking est effectuée en désaffectant $k$ variables du problème, tandis que la phase de reconstruction est réalisée grâce à une méthode arborescente partielle (LDS), combinée avec des mécanismes de propagation de contraintes (CP) sur les fonctions de coût.

L'algorithme 13 donne le pseudo-code de VNS/LDS+CP où $W$ est l'ensemble des fonctions de coût, $k_{init}$ (resp. $k_{max}$) est le nombre minimal (resp. maximal) de variables à désaffecter et $\delta_{max}$ est la valeur maximale de la discrepancy pour LDS. VNS/LDS+CP commence par la génération aléatoire d'une solution initiale $s$ (ligne 4). Un sous-ensemble de $k$ variables (avec $k$ la dimension du voisinage) est sélectionné dans le voisinage $N_k$ (i.e. l'ensemble des combinaisons de $k$ variables parmi $X$) par l'heuristique de choix de voisinage `Hneighborhood` (ligne 7). Une affectation partielle $\mathcal{A}$ est alors générée à partir de la solution courante $s$, en désaffectant les $k$ variables sélectionnées ; les autres variables (i.e. non sélectionnées) gardent leur affectation dans $s$ (ligne 8). $\mathcal{A}$ est alors reconstruite en utilisant une recherche arborescente partielle de type LDS, aidée par une propagation de contraintes (CP) sur les fonctions de coût. Si LDS trouve une solution de meilleure qualité $s'$ dans le voisinage de $s$, $s'$ devient la solution courante et $k$ est réinitialisé à $k_{init}$. Sinon, on cherche de nouvelles améliorations dans $N_{k+1}$ (structure de voisinage où $(k+1)$ variables de $X$ seront désaffectées). Ce traitement est assuré par la procédure `NeighborhoodChange` (ligne 10). L'algorithme s'arrête dès que l'on a atteint la dimension maximale des voisinages à considérer $k_{max}$ ou le $TimeOut$ (ligne 6).

**Algorithme 14 :** Pseudo-code de DGVNS.

1 `Fonction DGVNS` $(X, W, k_{init}, k_{max}, \delta_{max})$ ;
2 **début**
3 Soit $G$ le graphe de contraintes de $(X, W)$ ;
4 Soit ($\mathcal{C}_T$,$T$) une décomposition arborescente de $G$ ;
5 Soit $\mathcal{C}_T = \{C_1, C_2, ..., C_p\}$ ;
6 $s \leftarrow$ `genSolInit` $(X, W)$ ;
7 $k \leftarrow k_{init}$ ;
8 $i \leftarrow 1$ ;
9 **tant que** $(k < k_{max}) \wedge (not\, TimeOut)$ **faire**
10 $C_s \leftarrow$ `CompleteCluster` $(C_i, k)$ ;
11 $\mathcal{X}_{un} \leftarrow$ `Hneighborhood` $(N_{k,i}, C_s, W, s)$ ;
12 $\mathcal{A} \leftarrow s \backslash \{(x_i, a) \,|\, x_i \in \mathcal{X}_{un}\}$ ;
13 $s' \leftarrow LDS + CP(\mathcal{A}, \mathcal{X}_{un}, \delta_{max}, f(s), s)$ ;
14 `NeighbourhoodChangeDGVNS` $(s, s', k, i)$ ;
15 **retourner** $s$ ;

### 2.4.2 VNS guidée par la décomposition arborescente (DGVNS)

**2.4.2-$\alpha$ Structure de voisinage.** Récemment, Fontaine et al. [Fontaine, 2013] ont étudié l'exploitation d'une décomposition arborescente d'un graphe de contraintes au sein de VNS/LDS+CP afin de guider efficacement l'exploration des grands voisinages. L'intuition est d'utiliser les clusters de la décomposition arborescente afin de sélectionner les variables à désinstancier dans un même cluster. En effet, toute affectation d'une variable du cluster impacte directement l'affectation des autres variables appartenant au même cluster. Ils ont proposé DGVNS [Fontaine, 2013], une première approche de recherche locale qui exploite le graphe de clusters fourni par une décomposition arborescente pour construire des structures de voisinages pertinentes. Ainsi, à la place des structures de voisinage $N_k$ dans VNS/LDS+CP (cf. section 2.4.1), DGVNS utilise des structures de voisinage $N_{k,i}$.

**Définition 2.4.** STRUCTURE DE VOISINAGE $N_{k,i}$
Soit $G$ un graphe et $G_T$=($\mathcal{C}_T$,$E_T$) le graphe de clusters associé. Soit $C_i$ un cluster de $\mathcal{C}_T$. La structure de voisinage $N_{k,i}$ désigne l'ensemble des combinaisons de $k$ variables ($k$ étant la dimension du voisinage) parmi $C_i$.

**2.4.2-$\beta$ Algorithme de DGVNS.** L'algorithme 14 décrit le pseudo-code de DGVNS. Ainsi, pour favoriser les mouvements dans des régions fortement liées, à chaque itération, les $k$ variables à désaffecter sont sélectionnées au sein d'un même cluster $C_i$ (ou $C_i$ et ses clusters voisins dans le cas où $C_i$ a moins de $k$ variables) (ligne 10). En effet, le cluster est une structure qui permet d'identifier ces régions, du fait de sa taille (plus petite que le problème original), et du lien fort entre les variables qu'il contient. De plus, grâce à la forte connectivité entre les variables, l'étape de reconstruction pourra bénéficier d'un filtrage plus efficace et d'un calcul de minorant de bien meilleure qualité. Ensuite, un sous-ensemble de $k$ variables est sélectionné dans le voisinage $N_{k,i}$ (i.e. l'ensemble des combinaisons de $k$ variables parmi $C_s$) par la fonction `Hneighborhood` (ligne 11). Tout comme VNS/LDS+CP, une affectation partielle $\mathcal{A}$ est alors générée, puis reconstruite en utilisant une recherche arborescente partielle de type LDS, aidée par une propagation de contraintes (CP) sur les fonctions de coût (ligne 13). L'algorithme s'arrête dès que l'on a atteint la dimension maximale du voisinage à considérer $k_{max}$ ou le *TimeOut* (ligne 15).

DGVNS est paramétré par la procédure `NeighborhoodChangeDGVNS` qui détermine, à chaque itération, le prochain cluster $C_i$ (i.e. stratégie de changement de voisinage) en fonction de la qualité de la nouvelle solution $s'$ (ligne 14). Dans [Loudni *et al.*, 2013] deux stratégies qui exploitent le graphe de clusters ont été proposées. Elles correspondent à deux schémas de déplacement entre clusters en fonction de l'amélioration ou non de la qualité de la solution courante.

**i) DGVNS-1 : changer systématiquement de cluster.** L'idée principale est de considérer successivement tous les clusters $C_i$ avec un ordre aléatoire. Soit $p$ le nombre total de clusters, $succ$ une fonction successeur, qui associe à un cluster $i$ son successeur $C_{succ(i)}$ [12] et $N_{k,i}$ la structure de voisinage courante. Si aucune solution de meilleure qualité que la solution courante n'a été trouvée par `LDS+CP` dans le voisinage de $s$, DGVNS-1 cherche de nouvelles améliorations dans $N_{(k+1),succ(i)}$ (structure de voisinage où $(k+1)$ variables seront désaffectées). En effet, se déplacer du cluster $C_i$ vers le cluster $C_{succ(i)}$ permet de favoriser l'exploration de nouvelles régions de l'espace de recherche et de tenter de trouver des solutions de meilleure qualité qui peuvent se trouver dans ces régions. De plus, quand un minimum local est atteint dans le voisinage courant, passer de $k$ à $k+1$ permet de renforcer la diversification par l'élargissement de la dimension du voisinage. À l'inverse, si LDS+CP trouve une solution de meilleure qualité $s'$ dans $N_{k,i}$, $k$ est réinitialisé à $k_{init}$ et le cluster suivant est considéré : changer de cluster permettra de diversifier l'exploration autour de $s'$.

**ii) DGVNS-2 : changer de cluster en l'absence d'amélioration.** Le principe de cette heuristique est de passer au cluster $C_{succ(i)}$ si aucune amélioration de la solution courante n'a été trouvée par LDS+CP. En effet, rester dans le même cluster en cas d'amélioration de la solution courante permettra de propager, durant la phase de reconstruction, les conséquences des nouvelles affectations des variables de $s'$. De plus réinitialiser $k$ à $k_{init}$ favorisera de petits mouvements dans le voisinage de $s'$. Ainsi, et contrairement à DGVNS-1, DGVNS-2 effectue un compromis entre intensification et diversification. En effet, tant qu'aucune amélioration n'a été trouvée, DGVNS-2 considère successivement tous les clusters $C_i$ (i.e. diversification), mais dès que la solution courante est améliorée, DGVNS-2 passe à l'intensification.

Les résultats obtenus dans [Fontaine, 2013] sur différentes instances de plusieurs problèmes (RLFAP, GRAPH, SPOT5 et TagSNP) montrent clairement l'impact de la stratégie de changement de voisinage sur les performances de DGVNS. DGVNS-1 obtient un meilleur score sur le critère temps de calcul [13] (DGVNS-1 est plus rapide sur $65\%$ des instances considérées). En terme de taux de succès [14], DGVNS-2 présente de meilleures performances : DGVNS-2 obtient les meilleurs taux de succès sur 19 instances contre 13 pour DGVNS-1. Clairement, DGVNS-2 réalise un meilleur compromis entre intensification et diversification.

## 2.5 Stratégies parallèles pour VNS

### 2.5.1 Parallélisme

En optimisation combinatoire, le calcul parallèle est l'utilisation simultanée de plusieurs processus, opérant sur plusieurs processeurs, pour résoudre une instance d'un problème donné. Le parallélisme est donc une décomposition de la charge totale de calcul et la répartition des tâches résultantes sur un ensemble de processus. D'un point de vue algorithmique, la principale source de parallélisme pour les métaheuristiques est la parallélisation de leurs itérations internes, qui sont souvent implémentées en boucle séquentielle : évaluation et exploration des voisins, évaluation des solutions, etc.

Dans ce document, nous adoptons la classification faite dans [Crainic et Toulouse, 1998] pour décrire les différentes stratégies parallèles pour les métaheuristiques. Cette classification, qui porte sur trois niveaux, est suffisamment générale pour englober toutes les classes des métaheuristiques.

- **Parallélisation interne** [15]**.** Cette stratégie vise principalement à accélérer les calculs en exécutant en parallèle une ou plusieurs tâches gourmandes en temps de calcul et qui peuvent être parallélisées (par exemple,

12. si $(i < p)$ alors $succ(i) = i + 1$ sinon $succ(p) = 1$.
13. Temps de calcul moyen nécessaire à la méthode pour atteindre l'optimum sur l'instance considérée.
14. Le nombre d'essais avec succès (pour attendre l'optimum) obtenu par la méthode sur l'instance considérée.
15. Aussi appelée parallélisme de bas niveau.

évaluation des voisins) pour la même itération de la méthode. Mise en œuvre habituellement selon le modèle maître-travailleurs, la procédure séquentielle est exécutée sur un seul processeur (appelé maître), qui dispatche le travail sur les autres processeurs (appelés travailleurs), récupère et fusionne les résultats, puis poursuit l'algorithme séquentiel. Les variantes de cette approche peuvent être définies en fonction de la quantité de travail attribué.

- **Décomposition du domaine du problème.** La décomposition d'un problème réduit la taille de l'espace de recherche en divisant les variables de décision. Ainsi, une même procédure de recherche peut être appliquée en parallèle sur différentes parties du problème pour permettre une exploration rapide et complète de l'espace de recherche. Ce schéma est souvent mis en œuvre via un modèle maître-travailleurs, où le processus maître effectue la décomposition du problème et les processus travailleurs exécutent simultanément la métaheuristique sur les sous-problèmes issus de la décomposition. Le maître recueille les solutions partielles et en construit une solution complète.
- **Plusieurs processus de recherche.** Le parallélisme est obtenu par une exploration concurrente de l'espace de recherche. Ces explorations peuvent ou non exécuter les mêmes méthodes heuristiques, commencer à partir de solutions initiales différentes, etc. Elle peuvent impliquer différents degrés de synchronisation (synchrone vs. asynchrone ) et de coopération (communiquer pendant la recherche ou seulement à la fin pour identifier la meilleure solution globale).

### 2.5.2 Parallélisation de VNS

L'utilité de la parallélisation de VNS a été démontrée dans les travaux de [García-López *et al.*, 2002] et [Crainic *et al.*, 2004]. Dans ces travaux, les auteurs proposent quatre algorithmes de parallélisation de VNS.

### 2.5.3 Recherche VNS parallèle synchrone

Le premier algorithme propose de paralléliser la partie recherche locale de VNS. L'algorithme 15 illustre le pseudo-code de cette stratégie avec $npr$ processus. Un processus maître exécute une phase de shaking à partir de la solution initiale, puis lance $npr$ processus travailleurs qui vont se voir assigner un sous-ensemble des variables du problème sur lequel ils réaliseront la recherche locale (ligne 7). Chaque processus explore une sous-partie distincte de l'espace de recherche (notée $I_r$) et renvoie la meilleure solution trouvée (lignes 10-11). Les solutions trouvées sont alors passées au maître qui en sélectionne la meilleure (lignes 12-13) pour remplacer la solution courante (ligne 15). L'intérêt d'une telle approche est de diviser le coût d'exploration par le nombre de processus disponibles.

### 2.5.4 Recherche VNS parallèle répliquée (RPVNS)

Le second algorithme, appelé RPVNS, propose de mettre en parallèle plusieurs instances de VNS initialisées avec une solution de départ différente pour chaque processus. Les auteurs cherchent ici à augmenter l'espace de recherhe visité en partant de solutions initiales variées. A la fin de l'exécution des processus travailleurs, la meilleure solution est retournée. L'algorithme 16 présente le pseudo-code de RPVNS.

### 2.5.5 Recherche VNS parallèle avec phase de shaking répliquée (RSVNS)

Le troisième algorithme, appelé RSVNS, consiste en un unique processus VNS (dit maître) qui lance en parallèle plusieurs phases de shaking/recherche locale. Chaque processus travailleur va ensuite reporter sa meilleure solution au maître qui sera utilisée pour améliorer l'intensification de la recherche. L'algorithme 17 décrit le pseudo-code de cette stratégie.

**Algorithme 15 :** Pseudo-code de la recherche parallèle SPVNS.

```
1  Fonction SPVNS (npr, k_max, t_max);
2  début
3      k ← 1;
4      s ← InitSol();
5      répéter
6          s' ← Shake(s,N_k) ;
7          Diviser l'espace de recherche à explorer I en npr sous-ensembles I_r ;
8          répéter
9              s'' ← s' ;
10             pour tous les travailleurs en parallèle r = 1, ..., npr faire
11                 s_r ← Best(s'', I_r) ;
12             pour tous les r = 1, ..., npr faire
13                 si f(s_r) < f(s') alors s' ← s_r;
14         jusqu'à f(s') = f(s'');
15         si f(s'') < f(s') alors s ← s'', k ← 1;
16         sinon k ← k + 1;
17     jusqu'à k > k_max;
18     retourner s ;
```

RPVNS et RSVNS correspondent à des parallélisations naturelles permettant d'hybrider recherche VNS et recherche multi-démarrage. RPVNS peut être vue comme une recherche multi-démarrage où les recherches locales sont remplacées par des VNS et la seconde consiste en un VNS où la recherche locale unique dans le $k^{ieme}$ voisinage est remplacée par une recherche avec démarrages multiples (à partir de shaking différents).

### 2.5.6 Recherche VNS parallèle à voisinages coopératifs (CNVNS)

Le dernier algorithme, appelé CNVNS [Crainic *et al.*, 2004] (pour Cooperative Neighborhood VNS), implémente un mécanisme de coopération qui permet à chaque itération de la recherche d'accéder de manière individuelle à la meilleure solution globale courante sans interrompre ses processus courants. Les processus travailleurs communiquent exclusivement avec un processus central, appelé maître qui lance plusieurs processus à partir de solutions initiales différentes. Chaque processus travailleur réalise un VNS classique tout en reportant au maître leur meilleur résultat à chaque amélioration. En cas d'échec, il demande au maître la meilleure solution courante.

Les différents travaux décrits ci-dessus présentent trois facettes de parallélisation de VNS :
- parallélisation de l'exploration du voisinage ;
- différentes recherches à partir de solutions initiales différentes ;
- différentes explorations à partir de shaking différents.

Les stratégies basées sur le multi-démarrage (solution initiale/shaking différents) sont de loin les plus prometteuses [Crainic *et al.*, 2004, García-López *et al.*, 2002]. Nous allons montrer dans le chapitre 5 comment exploiter ces travaux pour paralléliser la recherche DGVNS.

## 2.6 Évaluation des métaheuristiques

Dans cette section, nous détaillons les différents critères que nous avons retenus pour évaluer et comparer les performances des méthodes étudiées, nous utilisons les mêmes critères présentés dans [Fontaine, 2013].

**Algorithme 16 :** Pseudo-code de la recherche parallèle RPVNS.

```
1  Fonction RPVNS (npr, k_max, t_max);
2  début
3      pour tous les travailleurs en parallèle r = 1, ..., npr faire
4          s_r ← InitSol() ;
5          k_r ← 1;
6          répéter
7              s'_r ← Shake(s_r, N_{k_r}) ;
8              s''_r ← RechercheLocale(s'_r) ;
9              si f(s''_r) < f(s_r) alors s_r ← s''_r, k_r ← 1;
10             sinon k_r ← k_r + 1;
11         jusqu'à k_r > k_max;
12     s_b ← s_1;
13     pour r = 2, ..., npr faire
14         si f(s_r) < f(s_b) alors s_b ← s_r;
15     retourner s_b ;
```

**Algorithme 17 :** Pseudo-code de la recherche parallèle RSVNS.

```
1  Fonction RSVNS (npr, k_max, t_max);
2  début
3      s ← InitSol() ;
4      k ← 1;
5      répéter
6          pour tous les travailleurs en parallèle r = 1, ..., npr faire
7              s'_r ← Shake(s, N_k) ;
8              s''_r ← RechercheLocale(s'_r) ;
9          pour r = 1, ..., npr faire
10             si f(s''_r) < f(s'') alors s'' ← s''_r;
11         si f(s'') < f(s) alors s ← s'', k ← 1;
12         sinon k ← k + 1;
13     jusqu'à k > k_max;
```

### 2.6.1 Temps CPU et taux de réussite

Pour tenir compte du caractère stochastique des méthodes testées, chaque instance sera résolue 50 fois par chaque méthode. Ainsi, pour chaque instance et chaque méthode, nous reportons :

— le nombre d'essais avec succès (c'est-à-dire ayant atteint l'optimum) ;
— le temps de calcul moyen pour atteindre l'optimum ;
— le coût moyen des solutions trouvées sur les 50 essais.

Le taux de réussite d'une méthode par rapport à une instance est défini par le rapport entre le nombre d'essais avec succès et le nombre total d'essais. Le taux de réussite permet de mesurer la capacité de la méthode à atteindre le plus souvent l'optimum quand celui-ci est connu ou la meilleure solution connue. Il donne aussi une indication sur la robustesse de la méthode pour atteindre le même résultat avec des paramètres d'initialisation différents sur l'instance considérée.

### 2.6.2 Profil de performance

Une méthode de résolution doit être en mesure de produire des solutions intermédiaires dont la qualité s'améliore graduellement au cours du temps. Cette information sur l'évolution de la qualité en fonction du temps de calcul alloué est appelée *Profil de performance* [Boddy et Dean, 1994].

**Définition 2.5. PROFIL DE PERFORMANCE**
Un profil de performance d'une méthode de résolution est la courbe représentant l'évolution de la qualité de la meilleure solution produite en fonction du temps de calcul $t$.

Le profil de performance d'un algorithme fournit une information sur sa capacité à offrir un réel compromis qualité/temps plus précis que la simple information "meilleure solution trouvée au bout d'un certain temps de calcul".

Pour nos comparaisons, nous considérons l'évolution en moyenne de la qualité des résultats intermédiaires en fonction du temps de calcul alloué. Ce type de profil est appelé *profil de performance moyen*.

**Définition 2.6. PROFIL DE PERFORMANCE MOYEN**
Le profil de performance moyen d'un algorithme est la courbe représentant l'évolution de la qualité de la meilleure solution produite, en moyenne, en fonction du temps de calcul $t$.

### 2.6.3 Profils de rapport de performance

Dans la seconde partie de ce mémoire, nous serons amenés à comparer les performances des méthodes que nous proposons, que ce soit entre elles, ou avec des méthodes existantes.

Pour cela, on considère les travaux de [Dolan et Moré, 2002] concernant les profils de rapport de performance. On souhaite comparer les performances d'un ensemble d'algorithmes $A = \{a_1, \ldots, a_m\}$ pour résoudre un ensemble de problèmes $P = \{p_1, \ldots, p_n\}$. Deux mesures de performance sont retenues : le temps de calcul moyen et le nombre d'essais avec succès (noté succès) (cf. remarque 2.1).

La grandeur $t(p_i, a_j)$ désigne le temps de calcul moyen nécessaire à l'algorithme $a_j$ pour atteindre l'optimum du problème $p_i$. Dans le calcul de la moyenne, le temps reporté pour les essais sans succès (optimum non atteint) est la valeur du $TimeOut$.

Le rapport de performance $r(p_i, a_j)$ permet de comparer la performance de l'algorithme $a_j$ sur le problème $p_i$ avec la meilleure performance obtenue par les autres algorithmes sur ce même problème. Ce rapport est défini par [Dolan et Moré, 2002] :

$$r(p_i, a_j) \quad = \quad \frac{t(p_i, a_j)}{\min\{t(p_i, a_k) \mid a_k \in A\}}. \tag{2.1}$$

Quels que soient l'algorithme $a_i$ et le problème $p_j$, on a toujours $r(p_i, a_j) \geq 1$. Si $r(p_i, a_j)$=1 alors $a_j$ est considéré comme étant le meilleur algorithme pour résoudre le problème $p_i$. L'algorithme $a_j$ peut être seul ou bien ex aequo avec un ou plusieurs autres algorithmes. Enfin, soit $\tau_{max}$ le maximum de tous les $r(p_i, a_j)$.

Il devient alors possible d'évaluer les performances d'un algorithme $a_j$ sur l'ensemble de problèmes $P$ grâce à la fonction $\rho_{a_j} : [1..\tau_{max}] \mapsto [0..1]$ définie par :

$$\rho_{a_j}(\tau) \quad = \quad \frac{1}{|P|}|\{p_i \in P : r(p_i, a_j) \leq \tau\}|. \tag{2.2}$$

Pour un algorithme $a_j$ fixé, $\rho_{a_j}(\tau)$ indique la proportion de problèmes $p_i \in P$ ayant un rapport de performance inférieur à $\tau$. Pour deux algorithmes $a_i$ et $a_j$, si $\rho_{a_i}(\tau) > \rho_{a_j}(\tau)$, alors, pour un facteur $\tau$, la proportion de problèmes résolus par $a_i$ est plus grande que celle de $a_j$ .

Dans la suite de ce document, nous désignerons par *profil de performance des rapports de temps d'un algorithme* $a_j$, la courbe associée à la fonction $\rho_{a_j}$. Nous noterons $\tau_j^*$ la valeur de $\tau$ telle que $\rho_{a_j}(\tau_j^*) = 1$ et $\tau_{min}$ la valeur minimale de $\tau_j^*$ parmi tous les algorithmes de $A$.

Plusieurs critères permettant de comparer les différents algorithmes peuvent être déduits de l'équation 2.2 :

1. $\rho_{a_j}(1)$ indique la proportion de problèmes de $P$ où l'algorithme a été le plus performant : plus la valeur de $\rho_{a_j}(1)$ est élevée, meilleur est $a_j$.
2. L'aire sous la courbe $\rho_{a_j}$ (notée $AUC_{a_j} = \int \rho_{a_j}(\tau)\,\mathrm{d}t$) donne un indicateur de la performance globale de $a_j$ sur les problèmes de $P$ : plus l'aire est grande, plus l'algorithme $a_j$ est efficace.
3. Le ratio $\tau_{min}/\tau_j^*$ permet de mesurer la robustesse d'un algorithme : l'algorithme le plus robuste est celui ayant un ratio de 1.

**Remarque 2.0.**
Dans l'équation 2.1, permettant d'évaluer le rapport de performance d'un algorithme, outre la mesure du temps de calcul moyen passé par l'algorithme $a_j$ pour atteindre l'optimum sur le problème $p_i$, nous considérons également le nombre d'essais avec succès obtenu par l'algorithme $a_j$ sur le problème $p_i$. Toutefois, pour générer la courbe $\rho_{a_j}(\tau)$ mesurant l'évolution du rapport de performance du nombre d'essais avec succès de l'algorithme $a_j$ en fonction de $\tau$, il est nécessaire de considérer l'inverse du nombre d'essais avec succès dans l'équation 2.1.

## 2.7 Conclusion sur les métaheuristiques

Dans ce chapitre, nous avons dressé un panorama non exhaustif des métaheuristiques proposées dans la littérature, nous nous sommes focalisés principalement sur les méthodes ayant un lien avec nos contributions. Dans la suite de ce document, nous nous intéresserons particulièrement à DGVNS. Cette méthode, basée sur de grands voisinages, permet en effet de traiter efficacement des problèmes de grande taille en profitant de la décomposition arborescente de l'instance. De plus, DGVNS permet de tirer parti des mécanismes de filtrage liés aux WCSP. Pour nos expérimentations, nous utiliserons EDAC, qui fournit le meilleur compromis entre la qualité de filtrage obtenu et la complexité temporelle de son calcul.

Pour aborder le deuxième volet de la thèse, nous présentons dans le chapitre suivant l'état de l'art sur l'extraction des motifs locaux sous contraintes et les ensembles de motifs.

# Chapitre 3

# Motifs ensemblistes et ensembles de motifs

## Sommaire


**3.1 Théorie d'une base transactionnelle . . . . . 46**
3.1.1 Contexte transactionnel . . . . . 46
3.1.2 Extraction sous contraintes . . . . . 47
**3.2 Extraction de motifs sous contraintes locales . . . . . 47**
3.2.1 Contraintes de base . . . . . 47
3.2.2 Élagage de l'espace de recherche . . . . . 49
3.2.3 Représentations condensées . . . . . 50
3.2.4 LCM : Extraction des motifs fermés en temps linéaire . . . . . 54
**3.3 Extraction d'ensembles de motifs . . . . . 55**
3.3.1 Définitions de base . . . . . 55
3.3.2 Contraintes n-aires . . . . . 56
**3.4 Problèmes de l'extraction d'ensembles de motifs . . . . . 58**
3.4.1 Clustering conceptuel . . . . . 58
3.4.2 Problème du tuilage . . . . . 59
3.4.3 Algorithmes d'extraction . . . . . 60
**3.5 Bilan . . . . . 61**


La résolution des requêtes inductives pour la découverte des ensembles de motifs satisfaisant un prédicat donné a été étudiée de manière approfondie en fouille de données. Le problème d'extraction des motifs fréquents sur de grandes bases de données a donné naissance à de nombreux algorithmes dédiés et efficaces. Le développement de ces algorithmes a été poussé grâce au concept des représentations condensées basées sur la fréquence. Les motifs fréquents sont très utiles dans de nombreuses tâches en fouille de données, plus particulièrement, dans des tâches de clustering, d'extraction des règles d'association, et de classification par association.

Nous présentons dans ce chapitre les concepts de base liés à l'extraction de motifs locaux et d'ensembles de motifs. Nous introduisons, dans la section 3.1, le contexte des bases de données transactionnelles. Nous présentons ensuite, dans la section 3.2, les concepts et les définitions de base liés aux tâches d'extraction des motifs locaux sous contraintes. Nous introduisons, dans la section 3.3, le problème de l'extraction d'ensembles de motifs et les contraintes de type n-aire. De plus, nous rapportons deux applications importantes, une sur le clustering conceptuel, et l'autre sur le problème de tuilage dans la section 3.4. Nous terminons enfin le chapitre par une discussion de certains points de l'état de l'art.

## 3.1 Théorie d'une base transactionnelle

### 3.1.1 Contexte transactionnel

Nous présentons dans cette section les définitions de base relatives au cadre formel introduit dans [Mannila et Toivonen, 1996] pour la découverte de motifs sous contraintes dans une base de données. Nous nous intéressons particulièrement aux bases de données transactionnelles.

Soit $\mathcal{D}$ une base de données composée d'un ensemble de transactions $\mathcal{T}$ décrites par un ensemble $\mathcal{I}$ de littéraux distincts appelés items, un *motif ensembliste d'items* (aussi appelé *itemset*) correspond à un sous-ensemble non vide de $\mathcal{I}$.

**Définition 3.1.** CONTEXTE TRANSACTIONNEL
Une base de données $\mathcal{D}$ est décrite par un triplet $(\mathcal{I}, \mathcal{R}, \mathcal{T})$ où $\mathcal{I} = \{i_1, ..., i_n\}$ est l'ensemble des items (attributs), $\mathcal{T} = \{t_1, ..., t_m\}$ celui des transactions (objets) et $\mathcal{R}$ une relation binaire entre $\mathcal{I}$ et $\mathcal{T}$. $\mathcal{R}$ indique quels items $i$ sont présents dans les transactions $t$ et on notera de façon équivalente $\mathcal{R}(i,t) = 1$ (présent) pour $i\mathcal{R}t$, et $\mathcal{R}(i,t) = 0$ (absent) pour $\neg i\mathcal{R}t$.

Un contexte transactionnel [16] $(\mathcal{I}, \mathcal{R}, \mathcal{T})$ peut être vu comme une matrice binaire. Le tableau 3.1 représente une base de transactions $\mathcal{T}$ où 11 transactions étiquetées $t_1, \ldots, t_{11}$ sont décrites par 8 items $A, B, \ldots, H$.

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

TABLE 3.1 – Représentation d'une base de données transactionnelle $\mathcal{T}$ .

Un motif ensembliste traduit un extrait de la base de transactions $\mathcal{T}$. Il décrit un comportement ou rend compte d'un phénomène ou des régularités observées dans certaines parties des données [Hand, 2002]. Dans un contexte transactionnel, on distingue deux types de motifs :

**Définition 3.2.** MOTIF

— **Un motif ensembliste d'items ou itemset** est un sous ensemble de $\mathcal{I}$.
— **Un motif ensembliste de transactions** est un sous ensemble de $\mathcal{T}$.

Un motif constitué de $k$ items est appelé un k-motif.

16. En analyse de concepts formel, on retrouve la même définition qui est appelée par un contexte formel.

Pour un ensemble d'items spécifiés $\mathcal{I}$, le langage regroupe tous les multi-ensembles possibles de $\mathcal{L}_{\mathcal{I}}$.

**Définition 3.3. LANGAGE**
L'espace de recherche des motifs ensemblistes d'items est défini par le langage $\mathcal{L}_{\mathcal{I}}$, construit sur l'alphabet $\mathcal{I} = \{i_1, i_2, ..., i_n\}$, et constitué de tous les items de $\mathcal{I}$. C'est aussi l'ensemble des parties de $\mathcal{I}$ qui correspondent exactement à tous les sous-ensembles non vides [17] de $\mathcal{I}$. Ces sous-ensembles forment le langage $\mathcal{L}_{\mathcal{I}} = 2^{\mathcal{I}} \backslash \emptyset$.

Dans ce document, nous nous intéressons aux motifs ensemblistes. Nous nous intéressons particulièrement aux motifs ensemblistes d'items. Dans la suite de ce document, nous utilisons le terme "motif" pour désigner un motif ensembliste d'items $\phi$. Ainsi, pour décrire un motif ou une transaction dans nos exemples, les items seront décrits par des lettres en majuscules $(A, B, \dots)$.

### 3.1.2 Extraction sous contraintes

En pratique, l'expert porte souvent son intérêt sur un sous-ensemble de tous les motifs extraits ayant des caractéristiques un peu particulières. Généralement, ces caractéristiques sont décrites par des contraintes permettant de délimiter l'ensemble des motifs extraits. Ainsi, il est possible de cibler les connaissances désirées par l'expert tout en réduisant leur nombre, rendant leur interprétation une tâche plus aisée. De plus, certaines contraintes permettent d'élaguer considérablement l'espace de recherche grâce à certaines propriétés (cf. sections 3.2.2b et 3.2.2c) offrant une dimension algorithmique importante pour l'extraction des motifs sous contraintes.

**Définition 3.4. CONTRAINTE DE MOTIFS**
Une contrainte $q(\mathcal{T}, \phi)$ est un prédicat défini sur un langage, relativement à un ensemble de transactions $\mathcal{T}$ et un motif $\phi$.

Dans le but de décrire avec plus de précisions les motifs recherchés, plusieurs contraintes peuvent être combinées, formant ainsi une *requête d'extraction* de motifs.

**Définition 3.5. REQUÊTE D'EXTRACTION**
Une requête est une conjonction ou une disjonction de contraintes des motifs recherchés.

L'extraction de motifs sous contraintes consiste à extraire d'une base $\mathcal{D}$ décrite par des éléments du langage $\mathcal{L}_{\mathcal{I}}$ les motifs satisfaisant une contrainte $q$. Plus formellement, il s'agit de déterminer la théorie correspondante (l'ensemble des motifs satisfaisant $q$ dans $\mathcal{D}$) [Mannila et Toivonen, 1996] :

**Définition 3.6. THÉORIE**
Pour un langage donné $\mathcal{L}_{\mathcal{I}}$, une base de données $\mathcal{D}$ et une contrainte $q$, la théorie $\mathcal{T}h(\mathcal{L}_{\mathcal{I}}, \mathcal{D}, q)$ est l'ensemble des motifs de $\mathcal{L}_{\mathcal{I}}$ satisfaisant la contrainte $q$ dans $\mathcal{D}$.

Nous nous focalisons, dans ce document, sur les bases de données transactionnelles, par conséquent, nous adoptons $\mathcal{T}$ pour toutes les formulations des différentes mesures et contraintes.

## 3.2 Extraction de motifs sous contraintes locales

### 3.2.1 Contraintes de base

Un motif local est un motif qui satisfait des contraintes locales unaires. Il existe de nombreuses contraintes permettant d'évaluer la pertinence et la qualité des motifs locaux, comme par exemple les contraintes exprimées à

17. L'ensemble vide est exclus car il est rarement porteur du sens.

l'aide de mesures définissant l'intérêt des motifs extraits [Geng et Hamilton, 2006]. Le nombre d'occurrences d'un motif dans la base de données, défini par la mesure de sa fréquence, est l'une des mesures les plus classiques. La contrainte de fréquence découlant de cette mesure (définition 3.8) servira d'exemple pour présenter les différentes notions dans la suite de ce chapitre. D'autres critères sont utiles pour construire des motifs intéressant l'utilisateur. Par exemple, supposons que l'utilisateur recherche des motifs suffisamment fréquents et longs, cela pourrait être achevé en contraignant que l'aire des motifs doit être supérieure à un certain seuil (cf. définition 3.10).

Les notions de support et de fréquence d'un motif sont définies comme suit :

**Définition 3.7. SUPPORT**
Une transaction $t \in \mathcal{T}$ supporte le motif $\phi$ (ou $\phi$ est présent dans $t$) ssi $\phi \subseteq t$ (ou $\forall i \in \phi, R(i,t) = 1$). Le **support** d'un motif $\phi$, noté $cover(\phi)$, est l'ensemble des transactions qui le supportent.

**Définition 3.8. FRÉQUENCE**
La **fréquence** d'un motif $\phi = (I, T)$, où $I$ est l'ensemble d'items ($I \subseteq \mathcal{I}$) constituant le motif $\phi$, et $T$ est l'ensemble de transactions ($T \subseteq \mathcal{T}$) supportant le motif $\phi$, est le cardinal du support :

$$freq(\phi, \mathcal{T}) = |T|.$$

**Définition 3.9. TAILLE, LONGUEUR**
La **taille** d'un motif $\phi = (I, T)$ est le nombre d'items constituant le motif $\phi$ :

$$taille(\phi, \mathcal{T}) = |I|.$$

**Définition 3.10. AIRE**
Soit $\phi = (I, T)$ un motif. L'**aire** du motif $\phi$ est le produit de sa taille par sa fréquence :

$$aire(\phi, \mathcal{T}) = |I| \times |T|.$$

**Exemple 3.1.**
La requête suivante est un exemple d'extraction des motifs locaux, constituée de la conjonction de contraintes locales sur les mesures de fréquence et de taille suivantes :

$$freq(\phi, \mathcal{T}) \geq 3 \wedge taille(\phi, \mathcal{T}) \geq 2.$$

Cette requête stipule que la fréquence d'un motif $\phi$, $freq(\phi, \mathcal{T})$, doit être supérieure ou égale à 3 et que sa taille, $taille(\phi, \mathcal{T})$, doit être supérieure ou égale à 2. L'ensemble des solutions répondant à cette requête pour notre exemple du tableau 3.1 est $\{C, E\}$, $\{C, G\}$, $\{E, G\}$, $\{A, E\}$ et $\{C, H\}$.

**Définition 3.11. MOTIF FRÉQUENT**
Un motif est fréquent (ou $min_{freq}$-fréquent) si et seulement si sa fréquence est supérieure ou égale à un seuil de fréquence minimale $min_{freq}$ fixé par l'utilisateur. On note $\mathcal{F}(\mathcal{T}, min_{freq})$ l'ensemble des motifs fréquents dans la base de transactions $\mathcal{T}$ pour une fréquence minimale $min_{freq}$.

**Exemple 3.2.**
Dans notre exemple du tableau 3.1, les motifs $\{A\}$, $\{B\}$, $\{C\}$, $\{E\}$, $\{G\}$, $\{C, E\}$, $\{C, G\}$, $\{E, G\}$, $\{A, E\}$ et $\{C, H\}$ sont 3-fréquents.

L'extraction des motifs fréquents est considérée comme une méthode non supervisée permettant l'identification des corrélations entre les données, telle que la construction des *règles d'association*.

**Définition 3.12. MOTIF FERMÉ**
Un motif $\phi = (I, T)$ est dit fermé dans $\mathcal{T}$ si et seulement si aucun sur-ensemble contenant $\phi$ n'a le même support que $\phi$ dans $\mathcal{T}$. On note $closed(\phi)$ l'unique sur-ensemble de $\phi$ ayant le même support que $\phi$, un motif fermé est égal à sa propre fermeture.

**Définition 3.13. MOTIF FERMÉ FRÉQUENT**
Un motif $\phi$ est dit fermé fréquent dans $\mathcal{T}$ si et seulement si $\phi$ est un motif fermé et que sa fréquence $freq(\phi, \mathcal{T})$ est supérieure ou égale à un certain seuil $min_{freq}$.

À partir de l'ensemble des motifs fermés fréquents, noté $\mathcal{F}_{clos}$, et de leurs fréquences, nous pouvons générer tous les motifs fréquents avec leurs fréquences exactes $\mathcal{F}(\mathcal{T}, min_{freq})$. Étant donné un motif $\phi$, si $\phi$ n'a pas un sur-ensemble dans $\mathcal{F}_{clos}$, alors $closed(\phi)$ n'est pas fréquent, et donc, $\phi$ ne peut pas être un motif fréquent. Si $\phi$ a au moins un sur-ensemble dans $\mathcal{F}_{clos}$, alors $freq(\phi, \mathcal{T}) = freq(\psi, \mathcal{T})$, où $\psi = closed(\phi)$ est le plus petit sur-ensemble de $\phi$ dans $\mathcal{F}_{clos}$.

### 3.2.2 Élagage de l'espace de recherche

#### 3.2.2a Espace de recherche

La formulation du langage pour l'extraction de motifs est simple. Toutefois, sa résolution est une tâche algorithmiquement difficile car l'espace de recherche des motifs est très grand : si $|\mathcal{I}| = n$, alors la taille de l'espace de recherche est $\mathcal{O}(2^n)$. Pour les motifs ensemblistes, l'espace $\mathcal{L}_{\mathcal{I}}$ dans lequel les motifs sont recherchés correspond à un treillis.

**Définition 3.14. SPÉCIALISATION - GÉNÉRALISATION**
Une relation de spécialisation, notée $\preceq$, est un ordre partiel défini sur les motifs de $\mathcal{L}_{\mathcal{I}}$. Un motif $\phi$ est plus spécifique qu'un motif $\psi$ (resp. plus général) si $\psi \preceq \phi$ (resp. $\phi \preceq \psi$).

L'inclusion entre deux motifs définit une relation de **spécialisation** [Mitchell, 1982]. Ainsi, si un motif $\phi$ est inclus dans un motif $\psi$, on dit que $\psi$ est plus spécifique que $\phi$ ou que $\phi$ est plus général que $\psi$. Par exemple, le motif $\{C, E\}$ est inclus dans le motif $\{C, E, G\}$.

L'idée maîtresse des travaux qui traitent la découverte de motifs locaux sous contraintes est d'exploiter les propriétés de monotonie/anti-monotonie de certaines contraintes afin d'élaguer l'espace de recherche.

#### 3.2.2b Anti-monotonie

**Définition 3.15. ANTI-MONOTONIE**
Une contrainte $q(\phi, \mathcal{T})$ est anti-monotone suivant la spécialisation si et seulement si :

$$\forall \phi, \psi \in \mathcal{L}_{\mathcal{I}},\ \phi \subseteq \psi \Rightarrow (q(\psi, \mathcal{T}) \Rightarrow q(\phi, \mathcal{T})).$$

$freq(\phi) \geq min_{freq}$ est un exemple de contrainte anti-monotone.

#### 3.2.2c Monotonie

**Définition 3.16. MONOTONIE**
Une contrainte $q(\phi, \mathcal{T})$ portant sur un motif $\phi$ de $\mathcal{L}_\mathcal{I}$ relativement à une base de données $\mathcal{T}$, est monotone suivant la spécialisation si et seulement si :

$$\forall \phi, \psi \in \mathcal{L}_\mathcal{I},\ \phi \subseteq \psi \Rightarrow (q(\phi, \mathcal{T}) \Rightarrow q(\psi, \mathcal{T})).$$

$taille(\phi) \geq min_{taille}$ est un exemple de contrainte monotone.

L'(anti-)monotonie conduit à une importante propriété d'élagage. Si un motif $\phi$ ne satisfait pas une contrainte anti-monotone (respectivement monotone) $q$, alors toutes ses spécialisations (respectivement généralisations) ne la satisfont pas.

#### 3.2.2d Bordure positive et négative

Étant donné une contrainte anti-monotone, les bordures [Mitchell, 1982, Mannila et Toivonen, 1996] partitionnent l'espace de recherche en deux sous-espaces : l'espace des motifs satisfaisant la contrainte et l'espace des motifs ne satisfaisant pas la contrainte.

**Définition 3.17. BORDURE POSITIVE - MOTIFS MAXIMAUX**
Soit $S$ un ensemble de motifs de $\mathcal{L}_\mathcal{I}$ satisfaisant une contrainte anti-monotone $q$. La bordure positive de $S$, notée $\mathcal{B}d^+(S)$ est l'ensemble des motifs $\phi$ de $\mathcal{L}_\mathcal{I}$ tels que :

$$\phi \in \mathcal{B}d^+(S) \quad \text{ssi} \quad q(\phi, \mathcal{T}) \wedge (\forall \psi, \phi \subsetneq \psi \Rightarrow \neg q(\psi, \mathcal{T})).$$

$\mathcal{B}d^+(S)$ rassemble les motifs maximaux de S.

De manière duale, on définit la bordure négative.

**Définition 3.18. BORDURE NÉGATIVE - MOTIFS MINIMAUX**
Soit $S$ un ensemble de motifs de $\mathcal{L}_\mathcal{I}$ satisfaisant une contrainte anti-monotone $q$. La bordure négative de $S$, notée $\mathcal{B}d^-(S)$ est l'ensemble des motifs $\phi$ de $\mathcal{L}_\mathcal{I}$ tels que :

$$\phi \in \mathcal{B}d^-(S) \quad \text{ssi} \quad \phi \in \mathcal{L}_\mathcal{I} \setminus S \ et\ \forall \psi \subsetneq \phi,\ q(\psi, \mathcal{T}).$$

$\mathcal{B}d^-(S)$ rassemble les motifs minimaux de $\mathcal{L}_\mathcal{I} \setminus S$.

### 3.2.3 Représentations condensées

Les représentations condensées sont un concept très utile pour la découverte de connaissances dans des grandes bases de données, plus particulièrement, dans le cadre des bases de données inductives, qui manipulent et extraient des motifs plutôt que des données. Le problème d'extraction des motifs fréquents a suscité l'intérêt des chercheurs depuis son introduction par [Agrawal *et al.*, 1993], et différents algorithmes ont été introduits pour ce problème. Cependant, le calcul d'une théorie $Th(\mathcal{L}, \mathcal{T}, p)$ pourrait être extrêmement coûteux dans beaucoup d'applications réelles. De plus, leur nombre ingérable dans certaines situations constitue un inconvénient majeur. Pour remédier à cet inconvénient, de nombreux travaux sur l'extraction des motifs condensés ont été menés dont l'objectif est de

manipuler des ensembles de taille réduite représentant l'ensemble des motifs fréquents tout entier. Ces ensembles sont souvent appelés représentations condensées de motifs fréquents.

L'origine des représentations condensées vient des représentations $\varepsilon$-adéquates introduites par [Mannila et Toivonen, 1996]. Ce sont des représentations alternatives de données qui permettent de répondre à une classe de requêtes avec une précision limitée (e.g. une requête qui limite la fréquence des motifs dans une base de transactions). Pour une valeur donnée $\varepsilon$, nous pouvons nous intéresser aux représentations plus petites qui sont alors appelées représentations condensées. De ce fait, une représentation condensée de $Th(\mathcal{L}, \mathcal{T}, p)$ est une collection $C \subset (\mathcal{L}, \mathcal{T}, p)$ telle que tout schéma de $Th(\mathcal{L}, \mathcal{T}, p)$ pourrait être dérivé efficacement de $C$ (voir figure 3.1). En effet, les représentations condensées jouent un rôle important dans les requêtes inductives, le principal but est de minimiser la redondance entre les motifs fournis par un algorithme d'extraction, et la possibilité de dériver efficacement des motifs, vus comme redondants, des motifs trouvés par la représentation condensée (étapes (b) et (c) de la figure 3.1). Un motif est redondant si sa présence dans l'ensemble des solutions peut être dérivée par d'autres motifs fréquents. De ce fait, la taille des représentations condensées par rapport à la fréquence pourrait être, en plusieurs ordres de grandeur, plus petite que la taille des motifs fréquents.

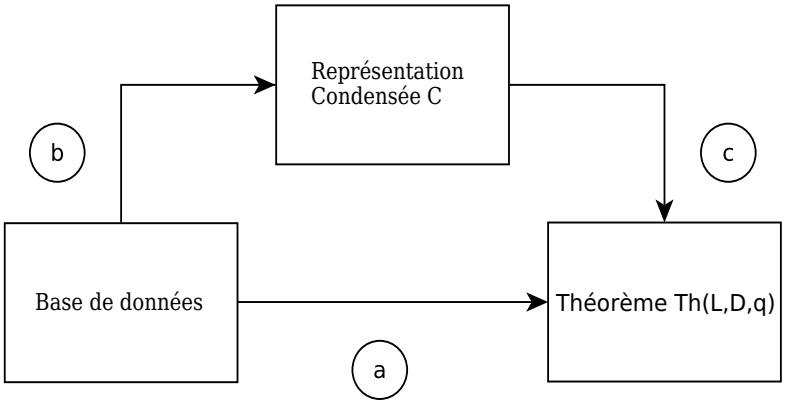



FIGURE 3.1 – Plusieurs applications utilisant des représentations condensées existent pour lesquelles les phases $b$ et $c$ sont beaucoup moins coûteuses que la phase $a$.

Il existe deux types de représentations condensées : les représentations dites exactes et celles dites approximatives. Avec une représentation condensée exacte, il est possible de régénérer la valeur exacte de la fréquence de chaque motif, alors que celle-ci est connue seulement avec une approximation (borne) dans le cas des représentations approximatives.

**Définition 3.19. REPRÉSENTATION CONDENSÉE**
La représentation condensée d'un ensemble de motifs $\mathcal{F}$ selon une contrainte $q$ est un ensemble de motifs $\mathcal{F}_{con}$ de cardinalité inférieure à celle de $\mathcal{F}$ tel que $\mathcal{F}_{con} \subseteq \mathcal{F}$ et tous les éléments de $\mathcal{F} \setminus \mathcal{F}_{con}$ peuvent être générés de manière efficace à partir de $\mathcal{F}_{con}$ sans aucun accès à la base de données.

Dans le cadre des motifs fréquents, de nombreuses représentations condensées ont été proposées ces deux dernières décennies. Nous citons, par exemple, les représentations fermées (ou closes) [Stumme *et al.*, 2000, Pasquier *et al.*, 1999, Zaki et Hsiao, 2002, Pei *et al.*, 2000], les représentations par motifs maximaux [Bayardo, 1998, Burdick *et al.*, 2001, Lin et Kedem, 1998, Zaki *et al.*, 1997], les représentations par ensembles libres [Boulicaut *et al.*, 2003], les représentations par motifs non-dérivables [Calders et Goethals, 2002, Calders et Goethals, 2007], etc.

Nous détaillons dans les sections suivantes les représentations condensées les plus répandues et les plus utilisées.

#### 3.2.3a Représentation par motifs maximaux

La représentation par motifs maximaux est composée des motifs formant la bordure positive (cf. définition 3.17). C'est une représentation approximative puisqu'elle ne permet pas de calculer précisément la fréquence des motifs mais de dériver une borne inférieure de celle-ci.

**Exemple 3.3.**
Considérons la base de données transactionnelle $\mathcal{T}$ représentée dans la table 3.1 et soit $min_{freq} = 3$ la valeur de la fréquence minimale pour l'extraction des motifs.

L'ensemble des motifs fréquents et leurs fréquences associées est : $\{A\} : 4$, $\{B\} : 3$, $\{C\} : 5$, $\{E\} : 9$, $\{G\} : 7$, $\{C, E\} : 4$, $\{C, G\} : 3$, $\{E, G\} : 6$, $\{A, E\} : 3$ et $\{C, H\} : 3$.
La représentation par motifs maximaux fréquents est : $\{C, H\} : 3$, $\{B\} : 3$, $\{E, A\} : 3$, $\{G, C\} : 3$, $\{E, C\} : 4$ et $\{E, G\} : 6$.

À partir de $\mathcal{F}_{con}(\mathcal{T}, 3)$, l'utilisateur ne peut pas régénérer de manière précise toutes les fréquences des motifs fréquents, par exemple des motifs inclus dans $\{G, C\}$, mais grâce à la propriété d'anti-monotonie, l'utilisateur peut déduire, par exemple, que comme $\{C\} \subseteq \{G, C\}$ alors $freq(\{C\}) \geq freq(\{G, C\})$ en approximant de cette manière la fréquence des motifs 3-fréquents.

Un algorithme pionnier d'extraction des motifs maximaux est celui présenté dans [Bayardo, 1998].

#### 3.2.3b Représentation par motifs fermés

La représentation fondée sur les motifs fermés est une représentation exacte [Bayardo, 1998]. Elle tire son origine de la théorie des treillis et plus précisément des travaux autour de l'analyse de concepts formels [Ganter et Wille, 1997a].

Soit $\mathcal{F}_{clos}(\mathcal{T}, min_{freq})$ la représentation condensée des motifs $min_{freq}$-fréquents obtenus à partir de la base de transactions $\mathcal{T}$. Il est possible de régénérer de manière exacte l'ensemble des motifs fréquents uniquement à partir de l'ensemble des motifs fermés comme suit :

Soit $\phi$ un motif dont on veut déterminer la fréquence,

— si $\phi$ ne possède pas de spécialisation contenue dans la collection $\mathcal{F}_{clos}(\mathcal{T}, min_{freq})$, alors $\phi$ n'est pas fréquent,
— sinon, il faut rechercher le plus petit fermé $\psi$ (au sens de l'inclusion) tel que $\phi \subset \psi$. $freq(\phi)$ est alors égale à $freq(\psi)$.

Cette notion de régénération exacte est illustrée dans l'exemple suivant.

**Exemple 3.4.**
Considérons le jeu de données représenté par la table 3.1 et un seuil minimal de fréquence fixé à 2.

Il y a 20 motifs 2-fréquents, mais seulement 15 motifs fermés, voir la table 3.2 contenant tous les motifs fermés. Ainsi, en possédant uniquement la collection des motifs fermés ainsi que leurs fréquences : $\{C, E\} : 4$, $\{C, E, G\} : 2$, $\{C, E, H\} : 2$, $\{C, G\} : 3$ etc., on peut régénérer complètement la collection de tous les motifs 2-fréquents.

Par exemple, la fréquence du motif $\{E, H\}$ est celle du plus petit fermé contenant $\{E, H\}$, ainsi $freq(\{E, H\}) = freq(\{C, E, H\}) = 2$.

| Fermés | Motifs | Transactions couvertes |
|---|---|---|
| $\phi_1$ | $\{E\}$ | $\{t_2,\ t_3,\ t_4,\ t_5,\ t_6,\ t_7,\ t_8,\ t_9,\ t_{10}\}$ |
| $\phi_2$ | $\{G\}$ | $\{t_3,\ t_4,\ t_5,\ t_6,\ t_7,\ t_8,\ t_{11}\}$ |
| $\phi_3$ | $\{E, G\}$ | $\{t_3,\ t_4,\ t_5,\ t_6,\ t_7,\ t_8\}$ |
| $\phi_4$ | $\{C\}$ | $\{t_7,\ t_8,\ t_9,\ t_{10},\ t_{11}\}$ |
| $\phi_5$ | $\{C,\ E\}$ | $\{t_7,\ t_8,\ t_9,\ t_{10}\}$ |
| $\phi_6$ | $\{C,\ G\}$ | $\{t_7,\ t_8,\ t_{11}\}$ |
| $\phi_7$ | $\{C,\ E,\ G\}$ | $\{t_7,\ t_8\}$ |
| $\phi_8$ | $\{A\}$ | $\{t_1,\ t_2,\ t_3,\ t_4\}$ |
| $\phi_9$ | $\{A,\ E\}$ | $\{t_2,\ t_3,\ t_4\}$ |
| $\phi_{10}$ | $\{A,\ E,\ G\}$ | $\{t_3,\ t_4\}$ |
| $\phi_{11}$ | $\{B\}$ | $\{t_1,\ t_5,\ t_6\}$ |
| $\phi_{12}$ | $\{B,\ E,\ G\}$ | $\{t_5,\ t_6\}$ |
| $\phi_{13}$ | $\{C,\ H\}$ | $\{t_9,\ t_{10},\ t_{11}\}$ |
| $\phi_{14}$ | $\{C,\ E,\ H\}$ | $\{t_9,\ t_{10}\}$ |
| $\phi_{15}$ | $\{F\}$ | $\{t_2,\ t_{11}\}$ |
| $\phi_{16}$ | $\{A,\ E,\ F\}$ | $\{t_2\}$ |
| $\phi_{17}$ | $\{C,\ F,\ G,\ H\}$ | $\{t_{11}\}$ |
| $\phi_{18}$ | $\{A,\ B,\ D\}$ | $\{t_1\}$ |

FIGURE 3.2 – Motifs fermés extraits sur la base de transactions sans fréquence minimale.

Les premiers travaux à avoir proposé une méthode pour calculer une représentation condensée par motifs fermés dans le cadre de l'extraction de motifs sont [Boulicaut et Bykowski, 2000], l'algorithme le plus efficace à nos jours pour l'extraction des motifs fermés est présenté dans [Uno *et al.*, 2004].

#### 3.2.3c Représentation par motifs libres

La collection des motifs libres fréquents munis de leur bordure négative est une représentation condensée exacte des motifs fréquents [Boulicaut *et al.*, 2003].

Étant donné un motif $\phi$, un seuil de fréquence minimale $min_{freq}$, l'ensemble des motifs libres fréquents $\mathcal{F}_{libre}(\mathcal{T}, min_{freq})$ et leur bordure négative $\mathcal{B}d^-(\mathcal{F}_{libre}(\mathcal{T}, min_{freq}))$, la fréquence exacte de $\phi$ est déduite comme suit :

— si $\exists \psi \in \mathcal{B}d^-(\mathcal{F}_{libre}(\mathcal{T}, min_{freq}))$ tel que $\psi \subseteq \phi$, alors $\phi$ n'est pas fréquent,
— sinon la fréquence de $\phi$ est la plus petite fréquence des libres fréquents inclus dans $\phi$.

**Exemple 3.5.**
Considérons le jeu de données représenté par la table 3.1 et un seuil minimal de fréquence fixé à 3. L'ensemble des motifs libres relatifs à ce jeu de données est $\mathcal{F}_{libre}(\mathcal{T}, 3) = \{\{B\} : 3, \{H\} : 3, \{A\} : 4, \{A, E\} : 3, \{C\} : 5, \{C, G\} : 3, \{C, E\} : 4, \{G\} : 7, \{G, E\} : 6$ et $\{E\} : 9.\ \}$

Par exemple, $\{C, E, G\}$ n'est pas fréquent car c'est une spécialisation d'un motif de $\mathcal{B}d^-(\mathcal{F}_{libre}(\mathcal{T}, 3))$ (i.e, $\{C, E\}$).

Dans la suite de ce document, nous nous intéressons principalement aux représentations condensées basées sur les motifs fermés.

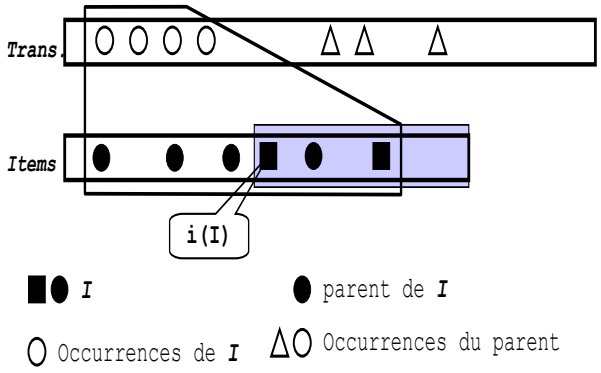



FIGURE 3.3 – Un exemple de parent de $\phi$ : le parent de $\phi$ est obtenu en supprimant des éléments plus grands que $i(\phi)$ (la zone sombre) et prendre une fermeture. Source [Uno *et al.*, 2003].

### 3.2.4 LCM : Extraction des motifs fermés en temps linéaire

L'algorithme LCM [Uno *et al.*, 2004] (connu sous la terminologie anglaise *Linear time Closed item set Miner*) est considéré comme l'un des meilleurs algorithmes d'extraction des motifs fermés, capable d'énumérer linéairement les motifs fermés fréquents, i.e. sans explorer des motifs fréquents non nécessaires. LCM définit une relation parent-enfant entre les motifs fermés fréquents, par conséquent, un arbre sous forme de trajets transversaux composés uniquement de tous les motifs fermés fréquents est créé pour l'exploration et l'énumération. Cet algorithme tire son origine des algorithmes d'énumération des cliques maximales bipartites. De plus, LCM utilise différentes techniques pour accélérer les mises à jour sur les occurrences des motifs. Une technique, appelée *occurrence deliver*, qui calcule simultanément les ensembles d'occurrences de tous les successeurs du motif courant durant une, et une seule, itération de balayage sur l'ensemble d'occurrences actuel, cette technique est très efficace quand il s'agit d'une base de données creuse. Une autre technique, appelée *diffsets* [Zaki et Hsiao, 2002], est utilisée pour réduire l'utilisation de la mémoire des calculs intermédiaires, cette technique est réputée très efficace sur une base de données dense. Une hybridation entre les deux techniques se basant sur une estimation du temps de calcul des deux techniques a été proposée pour utiliser la technique appropriée par rapport à la base de données.

Dans ce qui suit, nous détaillons quelques propriétés dont LCM tire profit pour extraire efficacement les motifs fermés. Plus de détails sur des versions améliorées de LCM se trouvent dans [Uno *et al.*, 2004].

**Définition 3.20. PRÉFIXE**
Soient $\phi = (I, T)$ un motif et $I = \{i_1, i_2, \ldots, i_k\}$ une séquence d'items ordonnés ($i_1 < i_2 < \cdots < i_k$). On définit par $tail(I) = i_k \in \mathcal{I}$. Soit $\psi = (I', T')$ un autre motif. Pour un indice $l$, $I(l) = I \cap \{1, \ldots, l\}$. $I$ est préfixe de $I'$ si $I = I'(l)$ est vraie pour $l = tail(I)$.

Pour un motif $\phi = (I, T)$, on définit par $\mathcal{I}(T) = I$ l'ensemble des items partagés par les transactions dans $T$, et $\mathcal{T}(I) = T$ l'ensemble des transactions partageant tous les items dans $I$.

La relation parent-enfant $P$ pour les motifs fermés $\mathcal{F}_{clos}$ est définie comme suit. Pour $\phi \in \mathcal{F}_{clos}$ tel que $\phi = (I, T)$, on définit parent de $\phi$ par $P(\phi) = \mathcal{I}(\mathcal{T}(I(l(I) - 1)))$, où $l(I)$ est l'item $i_l$ le plus petit tel que $\mathcal{T}(I) = \mathcal{T}(I(l))$ mais $\mathcal{T}(I) \neq \mathcal{T}(I(l-1))$. Si $\psi$ est le parent de $\phi$, on dit que $\phi$ est l'enfant de $\psi$. Soit $\perp = \mathcal{I}(\mathcal{T}(\emptyset))$ le plus petit motif dans $\mathcal{F}_{clos}$ appelé *racine*. Pour tout motif $\phi \in \mathcal{F}_{clos} \backslash \{\emptyset\}$, son parent $P(\phi)$ est toujours défini et appartient à l'ensemble des motifs fermés $\mathcal{F}_{clos}$. Une illustration est donnée dans la figure 3.3.

Pour tout motif $\phi = (I, T) \in \mathcal{F}_{clos}$ et son parent $\psi = (I', T')$, l'inclusion $I' \subseteq I$ est vraie puisque $\mathcal{T}(I(i(I) - 1)) \subset \mathcal{T}(I)$. Par conséquent, la relation $P$ est acyclique, et la représentation graphique forme un arbre. En traversant l'arbre avec une exploration en profondeur d'abord, on peut énumérer tous les motifs fermés fréquents en temps linéaire par rapport à la taille de l'arbre, égale au nombre de motifs fermés fréquents dans $\mathcal{F}_{clos}$.

**Algorithme 18 :** Pseudo-code de LCM basique.

```
1 Fonction LCM (φ = (I,T)) début
2     Afficher(φ) ;
3     pour tous les l > l(I) faire
4         si I[l] est fréquent ∧ I[l] = I(T(I[l])) alors
5             LCM ((I[l], T(I[l]))) ;
```

LCM utilise le lemme suivant pour trouver un enfant à partir d'un motif fermé courant. Pour un motif $\phi = (I, T)$ et un indice $l$, soit $I[l] = I \cup H$, où $H$ est l'ensemble d'items $i_j \in \mathcal{I}(\mathcal{T}(I \cup \{i_l\}))$ vérifiant $j \geq l$.

**Lemme 1** . $\phi = (I', T')$ est un enfant de $\psi = (i, T) \in \mathcal{F}_{clos}$, et le parent de $\phi$ est $\psi$ si et seulement si :

**(condition 1)** $I' = I[l]$ pour certains $l > l(I)$,

**(condition 2)** $I' = \mathcal{I}(\mathcal{T}(I'))$ ($I'$ forme un motif fermé $\phi$),

**(condition 3)** le motif $\phi = (I', T')$ est fréquent.

Clairement, $\mathcal{T}(I[l]) = \mathcal{T}(I \cup \{i_l\})$ est vérifiée si (condition 2) $\mathcal{I}(\mathcal{T}(I[l])) = I[l]$ est vérifiée. Notons que enfant $I'$ de $I$ satisfait $I[l] = I[l'],\ l' \neq l$ puisque l'item le plus petit de $I[l] \backslash I$ et $I[l'] \backslash I$ sont $l$ et $l'$, respectivement. A partir de ce lemme, l'algorithme 18, illustre le pseudo-code basique de LCM explorant l'arbre des motifs fermés à partir d'un motif $\phi$ donné.

## 3.3 Extraction d'ensembles de motifs

Comme présenté dans les sections précédentes, de nombreuses tâches, en particulier sur des bases de données transactionnelles, se résument au calcul d'une théorie $Th(\mathcal{L}, \mathcal{T}, p)$. Cela peut également être considéré comme l'évaluation de la requête inductive où tout motif satisfaisant $q$ doit être calculé. Cette hypothèse d'exhaustivité est assez courante dans des tâches d'extraction des motifs locaux. Un problème majeur bien connu en fouille de données est la taille du volume d'informations extraites rendant son interprétation par les analystes une tâche fastidieuse. Typiquement, un processus d'extraction de motifs locaux retourne de nombreux motifs pertinents, mais également un grand nombre de motifs non pertinents ou redondants rendant très délicate la découverte des motifs pertinents, car l'expert doit analyser tous les motifs extraits afin de déterminer des relations et corrélations entre ces motifs pour ensuite choisir un ensemble ne contenant que des motifs les plus intéressants. En pratique, l'expert est souvent intéressé par la découverte de motifs de plus haut niveau reposant sur des caractéristiques impliquant plusieurs motifs locaux.

Contrairement à l'extraction des motifs locaux, où les problèmes sont dans la plupart des cas bien structurés et l'anti-monotonie est l'un des fils conducteurs pour le développement des algorithmes, il n'existe pas de principes uniformes ou d'algorithmes efficaces pour l'extraction d'ensembles de motifs. Nous présentons dans cette section, les approches existantes permettant de réduire le nombre de motifs extraits en utilisant des contraintes n-aires impliquant plusieurs motifs.

### 3.3.1 Définitions de base

La tâche de recherche des ensembles de motifs implique la découverte d'un ensemble de motifs qui satisfait un ensemble de contraintes impliquant non seulement des motifs ayant des propriétés individuelles, comme dans le cadre d'extraction des motifs locaux, mais par rapport à tout le reste des motifs locaux [Raedt et Zimmermann, 2007, Zimmermann, 2009].

Nous noterons par **L** l'ensemble de tous les motifs locaux possibles qui peuvent être énumérés à partir d'un langage $\mathcal{L}$, $\mathbf{L} = 2^{\mathcal{L}}$. Les motifs locaux (ou individuels) qui apparaissent dans l'ensemble des motifs seront notés par des caractères minuscules comme $\phi, \ldots, \psi$. Nous utiliserons des caractères majuscules tels que $\Phi, \ldots, \Psi$ pour désigner les ensembles de motifs.

Plus formellement, le problème de l'extraction d'ensembles de motifs peut être formulé comme un problème de calcul d'une théorie $\mathbf{Th}(\mathbf{L}, \mathcal{T}, p) = \{\Phi \in \mathbf{L} \mid p(\Phi, \mathcal{T}) \ is \ true\}$, où $p(\Phi, \mathcal{T})$ est un prédicat de sélection indiquant les contraintes qui portent sur l'ensemble de motifs $\Phi$ par rapport à la base de transactions $\mathcal{T}$.

### 3.3.2 Contraintes n-aires

Dans cette section, nous présentons les contraintes proposées dans la littérature pour l'extraction d'ensembles de motifs.

**Définition 3.21. CONTRAINTE N-AIRE**
Une contrainte n-aire $p(\mathcal{T}, \Phi)$ est une contrainte portant sur $n$ motifs, relativement à un ensemble de transactions $\mathcal{T}$ et un ensemble de motifs $\Phi$.

Nous présentons certaines contraintes n-aires introduites dans [Raedt et Zimmermann, 2007]. Ces contraintes sont des adaptations directes des contraintes bien connues pour l'extraction des motifs locaux. Ces contraintes définissent et mesurent généralement la façon dont un ensemble de motifs couvre les données.

— **Couverture d'un ensemble de motifs**. La couverture d'un ensemble de motifs $\Phi$, notée par $cov(\Phi, \mathcal{T})$, est l'ensemble des transactions de $\mathcal{T}$ couvertes par $\Phi$ :

$$cover(\Phi, \mathcal{T}) = \bigcup_{\phi \in \Phi} cover(\phi, \mathcal{T}).$$

Selon cette définition, $\Phi$ est interprété comme la disjonction des motifs locaux $\phi$ qu'il contient.

— **Fréquence d'un ensemble de motifs**. La fréquence d'un ensemble de motifs $\Phi$, notée $sup(\Phi, \mathcal{T})$, est calculée de la même manière que le support des motifs locaux :

$$freq(\Phi, \mathcal{T}) = \mid cover(\Phi, \mathcal{T}) \mid .$$

— **Taille d'un ensemble de motifs**. La taille d'un ensemble de motifs, notée par $taille(\Phi)$, est le nombre de motifs locaux formant l'ensemble de motifs $\Phi$ :

$$taille(\Phi) = |\Phi|.$$

— **Aire d'un ensemble de motifs**. La superficie d'un ensemble de motifs a été étudiée dans le contexte de la recherche d'un ensemble de tuiles [Geerts *et al.*, 2004], L'aire d'un ensemble de motifs $\Phi$, notée par $aire(\Phi, \mathcal{T})$, est définie par l'aire de tous les tuiles des motifs locaux $\phi$ contenus dans l'ensemble :

$$aire(\Phi, \mathcal{T}) = \mid \bigcup_{\phi \in \Phi} aire(\phi, \mathcal{T})|.$$

— **Généralisation d'un ensemble de motifs**. Un ensemble de motifs $\Phi$ est plus général qu'un ensemble de motifs $\Psi$, noté par $\Phi \preceq \Psi$, si et seulement si, pour tout motif $\psi \in \Psi$, il existe un motif $\phi \in \Phi$ tel que $\phi \preceq \psi$.

Étant donné une base de transactions $\mathcal{T}$ organisée en bases de transactions partielles $\mathcal{T}_1, \ldots, \mathcal{T}_n \subseteq \mathcal{T}$ qui peuvent se chevaucher. Les contraintes discriminantes peuvent être utilisées pour mesurer et optimiser la façon

dont un ensemble de motifs distingue entre des transactions appartenant à une base de transactions partielle $\mathcal{T}_i$ [18]. Ces mesures sont généralement définies en comparant le nombre de transactions couvertes par l'ensemble de motifs sur une base de transactions partielle $\mathcal{T}_i$ au nombre total de transactions couvertes dans $\mathcal{T}$. Dans [Raedt et Zimmermann, 2007], certaines mesures utiles dérivées de la notion de support ont été proposées afin de discriminer les ensembles de motifs. Nous rappelons les mesures les plus importantes.

#### 3.3.2a Représentativité

La représentativité d'un ensemble de motifs indique la caractéristique des transactions couvertes par l'ensemble de motifs par rapport à une base de transactions partielle $\mathcal{T}_i$ :

$$rep(\Phi, \mathcal{T}_i, \mathcal{T}) = freq(\Phi, \mathcal{T}_i)/freq(\Phi, \mathcal{T}).$$

Dans le cas où l'appartenance à l'ensemble de données $D_i$ serait représentée comme un item, la représentativité correspondrait à la confiance de la règle d'association avec le motif comme un antécédent et la classe comme un conséquent.

#### 3.3.2b Précision

Dans l'apprentissage supervisé, nous pouvons rechercher un ensemble de motifs avec une grande précision sur des données libellées, c'est à dire des données dans lesquelles il y a un attribut de classe attaché à chaque transaction. En considérant la répartition de la base de transactions en plusieurs sous-ensembles de transactions $\mathcal{T}_i$, où chaque sous-ensemble $\mathcal{T}_i$ ne contient que des transactions d'une classe $i$, la précision d'un motif $\phi$ est définie comme suit :

$$acc(\phi) = \frac{\max_{\mathcal{T}_i \in \mathcal{T}} freq(\phi, \mathcal{T}_i)}{freq(\phi, \mathcal{T})}.$$

La précision d'un ensemble de motifs entier est plus difficile à quantifier. En effet, un motif peut être considéré comme une règle prédictive, qui prédit la classe majoritaire des transactions qu'il couvre. Pour un ensemble de motifs, nous considérons un *ensemble de règles*. Par conséquent, agréger des prédictions d'un ensemble de règles est un axe de recherche en soi.

En utilisant la précision d'un motif local, nous pouvons, néanmoins, définir une *approximation* de la précision d'un ensemble de motifs comme suit :

$$acc(\Phi) = \frac{\sum_{\phi \in \Phi} acc(\phi) \times freq(\phi, \mathcal{T})}{\sum_{\phi \in \Phi} freq(\phi, \mathcal{T})}.$$

Cette formulation implique implicitement que les motifs ne doivent pas (ou très peu) se chevaucher.

#### 3.3.2c Contraintes de redondance

La contrainte de redondance est utilisée pour limiter (ou minimiser) la redondance entre les différents motifs. Une façon de mesurer cette redondance est d'énumérer le nombre de transactions couvertes par des motifs

18. Est-ce que l'ensemble de motifs est aussi représentatif et précis sur une base de transactions partielle ?

multiples dans l'ensemble de motifs $\Phi$. En considérant la couverture de deux motifs $\phi, \psi$, et en particulier leur chevauchement, on peut quantifier leur redondance comme :

$$red(\Phi, \mathcal{T}) = |\{t \in \mathcal{T} \mid \exists(\phi, \psi) \in \Phi, t \in chevauche(\phi, \psi, \mathcal{T})\}|,$$

où $chevauche(\phi, \psi, \mathcal{T}) = cover(\phi, \mathcal{T}) \cap cover(\psi, \mathcal{T})$ désigne le chevauchement entre les deux motifs $\phi$, $\psi$. Une autre définition basée sur des pairs de motifs a été proposée et utilisée dans [Raedt et Zimmermann, 2007].

## 3.4 Problèmes de l'extraction d'ensembles de motifs

### 3.4.1 Clustering conceptuel

#### 3.4.1a Concept formel

Soit le contexte transactionnel $(\mathcal{I}, \mathcal{R}, \mathcal{T})$, appelé également *contexte formel* dans l'analyse de concepts formels [Ganter et Wille, 1997b]. Deux opérateurs sont définis :

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

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

Le couple ($I = int(T)$, $T = ext(I)$) est appelé un **concept formel**. Cette formulation définit une **propriété de fermeture** sur la base de transactions $\mathcal{T}$ :

$$closed(\phi) \Leftrightarrow I = int(ext(I)).$$

Un motif $\phi = (I, T)$ pour lequel $closed(\phi) = vrai$ est appelé un **motif fermé**.

En utilisant $ext(I)$, la *fréquence* d'un concept est définie par :

$$freq(\phi) = |ext(I)|,$$

sa *diversité* par :

$$divers(\phi) = \sum_{t \in ext(I)} |\{i \in \mathcal{I} \mid \ (i \ \notin \ I) \ \wedge \ (i \ \in \ t)\}|.$$

De plus, nous pouvons nous référer à sa *taille* par :

$$taille(\phi) = |\{i \mid i \in I\}|.$$

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

TABLE 3.2 – Trois clusterings conceptuels pour $k$=3 sur la base de transactions $\mathcal{T}$ (voir section 3.1.1).

#### 3.4.1b Problème du clustering conceptuel

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

Différentes tâches de clustering conceptuel existent dans la littérature, qui peuvent être définies par différentes relations entre les transactions, les items et les motifs fermés [Berkhin, 2006]. Outre le clustering conceptuel classique qui cherche un partitionnement de l'ensemble des transactions sans aucun chevauchement entre les concepts du clustering, on distingue le **soft clustering**, le **co-clustering**, et le **soft co-clustering**. Les requêtes correspondant à ces trois variantes sont détaillées dans le chapitre 7.

Formellement, le problème de clustering conceptuel peut être formulé comme la recherche d'un ensemble de $k$ clusters disjoints, couvrant l'ensemble des transactions $\mathcal{T}$, chaque cluster étant décrit par un motif fermé $\phi_i$ ($1 \leq i \leq k$). Par exemple, la table 3.2 illustre les différents clusterings pour $k$=3.

Une fonction d'évaluation $f$ est nécessaire pour évaluer la qualité d'un clustering. Ainsi, le clustering conceptuel cherche un ensemble disjoint de clusters sur $\mathcal{T}$ qui optimisent un critère donné sur la qualité du clustering. Par exemple, pour la base de transactions $\mathcal{T}$ et $k$=3, minimiser $f(\phi_1, ..., \phi_k) = \sum_{1 \leq i \leq k} \texttt{divers}(\phi_i)$ fournit le clustering $s_1$ de valeur optimale 18 (voir la table 3.2).

### 3.4.2 Problème du tuilage

#### 3.4.2a Tuiles

Étant donné une base de transactions $\mathcal{T}$ définie sur un ensemble d'items $I$, une représentation binaire de notre base de transactions représentée par la table 3.1 est illustrée par la figure 3.4. De façon informelle, une tuile est un bloc de "1" sur une base de transactions, le nombre de "1" couverts par une tuile est appelé l'aire d'une tuile.

Soit $\phi = (I, T)$ un motif, la tuile correspondant à $\phi$ est définie par :

$$\tau(\phi) = \{(t, i) \mid t \in cover(\phi), i \in I\}.$$

L'aire d'une tuile $\tau(\phi)$ est égale à son cardinal :

$$aire(\tau(\phi)) = |\tau(\phi)| = |I| \times |cover(I)|.$$

Par exemple, la figure 3.4 illustre trois tuiles représentées par des niveaux de gris différents. La tuile en gris foncé correspond au motif $\{A, E\}$ avec une aire égale à 3, la tuile en gris clair correspond au motif $\{B\}$ avec une aire égale à 3, la tuile en gris correspond au motif $\{C\}$ avec une aire égale à 5.

Soit $\Phi = \{\tau(\phi_1), \ldots, \tau(\phi_k)\}$ un ensemble fini de $k$ tuiles. L'aire d'un ensemble de tuiles est définie par :

$$aire(\Phi) = |\tau(\phi_1) \cup \cdots \cup \tau(\phi_k)|.$$

| Trans. | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| $t_1$ | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| $t_2$ | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| $t_3$ | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| $t_4$ | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| $t_5$ | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| $t_6$ | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| $t_7$ | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| $t_8$ | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| $t_9$ | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| $t_{10}$ | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| $t_{11}$ | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |

FIGURE 3.4 – Base de transactions avec une représentation binaire.

Une tuile $\tau(\phi)$ est maximale si le motif correspondant $\phi$ est un motif fermé.

Différentes variantes du problème de tuilage existent dans la littérature [Geerts *et al.*, 2004], nous détaillons dans le chapitre 7 quelques variantes les plus connues pour modéliser nos requêtes PLNE.

### 3.4.3 Algorithmes d'extraction

Contrairement à l'extraction des motifs locaux, où les problèmes sont souvent bien structurés et où l'anti-monotonie est l'un des fils conducteurs pour le développement des algorithmes, nous constatons qu'il y a rarement (pour ne pas dire aucune) des propriétés uniformes permettant le développement d'algorithmes dédiés efficaces pour l'extraction d'ensembles de motifs. Nous trouvons, généralement dans la littérature, des approches qui procèdent en plusieurs étapes et qui utilisent une large gamme d'heuristiques et de stratégies de recherche gloutonne (section 3.4.3a).

En effet, dans une approche à plusieurs étapes, l'étape d'extraction des motifs locaux constitue une étape intermédiaire dans le processus de découverte de connaissances. Dans une telle étape, les motifs locaux satisfaisants un ensemble de contraintes locales sont recherchés d'une manière exhaustive. Dans le reste des étapes, les motifs locaux sont sélectionnés et combinés, souvent moyennant une heuristique, pour créer un ensemble de motifs sur toute la base de transactions. L'ensemble des motifs résultant est généralement de mauvaise qualité et dépend fortement des paramètres d'initialisation. De plus, les contraintes sur les ensembles de motifs ne sont pas toujours explicitement spécifiées mais sont intégrées dans l'algorithme. Par conséquent, l'extension de ces algorithmes pour résoudre de nouveaux problèmes est une tâche difficile qui nécessite de revisiter tout l'algorithme ce qui est, dans certains cas, non réalisable.

Outre ces approches heuristiques, d'autres approches de l'extraction d'ensembles de motifs ont été proposées pour aborder les questions de déclarativité et d'extension vers de nouveaux problèmes (section 3.4.3b). Ces approches étaient cependant limitées à quelques paramètres, se focalisent sur une tâche particulière, ou ne passent pas à l'échelle. Au meilleur de nos connaissances, il n'existe pas d'approche efficace de l'extraction d'ensembles de motifs capable de résoudre un grand nombre de tâches.

#### 3.4.3a Approches heuristiques

Plusieurs méthodes ont exploré l'idée de découpler la phase de formation de clusters de la phase de recherche des descriptions conceptuelles. Pensa *et al.* [Pensa *et al.*, 2005] ont d'abord commencé par la fouille de motifs

(itemsets) fermés (ou $\delta$-fermés) et de son extension en lui appliquant ensuite un clustering *k-Means*. Perkowitz et Etzioni [Perkowitz et Etzioni, 1999] font le contraire : leur *fouille de clusters* utilise d'abord une technique de clustering pour former les clusters, puis, exploite ce clustering pour trouver des descriptions à l'aide de l'apprentissage de règles. Les instances sont appariées aux descriptions, conduisant potentiellement à des clusters qui se chevauchent.

Dans [Geerts *et al.*, 2004], deux problèmes de tuilage sont considérés : i) trouver les $k$ tuiles qui couvrent le plus de transactions possibles, et ii) trouver le nombre minimal de tuiles qui couvent toutes les transactions. Les deux problèmes sont évidemment des problèmes de clustering conceptuel, les tuiles peuvent se chevaucher, comme pour les clusters. Une analyse de la complexité de plusieurs algorithmes liés à la recherche d'un tuilage a été menée. Les auteurs proposent un algorithme d'approximation pour trouver des tuilages qui se rapprochent de la solution optimale avec des facteurs raisonnables.

#### 3.4.3b Approches déclaratives

Il y a très peu de tentatives dans la littérature sur la recherche d'ensembles de motifs par des approches complètes. L'idée originale de l'extraction d'ensembles de motifs a été proposée dans [Raedt et Zimmermann, 2007]. Les auteurs introduisent formellement une variété de contraintes au niveau des ensembles de motifs. Malheureusement, les techniques d'élagage sont faibles et l'algorithme reste limité à une petite collection de modèles.

**PPC/SAT pour la fouille d'ensemble de motifs.** Le paradigme de la programmation par contraintes est au cœur des approches génériques qui traitent le problème des ensembles de $k$-motifs [Guns *et al.*, 2013]. Ces méthodes permettent de modéliser les ensembles de motifs satisfaisant des propriétés sur l'ensemble des motifs, de manière déclarative et flexible, comme le clustering conceptuel, le $k$-tiling, pour ne citer que ceux-là. Elles nécessitent cependant de fixer au préalable le nombre de motifs et souffrent du problème de passage à l'échelle. Dans [Métivier *et al.*, 2012], les auteurs formulent le processus de clustering conceptuel sous forme de requêtes dans un langage donné. Ces requêtes sont traduites en clauses SAT et sont résolues par un solveur SAT. Cependant, la plate-forme proposée ne permet ni d'exprimer des critères d'optimisation ni le passage à l'échelle.

**PLNE/PPC pour le clustering basé-distance.** Le clustering basé-distance repose sur les dissimilarités (ou similarités) entre des attributs numériques. Dans cet axe, plusieurs approches déclaratives ont été proposées. Dans [Dao *et al.*, 2013a], une approche PPC a été utilisée. Cette approche supporte diverses sortes de contraintes utilisateurs. Dans [Babaki *et al.*, 2014], une approche exacte qui utilise la génération de colonnes pour la résolution d'un PLNE est présentée. Leur approche étend celle proposée par [Aloise *et al.*, 2012]. Dans [Mueller et Kramer, 2010b], une approche de clustering sous contraintes utilisant la PLNE a été proposée. Elle prend en entrée un ensemble de clusters de base possibles et construit un clustering en y sélectionnant un sous-ensemble approprié. Le nombre $k$ de clusters doit être fixé au préalable. Dans leurs expérimentations, l'ensemble des clusters de base est généré à partir des motifs fréquents. Ceci rend l'approche moins applicable car le nombre de clusters, représentés par les motifs fréquents, est énorme rendant ainsi la sélection d'un sous-ensemble approprié une tâche difficile.

Toutefois, ces méthodes nécessitent de fixer la taille de l'ensemble de motifs, rendant ainsi ces approches limitées au problème de l'extraction d'ensembles de $k$-motifs, ayant tendance à avoir des problèmes de passage à l'échelle.

## 3.5 Bilan

L'extraction des motifs locaux est une tâche fondamentale dans le processus de découverte de connaissances. Des techniques dans différents domaines ont été développées pour produire des collections étendues de motifs de différents types. Cependant, l'ensemble des motifs résultants est généralement de très grande taille et non structuré ce qui le rend difficile à exploiter. Ceci explique l'intérêt qu'a suscité l'extraction d'ensembles de motifs [Raedt

et Zimmermann, 2007]. Au lieu d'évaluer et de sélectionner individuellement les motifs, les ensembles de motifs regroupent des motifs locaux pour fournir des connaissances à partir d'un point de vue de haut niveau en utilisant des mesures de qualité et des contraintes qui limitent la sélection des motifs. Vu que le nombre des ensembles de motifs possibles est exponentiel ($2^{|\mathcal{L}|}$) par rapport au nombre de motifs locaux, qui est lui-même de grande taille ($|L| = 2^{|\mathcal{I}|}$), trouver une méthode qui soit efficace pour l'extraction d'ensembles de motifs est un défi en soi.

Les différents problèmes ainsi que les différentes bases de transactions seront détaillés dans le chapitre suivant afin d'évaluer, respectivement, nos contributions sur l'exploitation du parallélisme renforcé par la décomposition arborescente et notre démarche basée sur la PLNE pour l'extraction d'ensembles de motifs.

# Chapitre 4

# Jeu de test

## Sommaire


**4.1 Problème d'allocation de fréquences à des liens radio . . . . . 64**
4.1.1 Description du problème . . . . . 64
4.1.2 Modélisation sous forme d'un WCSP . . . . . 64
4.1.3 Description des instances . . . . . 64
**4.2 Instances générées par GRAPH . . . . . 67**
**4.3 Problème de planification d'un satellite de prise de vue . . . . . 67**
4.3.1 Description du problème . . . . . 67
4.3.2 Modélisation sous forme d'un WCSP . . . . . 68
4.3.3 Présentation des instances . . . . . 68
**4.4 Problème de sélection de TagSNP . . . . . 68**
4.4.1 Description du problème . . . . . 69
4.4.2 Modélisation sous forme d'un WCSP . . . . . 69
4.4.3 Présentation des instances . . . . . 70
**4.5 Bases de données de l'UCI . . . . . 70**
4.5.1 Descriptions des bases de données . . . . . 70
**4.6 Conclusion . . . . . 71**


Dans ce chapitre nous présentons les différents jeux de données (RLFAP, GRAPH, SPOT5 TagSNP, et les bases de données de l'UCI [19]) utilisés dans nos expérimentations. Afin d'expérimenter et d'évaluer correctement nos contributions sur les méthodes parallèles exploitant la décomposition arborescente, nous reprenons les mêmes instances utilisées dans la thèse de Fontaine [Fontaine, 2013] avec les mêmes configurations. Nous reprenons dans ce chapitre les mêmes descriptions introduites dans sa thèse. Pour chaque problème, nous indiquons ses caractéristiques principales, dont sa taille et sa difficulté ainsi que le coût de la solution optimale, si elle est connue, ou la meilleure solution connue. Pour les instances RLFAP, nous détaillons également la structure des réseaux de fonctions de coût de certaines instances. Sur le deuxième volet de la thèse, nous décrivons certains jeux de données de l'UCI considérés pour évaluer nos contributions. Nous présentons pour chaque jeu de données ses caractéristiques et le type de données qu'il contient.

Pour nos expérimentations du chapitre 5, nous utilisons, pour toutes les instances des différents problèmes, les décompositions arborescentes générées par la méthode MCS et MINFILL. Ces décompositions sont créées à partir du graphe de contraintes de chaque instance.

19. Dépôt d'apprentissage automatique en ligne, `https://archive.ics.uci.edu/ml/datasets.html`

## 4.1 Problème d'allocation de fréquences à des liens radio

Le but des instances RLFAP est d'assigner, à un ensemble de liens radios, une fréquence tout en minimisant les interférences entre les liens.

### 4.1.1 Description du problème

Le problème d'allocation de fréquences à des liens radio (Radio Link Frequency Assignment Problem, RLFAP) est composé d'un ensemble de liens radio, représentant chacun une communication entre deux sites. L'objectif consiste à attribuer, à chaque lien, une fréquence radio sous les contraintes suivantes :

— chaque lien radio est limité à une plage de fréquences,
— des interférences peuvent survenir lorsqu'un même site reçoit et émet en même temps sur des fréquences proches, ou qu'une certaine distance sépare deux sites,
— lorsqu'un site A communique avec un site B sur une fréquence, le site B doit lui répondre sur une fréquence décalée d'une valeur $\delta$. Les liens radio entre les sites A et B sont appelés *duplex links*,
— certains liens radios peuvent être initialement affectés.

### 4.1.2 Modélisation sous forme d'un WCSP

Ce problème peut être modélisé comme suit. L'ensemble des variables, $X$, représente l'ensemble des liens radio à affecter et $D$ l'ensemble des fréquences disponibles pour chaque lien. Les fonctions de coût sont les suivantes :

— une fonction de coût unaire (équation 4.1) est associée pour chaque lien radio $x_i$ initialement affecté à la fréquence $f_i$,
— une fonction de coût binaire (équation 4.2) représente les interférences possibles entre deux liens radio $x_i$ et $x_j$ ; $d_{ij}$ représente l'écart minimal entre les fréquences des deux liens radio.
— une fonction de coût binaire (équation 4.3) représente les *duplex links* entre les liens $x_i$ et $x_j$ ; $\delta_{ij}$ représente la valeur de décalage.

$$x_i = f_i \tag{4.1}$$

$$|x_i - x_j| > d_{ij} \tag{4.2}$$

$$|x_i - x_j| = \delta_{ij} \tag{4.3}$$

### 4.1.3 Description des instances

Suivant l'instance à résoudre, l'objectif ainsi que le type des fonctions de coût (i.e. dures ou molles) peuvent varier. Pour les instances fournies par le Centre d'Electronique de l'Armement (CELAR), les fonctions de coût du type 4.3 sont toujours dures et les autres molles. L'objectif peut être :

— CARD : de minimiser le nombre de fréquences utilisées pour affecter les liens radios,
— SPAN : de minimiser la valeur de la fréquence la plus élevée utilisée,
— MAX : de minimiser la somme des coûts. Ce critère est pertinent lorsque toutes les fonctions de coût ne peuvent être satisfaites simultanément.

Pour tous ces cas, le problème RLFAP est NP-Complet, puisque les fonctions de coût de type 4.3 permettent d'exprimer le problème de $k$-coloration [Cabon *et al.*, 1999]. Le tableau 4.1 résume les principales caractéristiques des instances fournies par le CELAR : nombre de variables $n$, la taille du plus grand domaine $d$, nombre de fonctions de coût $e$ et la meilleure solution connue $S^*$.

| Scen# | $n$ | $d$ | $e$ | Type | $S^*$ |
|---|---|---|---|---|---|
| 01 | 916 | 44 | 5548 | CARD | 16 |
| 02 | 200 | 44 | 1235 | CARD | 14 |
| 03 | 400 | 44 | 2760 | CARD | 14 |
| 04 | 680 | 44 | 3967 | CARD | 46 |
| 05 | 400 | 44 | 2598 | SPAN | 792 |
| 06 | 200 | 44 | 1322 | MAX | 3389 |
| 07 | 400 | 44 | 2866 | MAX | 343 592 |
| 08 | 916 | 44 | 5745 | MAX | 262 |
| 09 | 680 | 44 | 4103 | MAX | 15571 |
| 10 | 680 | 44 | 4103 | MAX | 31516 |
| 11 | 680 | 44 | 4103 | CARD | 22 |

TABLE 4.1 – Tableau des principales caractéristiques des instances fournies par le CELAR.

La figure 4.1 détaille le graphe de contraintes associé à l'instance Scen07. Sur cette instance, les variables contenues dans une même ellipse forment une clique. Scen06, Scen07 et Scen08 sont des instances réputées difficiles à résoudre. Nous allons donc limiter nos expérimentations à ces trois instances dans la suite de ce mémoire.

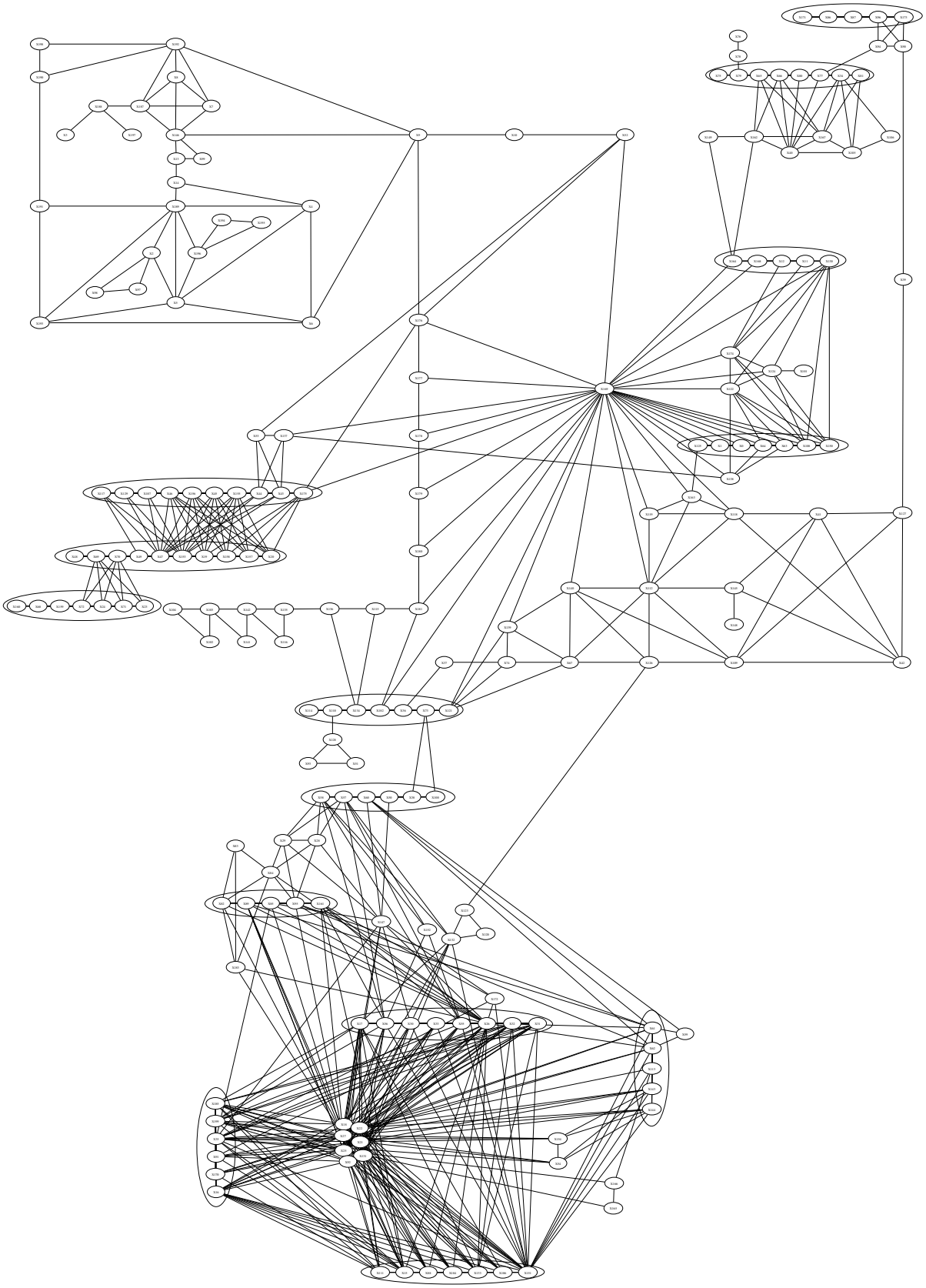


FIGURE 4.1 – Graphe de contraintes de l'instance Scen07 du CELAR.

## 4.2 Instances générées par GRAPH

GRAPH, *Generating Radio link frequency Assignment Problems Heuristically* [van Benthem, 1995], est un générateur d'instances aléatoires structurées. L'objectif de GRAPH est de générer des instances de difficulté et de structure similaires à celles fournies par le CELAR. Ces instances possèdent les caractéristiques suivantes :

— les domaines sont identiques à ceux des instances du CELAR,
— le ratio entre le nombre de liens radio et leurs interférences est approximativement le même que celui des instances du CELAR,
— la structure des instances pour les fonctions de coût de l'équation 4.3, est la même que celle des instances du CELAR.

Le type des instances générées par GRAPH peut être sélectionné parmi : Feas, CARD, SPAN ou MAX. Lors de la génération des instances de type Feas, CARD ou SPAN, GRAPH fournit les solutions optimales. Pour les instances de type MAX, le générateur fournit une borne supérieure du coût de la solution optimale. Le tableau 4.2 détaille les principales caractéristiques des instances générées par GRAPH.

| Graph# | $n$ | $d$ | $e$ | Type | $S^*$ |
|---|---|---|---|---|---|
| 01 | 200 | 44 | 1134 | CARD | 18 |
| 02 | 400 | 44 | 2245 | CARD | 14 |
| 03 | 200 | 44 | 1134 | SPAN | 380 |
| 04 | 400 | 44 | 2244 | SPAN | 394 |
| 05 | 200 | 44 | 1134 | MAX | 221 |
| 06 | 400 | 44 | 2170 | MAX | 4123 |
| 07 | 400 | 44 | 2170 | MAX | 4324 |
| 08 | 680 | 44 | 3757 | CARD | 18 |
| 09 | 916 | 44 | 5246 | CARD | 18 |
| 10 | 680 | 44 | 3907 | SPAN | 394 |
| 11 | 680 | 44 | 3757 | MAX | 3080 |
| 12 | 680 | 44 | 4017 | MAX | 11827 |
| 13 | 916 | 44 | 5273 | MAX | 10110 |
| 14 | 916 | 44 | 4638 | CARD | 10 |

TABLE 4.2 – Tableau des principales caractéristiques des instances générées par GRAPH.

## 4.3 Problème de planification d'un satellite de prise de vue

La gestion d'un satellite de prise de vue, tel que le satellite *Spot*, consiste à planifier chaque jour les prises de vue à réaliser le lendemain. Le but est de maximiser les gains liés à la vente de ces prises de vue tout en prenant en compte les contraintes techniques du satellite, mais aussi météorologiques ou géographiques. Une description detaillée du problème est donnée dans [Bensana *et al.*, 1999].

### 4.3.1 Description du problème

Un problème de planification SPOT5 est composé d'un ensemble $S$ de prises de vue pouvant être réalisées. On associe à chaque photo un poids, issu de l'agrégation de plusieurs critères (urgence, importance du client, prévision météorologique du lieu, etc.). Chaque photographie peut être réalisée sur l'un des trois appareils du satellite en mono. Si elle doit être réalisée en stéréo, cela nécessitera obligatoirement l'utilisation des appareils avant et arrière du satellite. Enfin, il faut respecter un certain nombre de contraintes dures :

— les prises de vue ne peuvent pas se chevaucher et il faut respecter un temps de transition avant de pouvoir prendre une nouvelle photographie avec un même appareil ;
— la quantité d'informations transitant dans le satellite à un moment donné est limitée ;
— la capacité d'enregistrement est, elle aussi, limitée.

Le but est de sélectionner un sous-ensemble de prises de vue $S' \subseteq S$ qui permet de satisfaire les contraintes dures tout en minimisant la somme des poids des prises de vue qui ne sont pas réalisées.

### 4.3.2 Modélisation sous forme d'un WCSP

La modélisation sous forme d'un WCSP des problèmes SPOT5 est réalisée de la façon suivante :
— Une variable $x_i$ est associée à chaque prise de vue.
— Le domaine $D_i$ associé à $x_i$ contient les appareils possibles pour cette prise de vue (0 désigne le fait de ne pas prendre la photographie) :
  — $\{1, 2, 3, 0\}$ pour une photographie mono, chaque valeur correspondant à un appareil ;
  — $\{13, 0\}$ pour une photographie stéréo, 13 signifiant l'utilisation des appareils 1 et 3.
— Une fonction de coût unaire $w_i$ associée à chaque variable $x_i$ interdisant la valeur 0 avec un coût égal au poids de la photographie (le coût engendré par le fait de ne pas prendre la photographie).
— Les contraintes de chevauchement sont traduites par des fonctions de coût binaires dures attribuant un coût $\top$ [20] si deux prises de vue ne respectent pas les temps de transition.
— Les contraintes de débit sont représentées par des fonctions de coût dures binaires ou ternaires.
— La contrainte de capacité mémoire est représentée par une fonction de coût $n - aire$ dure.

Cette modélisation assure l'existence d'une solution ne violant aucune des fonctions de coût dures. En effet, ne sélectionner aucune photographie produit une solution admissible de très mauvaise qualité.

### 4.3.3 Présentation des instances

Les données proposées par le CNES contiennent 498 instances réparties en deux groupes.

Le premier groupe contient 384 problèmes ne considérant qu'une orbite afin d'éliminer la limitation de capacité mémoire. Ils sont repartis en 3 groupes :
— les instances #1 à #362 ont été générées par un simulateur (LOSICORDOF) ;
— les instances #401 à #413 ont été construites à partir de la plus grande instance générée (la numéro 11) en réduisant le nombre de prises de vue ;
— les instances #501 à #509 elles aussi sont construites à partir de l'instance 11, mais cette fois-ci en réduisant le nombre de conflits entre les photographies.

Le second groupe contient 114 problèmes considérant plusieurs orbites consécutives, et donc cette fois-ci en prenant en compte la limitation de capacité mémoire. Nous considérons pour nos expérimentations 3 instances les plus difficiles, sans contraintes de mémoire, dont nous reportons les caractéristiques dans le tableau 4.3, avec $n$ le nombre de variables, $d$ la taille du plus grand domaine, $e$ le nombre de fonctions de coût et $S^*$ le coût de la solution optimale.

## 4.4 Problème de sélection de TagSNP

Le problème de sélection de TagSNP [Hirschhorn et Daly, 2005] est issu de la recherche en génétique et plus précisément en analyse du polymorphisme. Un SNP (*Single Nucleotide Polymorphism*) est une variation ponctuelle

20. $\top$ est un entier supérieur à la somme des poids de toutes les prises de vue.

| Instance | $n$ | $d$ | $e$ | $S^*$ |
|---|---|---|---|---|
| #414 | 364 | 4 | 10,108 | 38,478 |
| #505 | 240 | 4 | 2,242 | 21,253 |
| #509 | 348 | 4 | 8,624 | 36,446 |

TABLE 4.3 – Caractéristiques des instances SPOT5.

apparaissant dans le génome d'individus d'une même espèce. Un SNP se manifeste par la modification d'une seule paire de bases de nucléotides (A, T, C ou G) du génome, et existe pour environ 2 paires sur mille, représentant ainsi 90% des variations génétiques chez l'humain. L'étude des SNP est capitale pour la recherche sur les maladies génétiques héréditaires. Cependant, la taille des données les rend impossibles à manipuler en pratique. Le problème de sélection de TagSNP vise à compresser (avec perte) les données en choisissant un sous-ensemble de SNP qui représenteront les autres.

### 4.4.1 Description du problème

Le but est de trouver un ensemble de SNP représentant tous les SNP. Un TagSNP est considéré comme représentatif d'un autre SNP à traiter s'ils sont suffisamment corrélés dans la population étudiée. La corrélation entre deux SNP $i$ et $j$ est représentée par une mesure notée $r^2_{ij}$ et doit être supérieure à un seuil $\theta$ pour que la corrélation soit significative. Cette valeur est généralement fixée à $\theta = 0.8$. On construit alors un graphe $G = (V, E)$ dans lequel chaque SNP $i$ est représenté par un sommet $u_i$ et $\{u_i, u_j\} \in E$ si $r^2_{ij} > \theta$. On appelle ce graphe *graphe de corrélation*. Le problème est alors réduit à un *Set Covering problem* (NP-difficile), c'est à dire trouver un ensemble de TagSNP de taille minimale.

Le nombre de solutions étant très grand, on cherche à exprimer des préférences entre les solutions. L'objectif est alors de trouver un ensemble de TagSNP maximisant deux critères. Le premier est la représentativité du groupe. Celui-ci doit en effet représenter les SNP non sélectionnés en restant le plus petit possible. Le second critère est la dispersion. On cherche à minimiser la corrélation entre deux TagSNP.

### 4.4.2 Modélisation sous forme d'un WCSP

Soit un graphe pondéré $G = (V, E)$ dans lequel chaque sommet $u \in V$ représente un SNP et chaque arête $(u, v) \in E$ est pondérée par la mesure $r^2$ entre les SNP $u$ et $v$ ($r^2$ devant être supérieur à $\theta$). Le problème de sélection de TagSNP est modélisé par un WCSP binaire construit de la façon suivante :

- chaque SNP $i$ est représenté par deux variables :
  - $i_s$, variable booléenne indiquant la sélection du SNP en tant que TagSNP ;
  - $i_r$, variable de représentation ayant pour domaine l'ensemble des voisins de $i$ dans $G$. Elle indique le TagSNP représentant $i$.
- Une fonction de coût binaire dure est associée à chaque SNP $i$ afin d'assurer la cohérence entre $i_s$ et $i_r$ ($i_s \implies i_r = i$).
- Une fonction de coût binaire dure assure la cohérence entre un SNP et son représentant ($i_r = j \implies j_s$).
- Une fonction de coût unaire sur $i_s$ génère un coût de $\lfloor 100.\frac{1-r^2_{i,i_r}}{1-\theta} \rfloor$ si $i_r \neq i$. Cette fonction de coût capture la représentativité du SNP. Plus la corrélation du SNP $i$ avec son représentant $i_r$ est forte, plus le coût généré est faible.
- Une fonction de coût binaire entre $i_s$ et $j_s$ génère un coût $\lfloor 100.\frac{r^2_{ij}-\theta}{1-\theta} \rfloor$ si les SNP $i$ et $j$ sont sélectionnés ($i_s = i_j = vrai$). Cette fonction de coût permet de forcer la dispersion des TagSNP. En effet, plus deux TagSNP seront corrélés, plus le coût généré sera élevé.

### 4.4.3 Présentation des instances

Les instances que nous traitons dans ce document ont été générées à partir des données liées au chromosome humain 1. Afin de créer des instances suffisamment difficiles, le seuil $\theta$ a été fixé à $0.5$. Chacune des $19,750$ instances correspond à une composante connexe du graphe de SNP obtenu. Nous avons sélectionné 13 instances parmi les plus difficiles pour nos expérimentations. Le tableau 4.4 présente leurs caractéristiques.

| Instance | $n$ | $d$ | $e$ | $S^*$ |
|---|---|---|---|---|
| #3792 | 528 | 59 | $12,084$ | $6,359,805$ |
| #4449 | 464 | 64 | $12,540$ | $5,094,256$ |
| #6835 | 496 | 90 | $18,003$ | $4,571,108$ |
| #8956 | 486 | 106 | $20,832$ | $6,660,308$ |
| #9319 | 562 | 58 | $14,811$ | $6,477,229$ |
| #15757 | 342 | 47 | $5,091$ | $2,278,611$ |
| #16421 | 404 | 75 | $12,138$ | $3,436,849$ |
| #16706 | 438 | 30 | $6,321$ | $2,632,310$ |
| #6858 | 992 | 260 | $103,056$ | $20,162,249$ |
| #9150 | $1,352$ | 121 | $44,217$ | $43,301,891$ |
| #10442 | 908 | 76 | $28,554$ | $21,591,913$ |
| #14226 | $1,058$ | 95 | $36,801$ | $25,665,437$ |
| #17034 | $1,142$ | 123 | $47,967$ | $38,318,224$ |

TABLE 4.4 – Caractéristiques des instances TagSNP.

## 4.5 Bases de données de l'UCI

Le tableau 4.5 présente les bases de transactions sélectionnées pour évaluer notre démarche présentée dans les chapitres 6 et 7.

### 4.5.1 Descriptions des bases de données

**Mushroom.** Ce jeu de données inclut des descriptions des échantillons hypothétiques qui correspondent à un ensemble de 23 espèces de champignons à lamelles faisant partie de la famille Agaricus et Lepiota. Chaque espèce est identifiée comme définitivement comestible, définitivement toxique, ou de comestibilité inconnue et non recommandée. La dernière classe est combinée avec la classe toxique. Chaque tuple de cette base de données décrit les différents champignons avec 22 attributs non binarisés, où chaque attribut peut prendre différentes valeurs catégoriques. Cette base de données est souvent utilisée dans l'apprentissage supervisé, mais qui peut être utilisée aussi pour l'apprentissage non-supervisé en ignorant la classe.

**Vote.** Cette base de données comprend des votes pour chacun des membres du Congrès de la Chambre des représentants des États-Unis. Neuf types de votes différents sont considérés : voté pour, apparié et annoncé pour (ces trois sont simplifiés pour être considérés comme un pour), a voté contre, a été contraint et a annoncé contre (ces trois sont simplifiés pour être considérés comme un contre), votés présents, votés présents pour éviter Conflit d'intérêts, et n'a pas voté ni fait autrement une position connue (ces trois sont simplifiés pour être considérés comme une disposition inconnue). Les attributs portent sur différents projets proposés pour adoption (immigration, dépenses d'éducation, etc.).

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

TABLE 4.5 – Caractéristiques des bases de données.

**Tic-Tac-Toe.** Cette base de données encode l'ensemble complet des configurations de cartes possibles à la fin des jeux tic-tac-toe, où "x" est supposé avoir joué en premier. Le concept cible est "gagner pour x" (c'est-à-dire vrai lorsque "x" comporte l'une des 8 façons possibles pour créer un trois dans une ligne). Les attributs correspondent aux occupations des cases de la carte par les joueurs.

**Zoo-1.** Une base de données simple contenant différents animaux décrits par différents attributs booléens les caractérisant (queue, domestique, prédateur, etc.). Chaque animal est affecté à une classe parmi 7 classes.

**Hepatitis.** En médecine, l'inflammation qui apparaît dans le foie est appelée hépatite (hepatitis). Elle peut être causée par des infections avec des virus, des bactéries, des champignons, l'exposition à des toxines telles que l'alcool et l'auto-immunité. L'hépatite endommage les cellules du foie et la tendresse, l'enflure et l'inflammation du foie sont des symptômes de cette maladie. Les attributs de la base de données décrivent différentes informations, telles que l'age, la fatigue, etc., pour chaque entrée de la base de données à une classe, qui peut prendre soit mort ou vivant.

## 4.6 Conclusion

Dans ce chapitre, nous avons commencé par présenter les différents problèmes réels dans différents domaines (RLFAP, SPOT5, TagSNP) que nous avons choisi pour évaluer nos contributions relatives aux modèles parallèles proposés basés sur DGVNS. Tous les problèmes ont été modélisés sous forme de réseaux de fonctions de coût exhibant un graphe de contraintes, qui est exploité pour révéler la structure du problème via la décomposition arborescente.

Pour étudier nos contributions dans la partie liée à l'extraction d'ensembles de motifs, nous avons présenté les différentes bases de données transactionnelles et leurs différentes caractéristiques (nombre de transactions, nombre d'items, densité de la base de données). Ces caractéristiques influent fortement sur la combinatoire liée à l'extraction d'ensembles de motifs.

Nous présentons dans le chapitre suivant nos contributions et les résultats sur l'exploitation du parallélisme dans DGVNS.

# Deuxième partie

# Contributions

# Chapitre 5

# Méthodes parallèles exploitant la décomposition arborescente

## Sommaire


**5.1 Exploitation parallèle des clusters de la décomposition arborescente . . . . . 76**
5.1.1 Modèle maître-travailleurs . . . . . 77
5.1.2 Modes de communication . . . . . 77
**5.2 VNS coopératif parallèle exploitant la décomposition arborescente . . . . . 78**
5.2.1 Algorithme du processus maître . . . . . 78
5.2.2 Algorithme du processus travailleur . . . . . 80
**5.3 Stratégies de réplication parallèles pour DGVNS . . . . . 80**
5.3.1 Réplication parallèle asynchrone de DGVNS . . . . . 81
5.3.2 Réplication parallèle synchrone de DGVNS . . . . . 82
5.3.3 Algorithme du processus travailleur . . . . . 82
**5.4 Intensification et diversification . . . . . 82**
5.4.1 Stratégies d'intensification . . . . . 82
5.4.2 Stratégies de diversification . . . . . 83
**5.5 Résultats expérimentaux . . . . . 84**
5.5.1 Apports de la parallélisation . . . . . 84
**5.6 Bilan . . . . . 88**


Exploiter le parallélisme et les architectures multi-processeurs est un moyen efficace pour accélérer les calculs dans de nombreux domaines d'applications. Au cours des deux dernières décennies, le calcul parallèle a connu un grand succès dans différents domaines scientifiques dont l'optimisation combinatoire, un domaine très prometteur. Plusieurs versions parallèles de nombreuses métaheuristiques bien connues ont été proposées avec différents degrés de parallélisme [Crainic et Toulouse, 1998, Crainic et Nourredine, 2005]. Une approche naturelle consiste à utiliser la stratégie de diviser-et-conquérir pour diviser l'espace de recherche en sous-espaces et explorer les différents sous-espaces en parallèle [Chu *et al.*, 2008]. Cependant, sur des problèmes combinatoires difficiles de grande taille, ces approches semblent inefficaces en raison des interactions non-locales complexes entre les variables et les contraintes. Par exemple, en SAT, la définition d'un mécanisme efficace pour le partage et la communication des clauses apprises par différents processus demeure un problème ouvert [Hutter *et al.*, 2012]. C'est ainsi que plusieurs approches proposent d'exécuter un portefeuille de solveurs en parallèle afin d'obtenir un solveur globalement meilleur, évitant ainsi tout type de communication [Hutter *et al.*, 2012, Kotthoff, 2012].

Plus récemment, Mathieu Fontaine a étudié dans sa thèse [Fontaine, 2013], les apports de la décomposition arborescente dans les méthodes de recherche locale, et plus particulièrement, au sein de la méthode de recherche à voisinage variable (Variable Neighborhood Search ou VNS) [Mladenovic et Hansen, 1997]. Il a proposé la méthode, nommée DGVNS (Decomposition Guided VNS) [Fontaine *et al.*, 2013], qui exploite les clusters issus de la décomposition arborescente d'un graphe de contraintes afin de guider l'exploration de voisinages au sein de `VNS`. Bien que `DGVNS` soit efficace sur certaines instances de taille conséquente, elle reste moins efficace quand ces dernières sont de grande taille. En effet, les performances de `DGVNS` décroissent significativement en raison du nombre important de clusters à considérer.

Nous étudions, dans ce chapitre, les apports du parallélisme au sein de DGVNS. Tout d'abord, nous présentons une première stratégie de parallélisation de `DGVNS`, appelée `CPDGVNS` (Cooperative Parallel `DGVNS`) [Ouali *et al.*, 2014a, Ouali *et al.*, 2014b], qui consiste à explorer en parallèle tous les clusters fournis par une décomposition arborescente. `CPDGVNS` repose sur une architecture *maître-travailleurs*, où le processus maître mémorise, met à jour, et communique la meilleure solution courante, alors que les processus travailleurs gèrent, individuellement, l'exploration des clusters. Les processus travailleurs coopèrent par une communication asynchrone en échangeant des informations sur la meilleure solution trouvée. Ce schéma de communication permet, d'une part, de garantir une gestion locale des clusters, et d'autre part, d'initialiser l'exploration de ces clusters avec des solutions variées, favorisant ainsi la diversification de la recherche. Dans `CPDGVNS`, chaque processus travailleur effectue une recherche `DGVNS` dans le cluster qui lui est assigné. La principale limite de cette approche est que les travailleurs doivent effectuer un certain nombre d'itérations avant de partager leurs meilleures solutions avec le maître. Cela rend la coopération avec le maître moins fréquente, et limite la diversification de l'exploration de l'espace de recherche par les travailleurs. Pour améliorer la diversification dans `CPDGVNS`, nous introduisons deux nouvelles stratégies de parallélisation (1) *Replicated Asynchronous* `DGVNS` (`RADGVNS`) et (2) *Replicated Synchronous* `DGVNS` (`RSDGVNS`) [Ouali *et al.*, 2015] permettant la production rapide de solutions intermédiaires pour alimenter l'échange d'informations entre les travailleurs. Des expérimentations réalisées sur des instances aléatoires (GRAPH) et sur plusieurs instances difficiles de trois problèmes réels (RLFAP, SPOT5, TagSNP), détaillés dans le chapitre 4, montrent l'apport du parallélisme par rapport à la méthode séquentielle.

La section 5.1 présente le modèle utilisé pour paralléliser l'exploration des clusters de la décomposition arborescente. La section 5.2 détaille notre première contribution CPDGVNS. La section 5.3 détaille notre deuxième contribution en proposant deux autres stratégies parallèles qui améliorent la diversification et la fréquence d'échange des solutions. Dans la section 5.4, nous montrons comment les stratégies de diversification et d'intensification sont mises en œuvre dans nos méthodes parallèles. Enfin dans la section 5.5, nous présentons une évaluation complète de nos différentes stratégies parallèles et nous les comparons à la version séquentielle.

## 5.1 Exploitation parallèle des clusters de la décomposition arborescente

La décomposition arborescente introduite par Robertson et Seymour [Robertson et Seymour, 1986] peut révéler des propriétés structurales intéressantes en décomposant un problème en sous-problèmes (appelés clusters) constituant un graphe acyclique. Chaque cluster correspond à un sous-ensemble de variables fortement liées dans le graphe initial. Une fois le graphe est décomposé, le temps de résolution du problème initial peut être limité par un exposant de sa largeur, qui est la taille du plus grand cluster dans l'arbre (moins 1). Cette propriété explique pourquoi la décomposition arborescente a suscité un grand intérêt dans divers domaines : pour la vérification de la satisfaction dans les problèmes SAT [Rish et Dechter, 2000], pour la résolution des problèmes de satisfaction de contraintes (CSP) [Dechter et Pearl, 1989], dans des réseaux bayésiens ou probabilistes [Marinescu et Dechter, 2009], dans des bases de données relationnelles [Gottlob *et al.*, 2012], pour l'optimisation sous contraintes [de Givry *et al.*, 2006, Sánchez *et al.*, 2009, Jégou et Terrioux, 2003]. Toutes ces propositions exploitent la décomposition arborescente dans le contexte des méthodes de résolution complètes.

L'exploitation de la décomposition arborescente au sein de DGVNS a conduit à des améliorations très significatives comparée à la méthode VNS/LDS+CP, en guidant efficacement l'exploration des voisinages au sein de la méthode VNS. Cependant, sur des problèmes de (très) grande taille, les performances de la méthode peuvent décroître significativement en raison du nombre important de clusters à considérer. Paralléliser l'exploration des différents clusters semble donc une idée naturelle et prometteuse.

Le modèle de parallélisme *maître-travailleurs* a attiré l'attention de beaucoup de chercheurs dans différents domaines d'application importants tels que l'ordonnancement parallèle, le transport, les tests de semi-conducteurs, etc. Plusieurs métaheuristiques parallèles exploitent ce type de modèle. Nous nous appuyons également sur ce modèle pour paralléliser l'exploration des clusters de la décomposition arborescente. La tâche de chaque processus travailleur est d'explorer un cluster de la décomposition en utilisant une instance modifiée de DGVNS. La coordination des différents processus travailleurs est assurée par un processus maître.

### 5.1.1 Modèle maître-travailleurs

La principale motivation derrière le parallélisme est d'améliorer les performances des algorithmes de résolution en temps de calcul et en qualité des solutions. Plusieurs études sur les métaheuristiques parallèles confirment cette tendance. Les divers travaux sur les métaheuristiques parallèles indiquent également que, bien que les méthodes basées sur des stratégies de recherche indépendantes donnent de bons résultats, elles sont généralement surclassées par les stratégies de recherche coopératives (voir chapitre 2).

Les métaheuristiques parallèles proposées dans la littérature reposent sur différents modèles, le plus répandu est le modèle *maître-travailleurs*. Ce modèle est basé sur deux types d'entités : le premier est le maître, l'unité qui contrôle la communication sur tout l'ensemble des unités et conserve l'information à caractère centralisée. La deuxième unité, appelée travailleur, effectue les tâches de base et communique les résultats au maître. C'est ce modèle que nous avons adopté pour nos différentes stratégies parallèles dont l'objectif est d'explorer simultanément l'espace de recherche par des travailleurs coopérants. Les stratégies que nous proposons explorent en parallèle tous les clusters fournis par la décomposition arborescente, mais se différencient par la manière de gérer les voisinages et par le mode de communication entre les différents processus. Les principales tâches accomplies par chaque entité des trois stratégies peuvent être résumées comme suit :

- **Processus maître :**
  — envoi à chaque processus travailleur de la solution initiale et du cluster à explorer ;
  — réception de la meilleure solution d'un processus travailleur, et mise à jour de la meilleure solution globale ;
  — vérification de la condition d'arrêt et relance d'un processus travailleur disponible sur un autre voisinage à partir de la meilleure solution globale.
- **Chaque processus travailleur :**
  — reçoit la solution initiale et gère l'exploration du cluster affecté par le processus maître ;
  — communique sa solution au processus maître.

### 5.1.2 Modes de communication

Les processus travailleurs coopèrent en échangeant des informations sur la meilleure solution trouvée. Un élément important dans la conception des schémas coopératifs parallèles est la détermination à la fois de quand importer une solution externe et de comment l'exploiter dans les différents processus travailleurs (par exemple, redémarrer de la solution importée pour biaiser les mécanismes de sélection locaux). De plus, la fréquence d'échange de solutions entre les processus joue un rôle très important sur la détermination de la nature de la méthode parallèle. Un échange très fréquent donne un caractère plus diversifié à l'exploration alors que si l'échange est moins fré-

quent, l'exploration sera plus intensive, puisque chaque processus continue seul dans sa direction et ne redémarre sur une nouvelle solution qu'après un certain temps.

Dans le modèle adopté, les processus travailleurs communiquent exclusivement avec le processus maître. Les mises à jour des solutions et les communications s'effectuent de manière synchrone et asynchrone.

1. **Synchrone :** Le processus maître doit attendre la réception de toutes les solutions des processus travailleurs pour gérer le changement de voisinage pour le prochain mouvement.
2. **Asynchrone :** Le processus maître traite immédiatement et individuellement le changement de voisinage quand une solution est reçue d'un processus travailleur.

Les deux modes de communication offrent différents avantages, le premier mode donne la capacité de gérer globalement le changement de voisinage sur tout l'ensemble des processus travailleurs, tandis que le deuxième mode est plus orienté vers une gestion locale par rapport à un processus. En conséquence, les processus travailleurs peuvent démarrer leur nouvelle recherche à partir de solutions initiales différentes.

Dans ce cadre, nous avons proposé différentes stratégies de parallélisation de la méthode DGVNS. Ces contributions se résument aux points suivants :

1. Proposition d'une première stratégie de parallélisation de DGVNS nommée *Cooperative parallel* DGVNS (CPDGVNS) [Ouali *et al.*, 2014a] basée sur une architecture maître-travailleurs (cf Section 5.2),
2. Proposition de deux autres stratégies de parallélisation nommées *Replicated Asynchronous DGVNS* (RADGVNS) et *Replicated Synchronous DGVNS* (RSDGVNS) [Ouali *et al.*, 2015] destinées à améliorer la diversification de CPDGVNS (cf Section 5.3).

## 5.2 VNS coopératif parallèle exploitant la décomposition arborescente

L'idée de CPDGVNS consiste simplement à explorer tous les clusters fournis par la décomposition arborescente en parallèle. La parallélisation repose sur le modèle *maître-travailleurs*, où le processus maître mémorise, met à jour et communique la meilleure solution courante, et les processus travailleurs, qui se comportent comme des instances modifiées de DGVNS, gèrent de façon individuelle l'exploration des voisinages sur le cluster affecté par le processus maître. Autrement dit, un processus travailleur change de voisinage en fonction de la qualité de la solution locale, qui est indépendante des solutions du reste des processus. Les processus individuels coopèrent de façon *asynchrone* en échangeant des informations sur la meilleure solution courante. Ceci garantit l'indépendance des processus travailleurs individuels et permet de démarrer à partir de plusieurs solutions initiales différentes, favorisant ainsi une meilleure diversification.

### 5.2.1 Algorithme du processus maître

Soit $ncl = |\mathcal{C}_T|$ le nombre de clusters et $npr$ le nombre de processus travailleurs utilisés. Pour profiter pleinement de la parallélisation, nous avons fixé $npr$ [21] au nombre de clusters. Les processus travailleurs sont numérotés de 1 à $ncl$, alors que le processus maître sera désigné par l'identifiant zéro. L'algorithme 19 décrit le pseudo-code du processus maître. Il démarre du graphe de clusters du réseau de contraintes et d'une solution initiale s générée aléatoirement (ligne 2). Il procède en trois étapes :

21. Si le nombre de cœurs est inférieur au nombre de clusters, un même cœur sera utilisé pour traiter différents clusters.

**Algorithme 19 :** Pseudo-code du processus maître de CPDGVNS

**Entrées** : Le réseau de contraintes $G = (X, W)$ et une décomposition arborescente ($\mathcal{C}_T$,$T$) de $G$ avec $\mathcal{C}_T = \{C_1, C_2, \ldots, C_{ncl}\}$, $k_{init}$, $\delta_{max}$, $npr$

1 **début**
2 $s \leftarrow$ `genSolInit(`$X, W$`)` ;
3 $c \leftarrow 1$ ;
4 **pour tous les** $travailleur\ p = 1, \ldots, npr$ **faire**
5 $k_{max} \leftarrow |C_c|$ ;
6 `Send`$(p, c, k_{init}, k_{max}, \delta_{max}, s)$;
7 $c \leftarrow succ(c)$ ;
8 $Finished \leftarrow 0, adj \leftarrow 0$ ;
9 **tant que** $(Finished < npr)$ **faire**
10 `Receive`$(p, s'_p)$ ;
11 `MasterNeighbor`$(s, s'_p, c, adj, k_{max})$ ;
12 **si** $(not\ global_TimeOut)$ **alors**
13 `Send(`$p, c, k_{init}, k_{max}, \delta_{max}, s$`)` ;
14 **sinon**
15 $Finished \leftarrow Finished + 1$ ;
16 **retourner** $s$
17 `Procédure MasterNeighbor (`$s, s', c, adj, k_{max}$`)` ;
18 **début**
19 **si** $(f(s') < f(s))$ **alors**
20 $s \leftarrow s', adj \leftarrow 0$ ;
21 $c \leftarrow succ(c), k_{max} \leftarrow |C_c|$ ;
22 **sinon**
23 $c \leftarrow succ(c), k_{max} \leftarrow |C_c|$ ;
24 $adj \leftarrow adj + 1$ ;
25 $C_s \leftarrow$ `Adjacents`$(C_c, adj)$ ;
26 $k_{max} \leftarrow k_{max} + |C_s|$ ;

#### 5.2.1a Étape d'initialisation

Le maître initie la recherche en lançant l'exécution en parallèle de $npr$ processus travailleurs (lignes 4-7). Ceci est réalisé en envoyant à chaque processus travailleur $p$ la même solution initiale, le cluster associé $C_c$ de $\mathcal{C}_T$ et les valeurs des paramètres $k_{init}$, $k_{max}$ et $\delta_{max}$ (ligne 6). La liste $\mathcal{C}_T$ des clusters est gérée en FIFO pour assurer que tout cluster soit traité par un seul processus travailleur (ligne 7). La valeur de $k_{max}$ est initialisée à la taille du cluster affecté au processus travailleur (ligne 5). Ceci restreint le choix des variables à désaffecter uniquement aux variables de ce cluster.

#### 5.2.1b Étape de mise à jour

Durant cette phase, le maître attend la meilleure solution trouvée par chaque processus travailleur (ligne 10). Soit $s'_p$ la meilleure solution communiquée par le processus travailleur $p$ au maître. Si $s'_p$ est de meilleure qualité que s (ligne 19), $s'_p$ devient la meilleure solution globale, le cluster suivant $C_c$ est considéré et la valeur de $k_{max}$ est réinitialisée à $|C_c|$ (lignes 20-21). Sinon, on cherche de nouvelles améliorations dans le cluster suivant $C_c$ et on incrémente de un le nombre de clusters adjacents $adj$ qui doivent être considérés ainsi que la valeur de $k_{max}$ par le cardinal de l'union des clusters $C_j$ adjacents à $C_c$ (avec $j = 1..adj$) (lignes 23-26). L'augmentation de $k_{max}$ permettra d'élargir l'ensemble de variables candidates à désaffecter par les processus travailleurs. Comme indiqué précédemment, CPDGVNS utilise des instances modifiées de DGVNS pour la recherche parallèle, ces instances

utilisent la topologie du graphe de clusters afin d'augmenter la taille du voisinage. De plus, si le nombre de clusters $|C_T|$ est largement supérieure au nombre de processus $npr$, le processus maître affecte d'une manière cyclique les clusters aux processus travailleurs libres tout en respectant la stratégie FIFO. Enfin, l'algorithme termine la recherche quand le TimeOut_global est atteint (ligne 12), et envoie la meilleure solution trouvée par tous les processus travailleurs (ligne 16).

**Algorithme 20 :** Pseudo-code du processus travailleur de CPDGVNS

**Entrées** : Le réseau de contraintes $G = (X, W)$ et une décomposition arborescente ($\mathcal{C}_T$,$T$) de $G$, avec $\mathcal{C}_T = \{C_1, C_2, \ldots, C_{ncl}\}$

1 **début**
2 `Receive(`$0, c, k_{init}, k_{max}, \delta_{max}, s$`)` ;
3 $s_p \leftarrow s$ ;
4 $k \leftarrow k_{init}$ ;
5 **tant que** $(k < k_{max}) \wedge (not\ local_TimeOut)$ **faire**
6 $C_s \leftarrow$ `CompleteCluster(`$C_c, k$`)` ;
7 $X_{un} \leftarrow$ `Hneighborhood(`$N_{k,c}, C_s, W, s_p$`)` ;
8 $\mathcal{A} \leftarrow s_p \backslash \{(x_i, a) \mid x_i \in \mathcal{X}_{un}\}$ ;
9 $s'_p \leftarrow LDS + CP(\mathcal{A}, X_{un}, \delta_{max}, f(s_p), s_p)$ ;
10 `NeighbourhoodChange(`$s_p, s'_p, k$`)` ;
11 `Send(`$0, s_p$`)` ;

#### 5.2.1c Étape de coopération

Le but de cette étape est de faire coopérer les processus travailleurs en utilisant la meilleure solution globale disponible au niveau du maître. Pour tout processus $p$ disponible, le processus maître lance ce dernier avec la meilleure solution potentiellement trouvée par un autre processus (lignes 12-13). Si $global_TimeOut$ n'est pas atteint, on relance le processus travailleur $p$ à partir de la meilleure solution globale disponible. Sinon, il est arrêté. L'ensemble du processus de résolution se termine lorsque tous les processus travailleurs ont terminé (ligne 9).

### 5.2.2 Algorithme du processus travailleur

L'algorithme 20 décrit le pseudo-code du processus travailleur $p$. Il suit le schéma de DGVNS modifié, où le changement de cluster est géré par le maître. Par ailleurs, la stratégie de changement de voisinage est similaire à celle de la méthode VNS/LDS+CP (voir chapitre 2). Il nécessite une décomposition arborescente ($\mathcal{C}_T$,$T$) de $G$. Il reçoit du maître, l'indice $c$ du cluster qui lui est assigné, les valeurs des paramètres $k_{init}$, $k_{max}$, $\delta_{max}$, et la solution initiale s (ligne 2). L'algorithme ne communique sa solution que s'il atteint la dimension maximale du voisinage $k_{max}$ ou le $local_TimeOut$ (ligne 5).

## 5.3 Stratégies de réplication parallèles pour DGVNS

Dans CPDGVNS, les clusters de la décomposition arborescente sont explorés en parallèle par chaque processus travailleur. Chacun effectue une recherche DGVNS dans le cluster qui lui est assigné. La principale limite de cette approche est que les travailleurs doivent effectuer un certain nombre d'itérations avant de partager leurs meilleures solutions avec le maître. Cela rend la coopération avec le maître moins fréquente, et limite la diversification de l'exploration de l'espace de recherche par les processus travailleurs.

**Algorithme 21 :** Pseudo-code du processus maître de RADGVNS.

**Entrées** : Le réseau de contraintes $G = (X, W)$ et une décomposition arborescente ($\mathcal{C}_T$,$T$) de $G$, avec $\mathcal{C}_T = \{C_1, C_2, \ldots, C_{ncl}\}$

1 **début**
2 $s \leftarrow$ `genSolInit` $(X, W)$ ;
3 $c \leftarrow 1$ ;
4 **pour tous les** $travailleur\ p = 1, \ldots, npr$ **faire**
5 $Pr[p].k \leftarrow k_{init}$, $Pr[p].cl \leftarrow c$ ;
6 `Send` $(p, Pr[p], \delta_{max}, s)$ ;
7 $c \leftarrow succ(c)$ ;
8 $Finished \leftarrow 0$ ;
9 **tant que** $(Finished < npr)$ **faire**
10 `Receive` $(p, s'_p)$ ;
11 `RA-Neighbor` $(p, c, k_{init}, s'_p, s)$ ;
12 **si** $(not\ TimeOut)$ **alors**
13 `Send` $(p, Pr[p], \delta_{max}, s)$ ;
14 **sinon**
15 $Finished \leftarrow Finished + 1$ ;
16 **retourner** $s$
17 `Procédure RA-Neighbor` $(p, c, k_{init}, s', s)$ ;
18 **début**
19 **si** $(f(s') < f(s))$ **alors**
20 $s \leftarrow s'$, $Pr[p].k \leftarrow k_{init}$ ;
21 **sinon si** $(Pr[p].k < |X|)$ **alors**
22 $Pr[p].k \leftarrow Pr[p].k + 1$ ;
23 $Pr[p].cl \leftarrow c$, $c \leftarrow succ(c)$ ;

Pour améliorer la diversification dans CPDGVNS, nous proposons deux nouvelles stratégies de parallélisation nommées respectivement *Replicated Asynchronous DGVNS* (RADGVNS) et *Replicated Synchronous DGVNS* (RSDGVNS) permettant la production rapide de solutions intermédiaires pour alimenter l'échange d'informations entre les travailleurs. Ces stratégies offrent une meilleure exploration des voisinages sur tout l'ensemble de processus. Contrairement à CPDGVNS, le processus maître a une visibilité continue sur l'amélioration de la solution courante par les différents processus travailleurs, en conséquence, l'étape d'intensification et de diversification (voir section 5.4) sera plus renforcée en orientant la recherche vers des voisinages plus prometteurs et selon la solution globale.

### 5.3.1 Réplication parallèle asynchrone de DGVNS

Comme pour CPDGVNS, les processus travailleurs dans RADGVNS coopèrent de manière *asynchrone* en échangeant des informations sur la meilleure solution courante. Toutefois, et contrairement à CPDGVNS, chaque processus travailleur exécute une seule itération d'une recherche locale, en complétant une solution partielle dans le cluster qui lui est associé. Soit $Pr$ une structure de données permettant d'enregistrer la dimension du voisinage $k$ ainsi que l'indice du cluster $cl$ assigné à chaque processus travailleur. L'algorithme 21 décrit le pseudo-code du processus maître. Premièrement, le maître initie la recherche en lançant l'exécution en parallèle de $npr$ processus travailleurs (lignes 4-7). Ensuite, il attend les nouvelles solutions trouvées par chaque processus travailleur (lignes 9-13). Si une nouvelle solution $s'_p$ est reçue, elle est comparée à la meilleure solution globale courante s. Mais contrairement à CPDGVNS, en cas d'amélioration, s est mise à jour à $s'$, $k$ à $k_{init}$ et le prochain cluster est considéré. Sinon, le maître cherche de nouvelles améliorations dans $N_{(k+1),succ(i)}$ (voir la procédure `RA-Neighbor`, ligne 11). Notons que le changement de cluster dans RADGVNS s'effectue plus rapidement (i.e. à chaque fois

qu'une solution de moins bonne qualité est renvoyée par le travailleur), tandis que CPDGVNS nécessite beaucoup plus de temps avant de changer le cluster.

### 5.3.2 Réplication parallèle synchrone de DGVNS

Pour étudier l'impact de la synchronisation, nous avons proposé, sur le même modèle, une version synchrone de la communication entre le processus maître et les processus travailleurs. Dans RSDGVNS, le processus maître attend la fin de tous les processus travailleurs avant d'effectuer le prochain mouvement. À l'exception du changement systématique des clusters pour chaque processus travailleur, cette stratégie ne modifie pas le comportement de la méthode séquentielle DGVNS : à chaque itération de la recherche, plusieurs voisinages sont explorés en parallèle en utilisant la même solution courante contrairement à RADGVNS, qui maintient potentiellement différentes solutions pour relancer l'ensemble des processus travailleurs.

L'algorithme 22 illustre le pseudo code de RSDGVNS. L'étape d'initialisation (lignes 5-8) est incluse dans la boucle de recherche. Après chaque lancement, le processus maître récupère la meilleure solution $s'$ trouvée par tous les processus travailleurs, cette solution est appelée solution intermédiaire (lignes 11-14). Si la solution intermédiaire est de meilleure qualité que la solution courante s, alors s sera mise à jour par $s'$, la valeur de $k$ sera réinitialisée par $k_{init}$ (lignes 19-20). Sinon, tous les processus travailleurs seront réinitialisés sur des voisinages de taille $k + 1$ (ligne 22).

### 5.3.3 Algorithme du processus travailleur

Chaque processus travailleur exécute une seule itération d'une recherche locale, en complétant une solution partielle dans le cluster qui lui est associé. L'algorithme 23 détaille le pseudo-code du processus travailleur. Notons ici que la fonction `Hneighborhood` assure que tous les sous-ensembles $X_{un}$ de $k$ variables à désaffecter doivent être différents pour tous les processus travailleurs.

## 5.4 Intensification et diversification

L'efficacité de toute métaheuristique dépend principalement des mécanismes d'intensification et de diversification mis en œuvre. Ainsi toute "bonne" métaheuristique doit pouvoir générer de nouvelles solutions qui peuvent généralement être plus susceptibles d'améliorer les solutions précédentes et aussi être en mesure de couvrir les parties les plus importantes de l'espace de recherche où l'optimum global peut être localisé. Par ailleurs, elle doit pouvoir échapper à tout optimum local. Un bon compromis équilibrant entre l'intensification et la diversification durant la recherche peut conduire à des métaheuristiques de très bonne efficacité.

### 5.4.1 Stratégies d'intensification

L'intensification dans toutes les stratégies parallèles proposées est réalisée en continuant l'exploration dans chaque processus travailleur $p$ avec la meilleure solution globale disponible dans le processus maître. De plus, quand la méthode trouve une nouvelle solution de meilleure qualité que la solution courante, la taille du voisinage est réinitialisée (réduite) à la taille initiale. Les deux stratégies d'intensification permettent de concentrer la recherche dans une région locale en exploitant l'information qu'une meilleure solution courante a été trouvée dans cette région, et que potentiellement, d'autres solutions de bonne qualité y existent.

**Algorithme 22 :** Pseudo-code du processus maître de RSDGVNS.

**Entrées** : Le réseau de contraintes $G = (X, W)$ et une décomposition arborescente ($\mathcal{C}_T$,$T$) de $G$, avec $\mathcal{C}_T = \{C_1, C_2, \ldots, C_{ncl}\}$

1 **début**
2 $s \leftarrow$ `genSolInit` $(X, W)$ ;
3 $s' \leftarrow s$, $k \leftarrow k_{init}$, $c \leftarrow 1$ ;
4 **tant que** $((k < |X|) \wedge (not\, TimeOut))$ **faire**
5 **pour chaque** $slave\ p = 1, \ldots, npr$ **faire**
6 $Pr[p].k \leftarrow k$, $Pr[p].cl \leftarrow c$ ;
7 `Send` $(p, Pr[p], \delta_{max}, s)$ ;
8 $c \leftarrow succ(c)$ ;
9 $Finished \leftarrow 0$ ;
10 **tant que** $(Finished < npr)$ **faire**
11 `Receive` $(p, s''_p)$ ;
12 **si** $(f(s''_p) < f(s'))$ **alors**
13 $s' \leftarrow s''_p$ ;
14 $Finished \leftarrow Finished + 1$ ;
15 `RS-Neighbor` $(k, k_{init}, s, s')$ ;
16 **retourner** $s$ ;
17 `Procédure RS-Neighbor` $(p, c, k_{init}, s', s)$ ;
18 **début**
19 **si** $(f(s') < f(s))$ **alors**
20 $s \leftarrow s'$, $k \leftarrow k_{init}$ ;
21 **sinon**
22 $k \leftarrow k + 1$ ;

**Algorithme 23 :** Pseudo-code du processus travailleur

**Entrées** : Le réseau de contraintes $G = (X, W)$ et Une décomposition arborescente ($\mathcal{C}_T$,$T$) de $G$, avec $\mathcal{C}_T = \{C_1, C_2, \ldots, C_{ncl}\}$

1 **début**
2 `Receive` $(0, P, \delta_{max}, s)$ ;
3 $C_s \leftarrow$ `CompleteCluster` $(P.cl, P.k)$ ;
4 $X_{un} \leftarrow$ `Hneighborhood` $(N_{P.k,P.cl}, C_s, W, s)$ ;
5 $\mathcal{A} \leftarrow s \backslash \{(x_i, a) \mid x_i \in \mathcal{X}_{un}\}$ ;
6 $s' \leftarrow LDS + CP(\mathcal{A}, X_{un}, \delta_{max}, f(s), s)$ ;
7 `Send` $(0, s')$ ;

### 5.4.2 Stratégies de diversification

Contrairement à l'étape d'intensification, la diversification est réalisée en changeant systématiquement pour chaque processus travailleur $p$ le cluster courant par un autre cluster non couramment exploré. De plus, quand un processus travailleur $p$ n'arrive pas à améliorer la meilleure solution courante, la taille du voisinage sera augmentée afin d'explorer d'autres régions de l'espace de recherche et de générer des solutions plus diverses.

## 5.5 Résultats expérimentaux

Les expérimentations ont été réalisées sur les instances les plus difficiles des problèmes RLFAP, SPOT5, Tag-SNP et GRAPH, soit un total de 15 instances. Un ensemble de 50 essais par instance a été réalisé sur un cluster Linux Infiniband de 8 nœuds, localisé au centre de calcul intensif du Cerist [22]. Chaque nœud est constitué d'un dual-CPU Xeon E5-2650 de 16 cœurs, ayant 64 Go de RAM et une fréquence de 2.00 GHz.

Toutes les méthodes ont été implantées en C++ en utilisant la librairie `toulbar2`. La parallélisation a été mise en œuvre dans l'environnement MPI (Message Passing Interface). Quatre méthodes ont été comparées : DGVNS, CPDGVNS, RADGVNS et RSDGVNS. Pour chaque méthode, différentes décompositions arborescentes ont été considérées :

(i) décomposition arborescente obtenue par l'heuristique MCS et sa version raffinée obtenue par fusion de clusters avec l'absorption (nommée MCS-$abs$)

(ii) décomposition arborescente obtenue par l'heuristique MINFILL et sa version raffinée obtenue par fusion de clusters avec l'absorption (nommée MINFILL-$abs$)

Pour nos stratégies parallèles, le nombre de processus $npr$ est fixé à $ncl$. Si le nombre de clusters $ncl$ dépasse 128, $npr$ est alors fixé à 128 (i.e. le nombre maximum de processus disponibles). Pour CPDGVNS, le temps d'exécution limite $local_TimeOut$ alloué à chaque processus travailleur a été fixé à $(global_TimeOut)/npr$.

### 5.5.1 Apports de la parallélisation

#### 5.5.1a Temps CPU

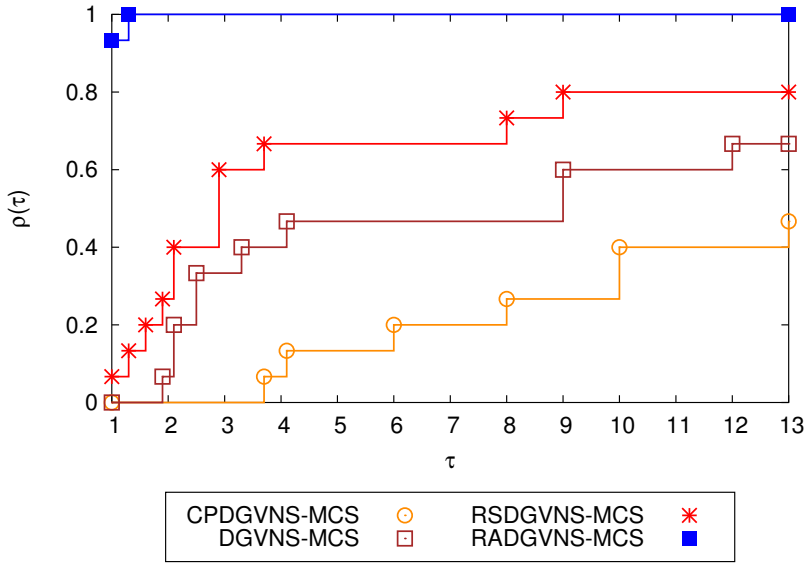
La figure 5.1a compare les courbes des profiles de performances des rapports de temps de DGVNS, CPDGVNS, RADGVNS et RSDGVNS sur les décompositions initiales de MCS (voir chapitre 4 pour le calcul des valeurs de $\tau$ et $\rho$). Premièrement, sur les 15 instantes considérées, RADGVNS et RSDGVNS dominent clairement DGVNS. Deuxièmement, la courbe du profil de performance de RADGVNS est systématiquement au-dessus de toutes les autres courbes. Cela signifie, que pour toute valeur de $\tau \geq 1$, RADGVNS résout plus de problèmes que n'importe quelle autre méthode.

De plus, en examinant la valeur de $\rho(1)$ (cf. colonne 2 de la figure 5.1b), on peut remarquer que RADGVNS est la méthode la plus rapide sur $93\%$ des instances contre $7\%$ pour RSDGVNS. Pour un facteur $\tau$ de $1.3$ de la meilleure méthode, RADGVNS parvient à résoudre $100\%$ les instances. Troisièmement, RSDGVNS est la seconde meilleure méthode. Elle surclasse nettement CPDGVNS et DGVNS. Les mauvaises performances de CPDGVNS par rapport à DGVNS peuvent probablement s'expliquer par le fait que les travailleurs restent dans le même cluster, même si aucune amélioration n'est obtenue, ce qui n'est pas le cas pour DGVNS qui change systématiquement de cluster. Cela limite grandement la diversification. En outre, comme les clusters se chevauchent fortement, CPDGVNS peut passer beaucoup plus de temps à essayer d'améliorer la solution sans succès.

#### 5.5.1b Robustesse

Une autre caractéristique importante qui peut être tirée à partir des courbes de la figure 5.1a est la mesure de robustesse, exprimée par le rapport $\tau_{min}/\tau_j^*$ (cf. colonne 3 de la figure 5.1b) : l'algorithme le plus robuste est celui ayant un ratio de 1. $\tau_j^*$ désigne la valeur de $\tau$ tel que $\rho_{m_j}(\tau^*) = 1$ et $\tau_{min}$ la valeur minimale de $\tau_j^*$ pour toutes les méthodes de $\mathcal{M}$. Clairement, RADGVNS est la méthode la plus robuste car elle résout toutes les instances avec le plus petit facteur. Ces résultats démontrent clairement l'intérêt d'échanger plus fréquemment les solutions intermédiaires avec le maître afin de favoriser la diversification de l'exploration de l'espace de recherche par les travailleurs.

22. `http://www.rx-racim.cerist.dz/?page_id=26`




(a) Temps CPU.

| | Temps | | |
|---|---|---|---|
| Méthode | $AUC$ | $\rho(1)$ | $\tau_{min}/\tau^*$ |
| DGVNS-MCS | 5.43 | 0.0 | 0.0004 |
| CPDGVNS-MCS | 2.41 | 0.0 | 0.0003 |
| RSDGVNS-MCS | 7.77 | 0.07 | 0.035 |
| RADGVNS-MCS | 11.97 | 0.93 | 1 |

(b) Critères de performance.

FIGURE 5.1 – Comparaison des profils de performances des rapports de temps de DGVNS, CPDGVNS, RSDGVNS et RADGVNS sur les décompositions initiales de MCS.

#### 5.5.1c Accélération

En complément des résultats, en temps CPU des quatre méthodes, présentés dans le tableau 5.1, les figures 5.2 et 5.3 montrent une autre vue des résultats portant sur les rapports de vitesse observés. Sur les instances TagSNP (cf. figure 5.2), RADGVNS est en moyenne 3.6 fois plus rapide que DGVNS. Sur l'instance la plus difficile #9150, RADGVNS est un ordre de grandeur plus rapide que DGVNS (speed-up de 11.93). Par rapport à RSDGVNS, les rapports de vitesse diminuent légèrement (cf. figure 5.3), mais RADGVNS est toujours plus rapide (speed-up allant jusqu'à 8.54). Sur les instances RLFAP et GRAPH, RADGVNS est jusqu'à deux ordres de grandeur plus rapide que DGVNS (speed-up de 203). Comparé à RSDGVNS, les rapports de vitesse de RADGVNS restent importants (speed-up allant jusqu'à 36). Enfin, sur les trois instances de SPOT5, RADGVNS et RSDGVNS présentent des speed-ups super-linéaires par rapport à DGVNS, avec un léger avantage pour RADGVNS.

#### 5.5.1d Impact de la décomposition arborescente

Pour évaluer l'intérêt d'exploiter des décompositions raffinées au sein de RADGVNS, différentes décompositions sont considérées : celles obtenues par MCS et MINFILL et leurs versions raffinées MCS-$abs$ et MINFILL-

| Instance | $ncl$ | Method | Succ. | Time |
|---|---|---|---|---|
| #10442, $n = 908$, $d = 76$, $e = 28,554$, $S^* = 21,591,913$ | 333 | DGVNS-MCS | 50/50 | 199.09 |
| | | CPDGVNS-MCS | 37/50 | 1401.69 |
| | | RSDGVNS-MCS | 50/50 | 226.51 |
| | | RADGVNS-MCS | **50/50** | **89.81** |
| #12976, $n = 920$, $d = 128$, $e = 44,823$, $S^* = 21,604,644$ | 332 | DGVNS-MCS | 50/50 | 1486.44 |
| | | CPDGVNS-MCS | 4/50 | 3493.51 |
| | | RSDGVNS-MCS | 10/50 | 3399.04 |
| | | RADGVNS-MCS | **50/50** | **455.86** |
| #14007, $n = 1,554$, $d = 100$, $e = 54,753$, $S^* = 50,290,563$ | 574 | DGVNS-MCS | 50/50 | 1556.80 |
| | | CPDGVNS-MCS | 0/50 | 3600.00 |
| | | RSDGVNS-MCS | 50/50 | 1073.35 |
| | | RADGVNS-MCS | **50/50** | **391.54** |
| #14226, $n = 1,058$, $d = 95$, $e = 36,801$, $S^* = 25,665,437$ | 409 | DGVNS-MCS | 50/50 | 285.31 |
| | | CPDGVNS-MCS | 33/50 | 1773.20 |
| | | RSDGVNS-MCS | 50/50 | 160.99 |
| | | RADGVNS-MCS | **50/50** | **143.23** |
| #17034, $n = 1,142$, $d = 123$, $e = 47,967$, $S^* = 38,318,224$ | 409 | DGVNS-MCS | 50/50 | 545.72 |
| | | CPDGVNS-MCS | 27/50 | 2251.72 |
| | | RSDGVNS-MCS | 50/50 | 448.38 |
| | | RADGVNS-MCS | **50/50** | **226.15** |
| #6858, $n = 992$, $d = 260$, $e = 103,056$, $S^* = 20,162,249$ | 272 | DGVNS-MCS | 50/50 | 1789.59 |
| | | CPDGVNS-MCS | 0/50 | 3600.00 |
| | | RSDGVNS-MCS | 50/50 | 1486.54 |
| | | RADGVNS-MCS | **50/50** | **907.39** |
| #9150, $n = 1,352$, $d = 121$, $e = 44,217$, $S^* = 43,301,891$ | 520 | DGVNS-MCS | 26/50 | 3000.98 |
| | | CPDGVNS-MCS | 0/50 | 3600.00 |
| | | RSDGVNS-MCS | 50/50 | 2146.88 |
| | | RADGVNS-MCS | **50/50** | **251.37** |
| #9319, $n = 562$, $d = 58$, $e = 14,811$, $S^* = 6,477,229$ | 252 | DGVNS-MCS | 50/50 | 38.00 |
| | | CPDGVNS-MCS | 40/50 | 822.87 |
| | | RSDGVNS-MCS | **50/50** | **20.60** |
| | | RADGVNS-MCS | 50/50 | 26.19 |

(a) TagSNP instances

| Instance | $ncl$ | Method | Succ. | Time |
|---|---|---|---|---|
| #414, $n = 364$, $d = 4$, $e = 10,108$, $S^* = 38,478$ | 99 | DGVNS-MCS | 33/50 | 1268.00 |
| | | CPDGVNS-MCS | 1/50 | 3529.57 |
| | | RSDGVNS-MCS | 50/50 | 18.89 |
| | | RADGVNS-MCS | **50/50** | **6.77** |
| #505, $n = 240$, $d = 4$, $e = 2,242$, $S^* = 21,253$ | 93 | DGVNS-MCS | 15/50 | 2520.79 |
| | | CPDGVNS-MCS | 0/50 | 3600.00 |
| | | RSDGVNS-MCS | 50/50 | 1.29 |
| | | RADGVNS-MCS | **50/50** | **0.88** |
| #509, $n = 348$, $d = 4$, $e = 8,624$, $S^* = 36,446$ | 98 | DGVNS-MCS | 27/50 | 1674.13 |
| | | CPDGVNS-MCS | 2/50 | 3457.12 |
| | | RSDGVNS-MCS | 50/50 | 11.00 |
| | | RADGVNS-MCS | **50/50** | **3.00** |
| Graph11, $n = 340$, $d = 44$, $e = 3,417$, $S^* = 3080$ | 195 | DGVNS-MCS | 42/50 | 604.61 |
| | | CPDGVNS-MCS | 50/50 | 30.51 |
| | | RSDGVNS-MCS | 50/50 | 308.45 |
| | | RADGVNS-MCS | **50/50** | **8.57** |
| Graph13, $n = 458$, $d = 44$, $e = 4,815$, $S^* = 10,110$ | 256 | DGVNS-MCS | 49/50 | 175.81 |
| | | CPDGVNS-MCS | 49/50 | 126.15 |
| | | RSDGVNS-MCS | 50/50 | 44.12 |
| | | RADGVNS-MCS | **50/50** | **21.20** |
| Scen07, $n = 200$, $d = 44$, $e = 2,665$, $S^* = 343,592$ | 111 | DGVNS-MCS | 48/50 | 470.01 |
| | | CPDGVNS-MCS | 0/50 | 3600.00 |
| | | RSDGVNS-MCS | 50/50 | 886.16 |
| | | RADGVNS-MCS | **50/50** | **54.10** |
| Scen08, $n = 458$, $d = 44$, $e = 5,286$, $S^* = 262$ | 258 | DGVNS-MCS | 9/50 | 3001.23 |
| | | CPDGVNS-MCS | 34/50 | 1452.74 |
| | | RSDGVNS-MCS | 50/50 | 205.30 |
| | | RADGVNS-MCS | **50/50** | **14.74** |

(b) SPOT5, GRAPH and RLFAP instances

TABLE 5.1 – Comparaison des stratégies parallèles et la méthode séquentielle en utilisant les décompositions arborescentes de MCS.

$abs$. La figure 5.4a compare les temps de calcul moyen cumulatifs [23] obtenus sur l'ensemble des instances. Comme le montre les résultats, les décompositions arborescentes raffinées procurent les meilleurs temps cumulatifs, avec toutefois un avantage pour RADGVNS-MINFILL-$abs$. Si on regarde plus en détail les résultats par catégorie de problèmes, on peut noter que sur les instances TagSNP (cf. figure 5.4b), RADGVNS-MINFILL-$abs$ obtient les meilleurs temps de calcul sur la moitié des instances (#12976, #14226, #6858 et #9319), alors que sur l'autre moitié c'est RADGVNS-MCS-$abs$ qui est le plus rapide. On retrouve le même comportement sur les instances RLFAP et GRAPH. Toutefois, sur les instances SPOT5, RADGVNS se comporte différemment. En effet, sur les décompositions initiales obtenues par MCS et MINFILL, RADGVNS surpasse clairement toutes les autres méthodes.

#### 5.5.1e Impact du nombre de processus

Notre dernière expérimentation évalue l'impact du nombre de processus travailleurs $npr$ sur la performance de RADGVNS. Les tableaux 5.2 et 5.3 donnent les temps de calcul moyens et les speed-ups obtenus pour différentes valeurs de $npr$ sur les décompositions MCS-$abs$ et MINFILL-$abs$ respectivement. Les Colonnes (2–5) indiquent les temps de calcul pour $npr$ [24] égal à 2, $ncl/2$, $ncl$ et 128. Notons que pour $npr = 2$, RADGVNS peut être vu comme une sorte de DGVNS. Comme le montrent les résultats, la meilleure performance est obtenue avec $npr$ égale à 128 processus. Ces résultats montrent aussi que des gains significatifs en termes de temps de calcul peuvent être obtenus avec l'augmentation du nombre de processus, en particulier pour les instances SPOT5 où les gains sont fortement amplifiés par rapport à ceux obtenus avec $npr = ncl$.

23. Pour chaque méthode $m_j$ et chaque point $x$ de l'axe des abscisses, la valeur associée sur l'axe des ordonnées indique le temps moyen cumulatif qu'à pris la méthode $m_j$ pour résoudre les premiers $x$ instances.
24. Cela inclut le processus maître.

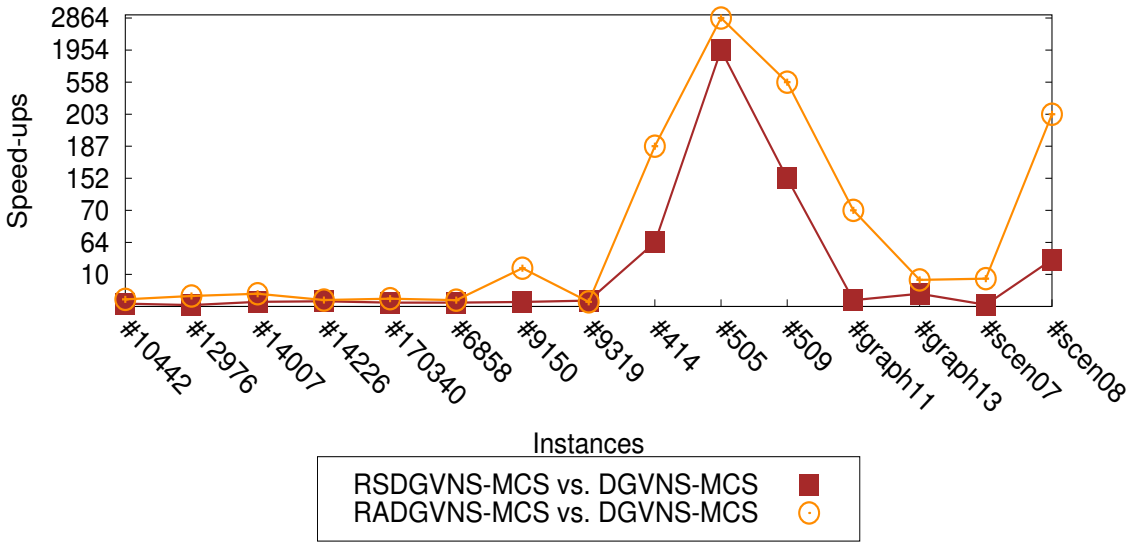



FIGURE 5.2 – RSDGVNS et RADGVNS versus DGVNS

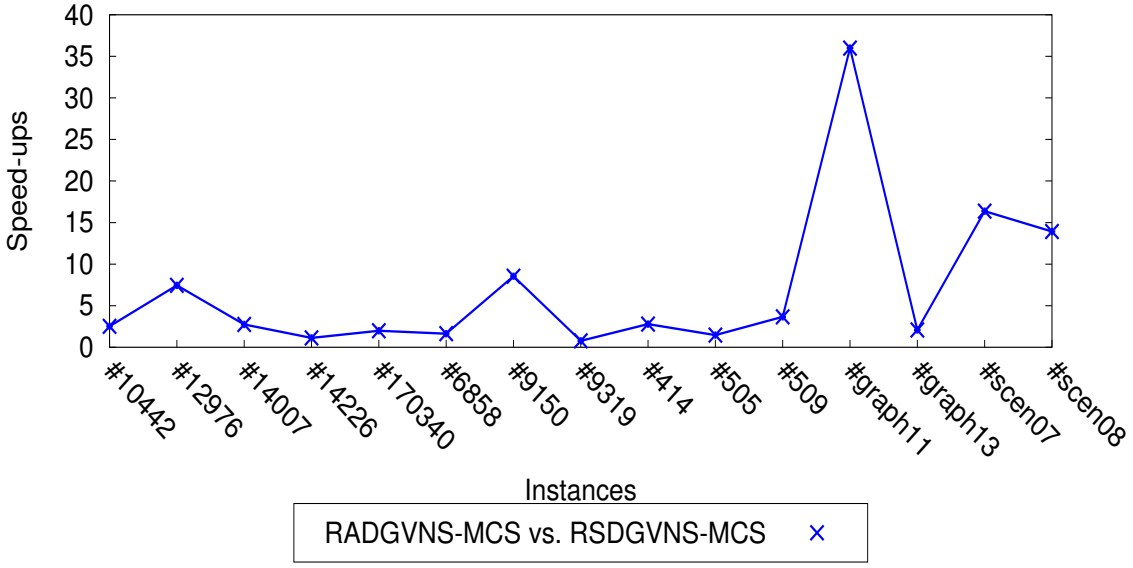
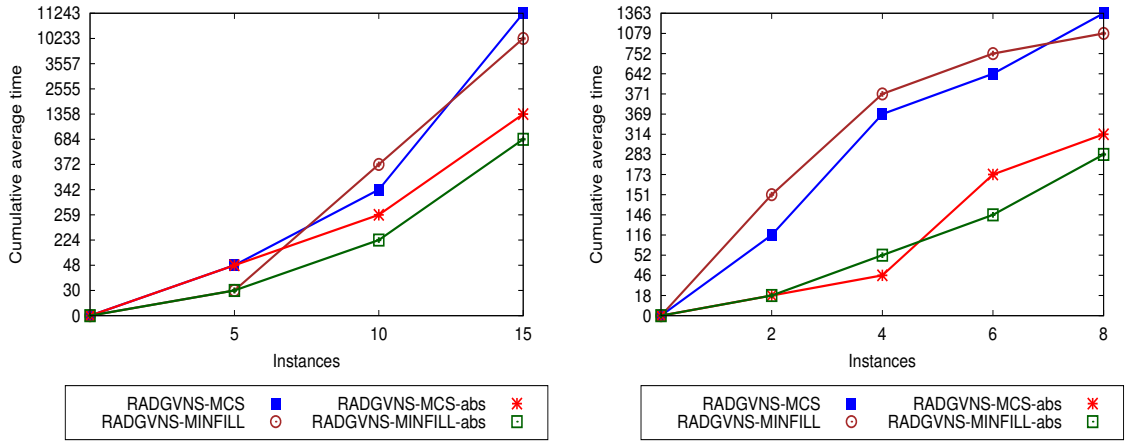



FIGURE 5.3 – RSDGVNS et RADGVNS versus DGVNS

Rappelons que pour RADGVNS, lorsque $(npr > ncl)$, différents processus travailleurs peuvent être affectés au même cluster. Par ailleurs, la fonction HNEIGHBORHOOD (voir l'algorithme 23) assure que tous les sous-ensembles $X_{un}$ de $k$ variables à désaffecter dans la solution courante doivent être différents pour tous les processus travailleurs (y compris ceux assignés au même cluster).

En outre, la communication asynchrone permet aux processus travailleurs d'être initialisés avec différentes meilleures solutions globales courantes. Toutes ces caractéristiques procurent à RADGVNS un niveau élevé de diversification, en explorant différentes régions en parallèle au sein du même cluster. Ceci explique probablement les bonnes performances de RADGVNS lorsque le nombre de processus augmente.




(a) Toutes les instances.

(b) Instances TagSNP

FIGURE 5.4 – Comparing cumulative time results for RADGVNS on the four decompositions.

| Instance | Temps | | | | Speed-up | | |
|---|---|---|---|---|---|---|---|
| | (1) | (2) | (3) | (4) | (1/2) | (1/3) | (1/4) |
| Scen07 | 186.53 | 60.45 | 58.13 | **41.55** | 3.08 | 3.20 | **4.48** |
| Scen08 | 2437.24 | 25.19 | 13.07 | **9.12** | 96.75 | 186,47 | **267.24** |
| Graph11 | 22.30 | 9.43 | 7.00 | **3.83** | 2.36 | 3.18 | **5.82** |
| Graph13 | 28.61 | 12.65 | 9.83 | **7.14** | 2.26 | 2.91 | **4.0** |
| #414 | 2244.27 | 377.98 | 659.91 | **8.28** | 5.93 | 3.40 | **271.04** |
| #505 | 2448.48 | 216.64 | 72.39 | **1.24** | 11.30 | 33.82 | **1974.58** |
| #509 | 1595.74 | 367.77 | 292.44 | **4.39** | 4.33 | 4.45 | **363.49** |
| #10442 | 85.04 | 19.16 | 14.71 | **11.80** | 4.43 | 5.78 | **7.20** |
| #12976 | 302.20 | 98.44 | 82.60 | **69.43** | 3.06 | 3.65 | **4.35** |
| #14007 | 1253.40 | 127.71 | 91.35 | **66.24** | 9.81 | 13.72 | **18.92** |
| #14226 | 119.99 | 21.13 | 15.19 | **11.40** | 5.67 | 7.89 | **10.52** |
| #17034 | 197.31 | 42.30 | 31.42 | **24.73** | 4.66 | 6.27 | **7.79** |
| #6858 | 917.80 | 279.28 | 208.75 | **134.44** | 3.28 | 4.39 | **6.82** |
| #9150 | 1171.47 | 154.99 | 105.98 | **60.50** | 7.55 | 11.05 | **19.36** |
| #9319 | 11.00 | 4.58 | **3.48** | 3.56 | 2.40 | **3.16** | 3.08 |

TABLE 5.2 – Impact du nombre de processus sur les performances de RADGVNS-MCS-abs.

## 5.6 Bilan

Ces résultats démontrent clairement l'efficacité de RADGVNS par rapport à DGVNS, CPDGVNS et RSDGVNS sur les décompositions initiales de MCS et MINFILL ainsi que l'importance d'échanger souvent la meilleure solution globale entre les travailleurs pour améliorer la diversification. Ces résultats montrent également que RSDGVNS est la deuxième meilleure stratégie parallèle. Enfin, les expériences montrent que les décompositions arborescentes raffinées permettent de bien améliorer l'efficacité de nos approches parallèles.

| Instance | Temps | | | | Speed-up | | |
|---|---|---|---|---|---|---|---|
| | (1) | (2) | (3) | (4) | (1/2) | (1/3) | (1/4) |
| Scen07 | 56.92 | 10.95 | 5.63 | **3.99** | 5.19 | 10.11 | **14.26** |
| Scen08 | 2910.22 | 91.21 | 50.72 | **37.21** | 31.90 | 57.37 | **78.21** |
| Graph11 | 6.74 | 2.40 | **2.06** | 2.38 | 2.80 | **3.27** | 2.83 |
| Graph13 | 16.20 | 5.08 | 4.26 | **4.15** | 3.18 | 3.80 | **3.9** |
| #414 | 2176.95 | 879.93 | 84.17 | **7.32** | 2.47 | 25.86 | **297.39** |
| #505 | 2017.06 | 216.62 | 72.32 | **1.18** | 9.31 | 27.89 | **1709.37** |
| #509 | 1163.37 | 294.16 | 219.59 | **3.98** | 3.95 | 5.29 | **292.30** |
| #10442 | 95.03 | 20.40 | 15.72 | **11.86** | 4.65 | 6.04 | **8.01** |
| #12976 | 252.82 | 65.16 | 49.20 | **42.42** | 3.87 | 5.13 | **5.95** |
| #14007 | 1163.70 | 115.29 | 96.93 | **82.66** | 10.09 | 12.00 | **14.07** |
| #14226 | 118.87 | 21.54 | 14.87 | **11.20** | 5.51 | 7.99 | **10.61** |
| #17034 | 250.22 | 52.89 | 36.92 | **27.07** | 4.73 | 6.77 | **9.24** |
| #6858 | 527.10 | 218.83 | 175.11 | **159.13** | 2.40 | 3.01 | **3.31** |
| #9150 | 1761.29 | 166.92 | 108.51 | **46.06** | 10.55 | 16.23 | **38.23** |
| #9319 | 10.85 | 4.62 | 3.46 | **3.55** | 2.34 | 3.13 | **3.05** |

TABLE 5.3 – Impact du nombre de processus sur les performances de RADGVNS-MINFILL-abs.

# Chapitre 6

# Découverte d'ensembles de motifs utilisant la PLNE

## Sommaire


**6.1 Démarche en deux étapes . . . 92**
6.1.1 Aperçu général . . . 92
6.1.2 Catégorisation des contraintes . . . 93
6.1.3 Résolution des contraintes locales . . . 94
6.1.4 PLNE et contrainte n-aires . . . 94
**6.2 Reformulation linéaire des contraintes n-aires . . . 95**
6.2.1 Contrainte de taille ($C_{taille}^{\theta_{taille},\lesseqgtr}$) . . . 95
6.2.2 Contrainte de couverture ($C_{cov}^{\theta_{freq},\lesseqgtr}$) . . . 96
6.2.3 Contrainte d'aire ($C_{aire}^{\theta_{aire},\lesseqgtr}$) . . . 97
6.2.4 Contrainte de redondance ($C_{redd}^{\theta,\lesseqgtr}$) . . . 98
6.2.5 Contrainte de généralisation ($C_{gen}^{\Psi}$) . . . 99
6.2.6 Contrainte de spécialisation ($C_{spe}^{\Psi}$) . . . 100
6.2.7 Contrainte de représentativité ($C_{rep}^{\theta_{rep},\lesseqgtr,\mathcal{T}_i}$) . . . 101
**6.3 Contraintes d'agrégation . . . 101**
6.3.1 Contrainte d'agrégation moyenne ($C_{avg}^{\theta_{avg},\lesseqgtr}$) . . . 101
6.3.2 Contrainte d'agrégation somme ($C_{sum}^{\theta_{sum},\lesseqgtr}$) . . . 102
6.3.3 Contrainte d'agrégation maximum et minimum ($C_{max}^{\theta_{max}}$, $C_{min}^{\theta_{min}}$) . . . 102
**6.4 Approximations linéaires des contraintes n-aires difficiles . . . 102**
**6.5 Conclusion . . . 103**


Dans ce chapitre, nous proposons une approche déclarative et efficace pour l'extraction d'ensembles de motifs. Notre approche utilise la PLNE et adresse les trois aspects discutés dans la section *bilan* du chapitre 3, à savoir la complétude, la déclarativité et le passage à l'échelle

Pour cela, nous avons retenu une démarche en deux étapes faisant coopérer des extracteurs de motifs locaux avec la PLNE. Notre approche peut être vue comme un schéma hybridant des outils issus de la fouille de motifs avec ceux issus de la recherche opérationnelle. La première étape, dite de pré-traitement, permet d'extraire l'ensemble des concepts formels candidats à l'aide d'un outil dédié au problème de l'extraction de motifs fermés. Dans la seconde étape, la PLNE est utilisée pour sélectionner un sous-ensemble de ces concepts pour former le

meilleur ensemble de motifs qui optimise une mesure (i.e. fonction objectif) donnée. Nous proposons une reformulation linéaire pour plusieurs contraintes n-aires introduites en section 3.3.2 et portant sur des ensembles de motifs recherchés. Ces contraintes linéaires seront utilisées pour instancier différentes requêtes issues de plusieurs applications. Pour réduire la complexité de la tâche d'extraction, nous avons séparé les contraintes locales portant sur les motifs locaux des contraintes n-aires définies sur les ensembles de motifs. Chaque catégorie de contraintes faisant intervenir des algorithmes de résolution dédiés et efficaces. À notre connaissance, il s'agit de la première approche générique et flexible utilisant la PLNE pour l'extraction d'ensembles de motifs.

**Plan du chapitre.** Nous dressons, dans la section 6.1, un aperçu général des difficultés posées par l'extraction d'ensembles de motifs, et nous justifions nos différents choix. Dans la section 6.2, nous présentons les reformulations linéaires des contraintes n-aires introduites en section 3.3.2. La section 6.3 montre comment des contraintes impliquant des agrégats peuvent être modélisées par des expressions linéaires. Dans la section 6.4, nous donnons des approximations linéaires pour certaines contraintes n-aires difficilement exploitables par les solveurs PLNE. Enfin, nous discutons, dans la section 6.5 les points forts et les limitations de notre approche pour l'extraction d'ensembles de motifs.

## 6.1 Démarche en deux étapes

Dans cette section, nous présentons les principales difficultés liées à l'extraction d'ensembles de motifs. Puis, nous justifions nos différents choix. Enfin, nous terminons la section en décrivant notre approche en deux étapes.

### 6.1.1 Aperçu général

Très peu d'approches génériques existent pour l'extraction d'ensembles de motifs. Ce problème est généralement traité avec des méthodes heuristiques. Les seules approches déclaratives exhaustives se restreignent à celles basées sur la PPC, ou à l'approche proposée dans [Raedt et Zimmermann, 2007]. Nous dressons ci-dessous différents constats et recensons les principales difficultés auxquelles font face les approches existantes.

— La plupart des approches existantes procèdent en plusieurs étapes et exploitent diverses heuristiques et stratégies de recherche gloutonne pour sélectionner, à partir d'un ensemble de motifs candidats, un sous-ensemble de motifs satisfaisant un ensemble de contraintes sur les ensembles de motifs. Toutefois, la qualité des résultats obtenus par ces approches reste très variable et dépend fortement des paramètres d'initialisation, rendant ainsi la tâche d'extraction complexe et fastidieuse. Par ailleurs, l'extension de ces algorithmes pour résoudre de nouveaux problèmes est souvent une tâche difficile qui nécessite de revisiter tout l'algorithme ce qui est, dans certains cas, non réalisable.
— En traitant l'extraction des motifs locaux dans une étape séparée, la taille de l'espace de recherche pour l'extraction des ensembles de motifs est réduite au nombre de motifs locaux extraits,
— Contrairement aux contraintes locales qui exploitent des propriétés (e.g. l'anti-monotonicité) pour produire un meilleur élagage de l'arbre de recherche, les contraintes n-aires sont souvent caractérisées par des propriétés d'élagage qui sont faibles. De plus, la combinaison simultanée de plusieurs types de contraintes au-delà des quelques classes de contraintes étudiées est loin d'être triviale. En effet, trouver un algorithme général offrant un bon compromis entre efficacité et déclarativité est un défi en soi.
— Les approches déclaratives exactes proposées dans la littérature se heurtent au problème du passage à l'échelle, notamment en raison de l'encodage utilisé. Bien que ces approches procurent un cadre déclaratif très flexible, l'absence d'algorithmes efficaces réduit leur intérêt pratique.
— En pratique, l'expert est souvent intéressé par la découverte du meilleur ensemble de motifs. La prise en compte de la dimension optimisation permet de mieux cibler le processus d'extraction vers les meilleurs motifs mais rend la tâche d'extraction encore plus ardue. Il est donc nécessaire de disposer de méthodes/-solveurs efficaces qui offrent la garantie de trouver une solution optimale et d'en prouver l'optimalité.

Disposer d'un cadre offrant un bon compromis entre complétude (pour garantir la preuve d'optimalité de la solution optimale) et déclarativité (pour permettre l'extension vers de nouveaux problèmes et le passage à l'échelle) constitue un réel défi. L'approche que nous proposons procède en deux étapes successives, permettant de ramener la résolution d'un problème globalement complexe à celle d'un ensemble de sous-problèmes plus faciles à résoudre, et/ou mieux connus. On peut alors faire appel à des algorithmes dédiés et efficaces, pour résoudre tout ou partie de ces sous-problèmes. La résolution exacte du problème global, est intégrée efficacement dans une approche de résolution générique et flexible basée sur la PLNE. Cette démarche est intéressante et permet le passage à l'échelle par rapport à une approche utilisant une seule étape de résolution.

### 6.1.2 Catégorisation des contraintes

Notre approche en plusieurs étapes profite pleinement de la catégorisation des contraintes du problème en différentes classes. Cette catégorisation permet, d'une part, d'exhiber un ensemble de sous-problèmes plus faciles à résoudre, et d'autre part, d'utiliser des algorithmes dédiés efficaces sur des contraintes ayant de bonnes propriétés d'élagage. En fouille de données, une telle démarche est bien connue, par exemple, en extrayant tout d'abord les motifs fréquents ; puis en les post-traitant de manière heuristique afin de trouver un ensemble de motifs unique. Toutefois, contrairement aux approches heuristiques, notre approche est complète et fournit des garanties pour trouver l'ensemble de motifs optimal satisfaisant les différentes contraintes du problème.

Les contraintes utilisées pour l'extraction d'ensembles de motifs sont soit locales, c'est-à-dire portant sur un et un seul motif, soit n-aires impliquant plusieurs motifs dans leurs portées. Contrairement aux contraintes n-aires, les contraintes locales sont caractérisées par des propriétés mathématiques, telle que l'anti-monotonie, permettant un élagage très efficace de l'espace de recherche. Exploiter pleinement les bonnes propriétés des contraintes locales permet donc de mieux appréhender la question du passage à l'échelle par rapport à une démarche en une seule étape. Ainsi, dans ce qui suit, nous séparons les contraintes d'une requête en deux sous-ensembles :

— $C_{locales}$ est l'ensemble de contraintes locales à satisfaire portant sur des motifs individuels. Un exemple de contrainte locale est la contrainte de fermeture ;
— $C_{n-aires}$ est l'ensemble des contraintes restantes, souvent n-aires, i.e. impliquant plusieurs motifs. Elles seront traitées par un solveur générique.

**Exemple 6.1.**
La requête suivante cherche sur une base de transactions des ensembles de motifs couvrant toutes les transactions sans chevauchement. Chaque motif de l'ensemble doit avoir une fréquence minimale égale à 2, et une taille minimale égale 3.

$$(Q_{demo}) \begin{cases} taille(\Phi) = k \wedge \\ \wedge_{1 \leq i \leq k}\, closed(\phi_i) \wedge \\ cover(\Phi, \mathcal{T}) = \mathcal{T} \wedge \\ \wedge_{1 \leq i < j \leq k}\, chevauche(\phi_i, \phi_j, \mathcal{T}) = 0 \wedge \\ \wedge_{1 \leq i \leq k}\, freq(\phi_i, \mathcal{T}) \geq 2 \wedge \\ \wedge_{1 \leq i \leq k}\, taille(\phi_i) \geq 3. \end{cases}$$

Les contraintes associées à la requête $Q_{demo}$ sont séparées en deux ensembles :

— $C_{locales} = \{closed(\phi_i), freq(\phi_i, \mathcal{T}), taille(\phi_i)\}$.
— $C_{n-aires} = \{taille(\Phi), chevauche(\phi_i, \phi_j, \mathcal{T}), cover(\Phi, \mathcal{T})\}$.

L'ensemble de contraintes $C_{locales}$ spécifie un problème bien étudié en extraction des motifs locaux sous contraintes et pour lequel il existe des algorithmes efficaces, alors que l'ensemble $C_{n-aires}$ est restreint aux contraintes impliquant plusieurs motifs.

### 6.1.3 Résolution des contraintes locales

Dans une étape de pré-traitement, nous extrayons l'ensemble des motifs locaux en utilisant des extracteurs efficaces (voir section 3.2.4). De ce fait, la résolution des contraintes n-aires sera réduite à l'espace des solutions des contraintes locales. Notons que, plus les contraintes locales seront sélectives, moins le nombre de motifs candidats générés sera élevé. Par ailleurs, nous nous intéressons aux représentations condensées de motifs. Selon le type de représentation considérée, nous utiliserons le meilleur algorithme permettant de les extraire ; ainsi, nous avons utilisé l'algorithme LCM pour l'extraction des motifs fermés.

**Exemple 6.2.**
Reprenons le jeu de données illustré dans la table 3.1 du chapitre 3. La requête $Q_{C_{locales}}$ définie par les contraintes de $C_{locales}$ permet d'extraire l'ensemble de tous les motifs fermés ayant une fréquence supérieure ou égale 2 avec une taille supérieure ou égale 3.

$$(Q_{C_{locales}}) \begin{cases} closed(\phi) \wedge \\ freq(\phi, \mathcal{T}) \geq 2 \wedge \\ taille(\phi) \geq 3. \end{cases}$$

Les solutions à cette requête sont : $\phi \in \{\{B, E, G\}, \{A, E, G\}, \{C, E, G\}, \{C, E, H\}\}$.

Certaines contraintes locales, telles que la fréquence minimale, sont directement prises en charge par l'extracteur de motifs, ces contraintes respectent généralement les propriétés d'anti-monotonie. Pour d'autres types de contraintes, telle que les contraintes `MustLink` et `CannotLink` [25], elles seront filtrées séparément après l'extraction des motifs locaux. Notons, toutefois, que ces contraintes pourraient aussi être modélisées comme des contraintes linéaires et résolues avec l'ensemble des contraintes n-aires.

### 6.1.4 PLNE et contrainte n-aires

Nous justifions ci-dessous notre choix d'utiliser la PLNE pour l'extraction des ensembles de motifs.

**Cadre déclaratif.** La déclarativité est un aspect important pour l'extraction de motifs locaux et d'ensembles de motifs. Elle offre la possibilité d'exprimer différentes propriétés sur l'ensemble de motifs sans pour autant changer l'algorithme de résolution. En effet, traiter une variété de contraintes complique la résolution puisque certaines contraintes respectent quelques propriétés mathématiques (e.g. anti-monotonicité) alors que d'autres non. *La PLNE offre un cadre flexible et rigoureux pour la modélisation et la résolution de nombreux problèmes d'optimisation complexes. Plusieurs types de contraintes n-aires décrites dans la section 3.3.2 ont un lien avec les problèmes de partitionnement (et de couverture par ensembles), largement étudiés par la communauté PLNE.*

**Optimisation.** Dans de nombreuses applications, le nombre d'ensembles de motifs extraits est souvent élevé. Dans ce cas, l'expert doit recourir à des étapes de post-traitement afin de déterminer le meilleur ensemble de motifs qui répond le plus à ses besoins. La prise en compte de la dimension optimisation permet de mieux cibler le processus d'extraction vers les meilleurs motifs mais rend la tâche d'extraction plus ardue. *Les algorithmes de résolution de la PLNE profitent des techniques d'élagage efficaces pour garantir l'obtention de la solution optimale et de sa preuve d'optimalité.*

**Solveurs d'optimisation : PLNE vs. PPC.**

— Un solveur PPC prend des décisions sur les variables et les valeurs ; après chaque décision, il effectue un ensemble d'inférences logiques pour réduire les choix disponibles dans les domaines des variables

25. Des contraintes forçant que deux transactions $t_1$ et $t_2$ doivent être couvertes $\{t_1, t_2\} \subseteq cover(\phi, \mathcal{T})$ (resp. non couvertes) par un motif $\phi$.

restantes. En revanche, un solveur de programmation mathématique, notamment dans le contexte de l'optimisation discrète, utilise une combinaison de relaxations renforcée par des plans de coupe, de branchement et des calculs de bornes.

— Notre choix de la PLNE est lié au fait qu'un solveur PPC prouve l'optimalité en montrant qu'aucune solution (meilleure la solution courante) ne peut être trouvée que par une énumération explicite de toutes les solutions du problème, alors qu'un solveur de programmation linéaire utilise une borne inférieure imposée par des coupes et des relaxations linéaires afin d'énumérer implicitement les solutions du problème. Ainsi, le processus de résolution est générique et plus efficace.

— Un solveur de programmation par contraintes ne fait pas d'hypothèses sur les propriétés mathématiques de l'espace de solution (linéarité, convexité, etc.), alors qu'un solveur de programmation linéaire (ou mathématique) exige que le modèle soit dans une catégorie mathématique bien définie, ayant dans l'esprit, qu'au fil du temps, beaucoup de problèmes ont été étudiés dans ce contexte.

En résumé, la PLNE est un cadre rigoureux et flexible permettant :

1. de modéliser un large éventail de contraintes,
2. d'offrir un niveau élevé de modélisation des problèmes d'optimisation,
3. de traiter efficacement des problèmes liés au partitionnement et à la couverture par ensembles (Set covering en anglais).

De plus, les solveurs PLNE modernes sont très efficaces et intègrent de nombreuses heuristiques de recherche. Par conséquent, nous extrayons le meilleur ensemble de motifs en utilisant des modèles PLNE définis sur des **motifs locaux candidats** notés $\mathcal{P}$, qui sont extraits durant l'étape de pré-traitement. Nous donnons, ci-dessous, la formulation générale de notre modèle PLNE de base :

**- Variables.** Pour chaque motif candidat $p \in \mathcal{P}$, nous introduisons une variable booléenne $x_p$ telle que $x_p = 1$ si, et seulement si, le motif candidat $p$ est sélectionné.

**- Contraintes.** Les contraintes n-aires sont reformulées avec des expressions linéaires sur les variables booléennes $x_p$. Nous détaillons, dans les prochaines sections, les reformulations linéaires de différentes contraintes n-aires utilisées pour l'extraction d'ensembles de motifs.

**- Fonction objectif :** Dans la plupart de nos modèles, la fonction objectif est une expression linéaire portant sur une ou plusieurs mesures : la fréquence, la taille, la diversité, etc., sur tous les motifs candidats $\mathcal{P}$.

## 6.2 Reformulation linéaire des contraintes n-aires

Dans cette section nous détaillons les reformulations linéaires des contraintes n-aires présentées dans la section 3.3.2. Soient $\mathcal{T}$ l'ensemble de transactions, $\mathcal{P}$ l'ensemble de $n$ motifs candidats, et $(a_{t,p})$ la matrice binaire telle $(a_{t,p} = 1)$ si et seulement $(p \subseteq t)$, i.e. le motif $p$ couvre la transaction $t$. Par conséquent, un ensemble de motifs est une solution d'un PLNE avec $n$ variables booléennes $(x_p,\ p \in \mathcal{P})$, où $(x_p = 1)$ si, et seulement si, le motif $p$ appartient à l'ensemble de motifs recherché $\Phi$.

### 6.2.1 Contrainte de taille $(C_{taille}^{\theta_{taille},\leqslant})$

Rappelons que la taille d'un ensemble de motifs $\Phi$ correspond au nombre de motifs locaux qui le constituent. Comme l'encodage PLNE utilise des variables binaires, qui prennent des valeurs non nulles $(x_p = 1)$ si le motif $p$ est sélectionné pour former l'ensemble de motifs $\Phi$ , alors la taille de $\Phi$ correspond à :

$$taille(\Phi) = \sum_{p \in \mathcal{P}} x_p.$$

Étant donné un seuil $\theta_{taille}$, la contrainte de taille $C_{taille}^{\theta_{taille},\lesseqgtr}$ est modélisée comme suit :

$$(C_{taille}^{\theta_{taille},\lesseqgtr}) : taille(\Phi) \lesseqgtr \theta_{taille} \Leftrightarrow \sum_{p \in \mathcal{P}} x_p \lesseqgtr \theta_{taille}$$

### 6.2.2 Contrainte de couverture $(C_{cov}^{\theta_{freq},\lesseqgtr})$

La couverture d'un ensemble de motifs correspond aux transactions couvertes par au moins un motif local dans l'ensemble de motifs recherché $\Phi$. Nous définissons $m$ variables booléennes $(y_t,\ t \in \mathcal{T})$ telles que $(y_t = 1)$ si, et seulement si, il existe au moins un motif $(\phi \in \Phi)$ tel que le motif $\phi$ couvre la transaction $t$, i.e $(\phi \subseteq t)$.

Par conséquent, nous modélisons la fréquence d'un ensemble de motifs ($\Phi$) par :

$$freq(\Phi, \mathcal{T}) = \sum_{t \in \mathcal{T}} y_t.$$

Étant donné un seuil $(\theta_{freq})$, la contrainte de couverture $(C_{cov}^{\theta_{freq},\lesseqgtr})$ est modélisée par :

$$(C_{cov}^{\theta_{freq},\lesseqgtr}) : freq(\Phi, \mathcal{T}) \lesseqgtr \theta_{freq} \Leftrightarrow \sum_{t \in \mathcal{T}} y_t \lesseqgtr \theta_{freq}$$

Afin de fixer les valeurs des variables $y_t$ en fonction de l'ensemble de motifs recherché $\Phi$, la contrainte suivante établit la relation entre les variables $y_t$ et les variables $x_p$. Notons qu'une variable $(y_t = 0)$ si, et seulement si, il n'existe aucun motif $(\phi \in \Phi)$ couvrant la transaction $t$.

$$(C_{x,y}) : y_t \leq \sum_{p \in \mathcal{P}} a_{t,p} \cdot x_p \leq |\mathcal{P}| \cdot y_t, \forall t \in \mathcal{T}$$

**Exemple 6.3.**
Afin d'illustrer la reformulation linéaire de la contrainte de couverture, nous considérons les motifs fermés extraits à partir du jeu de données de la table 3.1. Sur un ensemble de 18 motifs candidats, nous cherchons un ensemble de motifs de taille 3 représentant une variante relâchée du problème de clustering conceptuel. L'ensemble de motifs trouvé est :

**motif 3 :** $\{E, G\}$ couvrant les transactions $\{t_3, t_4, t_5, t_6, t_7, t_8\}$.
**motif 8 :** $\{A\}$ couvrant les transactions $\{t_1, t_2, t_3, t_4\}$.
**motif 13 :** $\{C, H\}$ couvrant les transactions $\{t_9, t_{10}, t_{11}\}$.

Seules les variables $\{x_2, x_7, x_{12}\}$ sont égales à 1, toutes les autres variables sont à 0. Considérons la transaction $t_3$ couverte par les motifs 2 et 7, le reste des motifs non sélectionnés couvrant la même transaction $t_3$ ne sont pas pris en considération puisque $(a_{t,p} \cdot x_p = 0)$. La variable binaire $y_3$ associée à la transaction $t_3$ ne peut être nulle, car :

$$\begin{cases} a_{3,2} \cdot x_2 + a_{3,7} \cdot x_7 \leq 18 \cdot y_3 \\ a_{3,2} \cdot x_2 + a_{3,7} \cdot x_7 \geq y_3 \end{cases} \quad \rightarrow \quad y_3 = 1,$$

Par conséquent, la variable $y_3$ sera forcément égale à 1, et ainsi la couverture de $t_3$ sera considérée dans le calcul de la fréquence de l'ensemble de motifs recherché.

### 6.2.3 Contrainte d'aire $(C_{aire}^{\theta_{aire},\lesseqgtr})$

L'aire d'un ensemble de motifs correspond au nombre d'items couverts sur chaque transaction. Si la base de transactions est représentée par une matrice binaire, alors l'aire correspond à la somme des 1 sur toutes les lignes couvertes.

Nous définissons $(|\mathcal{T}| \times |\mathcal{I}|)$ variables booléennes $(q_{t,i},\ t \in \mathcal{T} \wedge i \in \mathcal{I})$ telles que $(q_{t,i} = 1)$ si, et seulement si, le couple transaction et item $(t, i)$ est couvert par au moins un motif dans l'ensemble de motifs recherché $\Phi$. Par conséquent, nous modélisons l'aire d'un ensemble de motifs par :

$$aire(\Phi, \mathcal{T}) = \sum_{i \in \mathcal{I}, t \in \mathcal{T}} q_{t,i}$$

Étant donné un seuil $\theta_{aire}$, la contrainte d'aire $(C_{aire}^{\theta_{aire},\lesseqgtr})$ est modélisée par :

$$(C_{aire}^{\theta_{aire},\lesseqgtr}) : aire(\Phi, \mathcal{T}) \lesseqgtr \theta_{aire} \Leftrightarrow \sum_{i \in \mathcal{I}, t \in \mathcal{T}} q_{t,i} \lesseqgtr \theta_{aire}$$

Afin d'établir la relation entre les variables $q_{t,i}$ et les variables $x_p$, nous utilisons une matrice binaire $cq_{t,i}^p$ représentant la couverture des couples transaction et item $(t, i)$ pour chaque motif candidat $(p \in \mathcal{P})$. Par conséquent, il est possible pour un couple $(t, i)$ de savoir s'il existe au moins un motif, sélectionné dans l'ensemble de motifs recherché $(x_p = 1)$, couvrant ce couple $(q_{t,i} = 1)$ par la contrainte suivante :

$$(C_{x,q}) : q_{t,i} \leq \sum_{p \in \mathcal{P}} cq_{t,i}^p x_p \leq |\mathcal{P}| \,.\, q_{t,i}, \forall t \in \mathcal{T},\ \forall i \in \mathcal{I}.$$

**Exemple 6.4.**
Considérons à nouveau le jeu de données de la table 3.1. Sur les 18 motifs candidats, nous recherchons un ensemble de motifs de taille égale à 3 ayant une aire maximale. L'ensemble de motifs extrait est :

**motif 1 :** $\{E\}$ couvrant les transactions $\{t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}\}$.
**motif 2 :** $\{G\}$ couvrant les transactions $\{t_3, t_4, t_5, t_6, t_7, t_8, t_{11}\}$.
**motif 13 :** $\{C, H\}$ couvrant les transactions $\{t_9, t_{10}, t_{11}\}$.

La matrice de couverture associée au motif $\{E\}$ (resp. $\{G\}$) est illustrée par le tableau 6.1a (resp. tableau 6.1b). Le motif $\{E\}$ couvre la transaction $t_3$, néanmoins le couple $(t_3, C)$ n'est pas couvert par ce dernier, car $cq_{t_3,C}^{\{E\}} = 0$, mais est couvert par un autre motif $\{G\}$ faisant partie de l'ensemble de motifs. Seules les variables $\{x_0, x_1, x_{12}\}$ sont égales à 1, toutes les autres variables prennent la valeur 0. Dans ce qui suit, nous donnons uniquement les termes ayant $x_p \neq 0$. La variable binaire $q_{t_3,E}$ associée à la transaction $t_3$ et l'item $E$ est nulle, car :

$$\begin{cases} q_{t_3,E} \leq cq_{t_3,E}^0 \cdot x_0 + cq_{t_3,E}^1 \cdot x_1 + cq_{t_3,E}^{12} \cdot x_{12} \leq 18 \,.\, q_{t_3,E} & \rightarrow \quad q_{t_3,E} = 1, \\ q_{t_3,G} \leq cq_{t_3,G}^0 \cdot x_0 + cq_{t_3,G}^1 \cdot x_1 + cq_{t_3,E}^{12} \cdot x_{12} \leq 18 \,.\, q_{t_3,G} & \rightarrow \quad q_{t_3,G} = 1, \end{cases}$$

Par conséquent, l'aire de l'ensemble de motifs recherché pourrait être calculée par une sommation sur les variables $q_{t,i}$, i.e. $\sum_{i \in \mathcal{I}, t \in \mathcal{T}} q_{t,i}$.

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| $t_1$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $t_2$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| $t_3$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| $t_4$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| $t_5$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| $t_6$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| $t_7$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| $t_8$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| $t_9$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| $t_{10}$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| $t_{11}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

(a) Matrice de couverture $cq_{t,i}^{\{E\}}$ du motif $\{E\}$.

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| $t_1$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $t_2$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $t_3$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| $t_4$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| $t_5$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| $t_6$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| $t_7$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| $t_8$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| $t_9$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $t_{10}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $t_{11}$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

(b) Matrice de couverture $cq_{t,i}^{\{G\}}$ du motif $\{G\}$.

TABLE 6.1 – Matrice de couverture par motif.

### 6.2.4 Contrainte de redondance $(C_{redd}^{\theta,\lessgtr})$

Les variables $(y_t,\ t \in \mathcal{T})$ modélisant la couverture des transactions par l'ensemble de motifs peuvent être aussi utilisées pour modéliser des contraintes sur la redondance de l'ensemble de motifs.

Afin de contraindre les transactions couvertes plusieurs fois par les différents motifs sélectionnés, nous introduisons des variables booléennes $(u_t,\ t \in \mathcal{T})$ tel que $(u_t = 1)$ si, et seulement si, la transaction $t$ est couverte par au moins par deux motifs. Le nombre total de transactions redondantes est modélisé par :

$$red(\Phi, \mathcal{T}) = \sum_{t \in \mathcal{T}} u_t.$$

Étant donné un seuil $(\theta_{redd})$, la contrainte de redondance $(C_{redd}^{\theta_{redd},\lessgtr})$ est modélisée par :

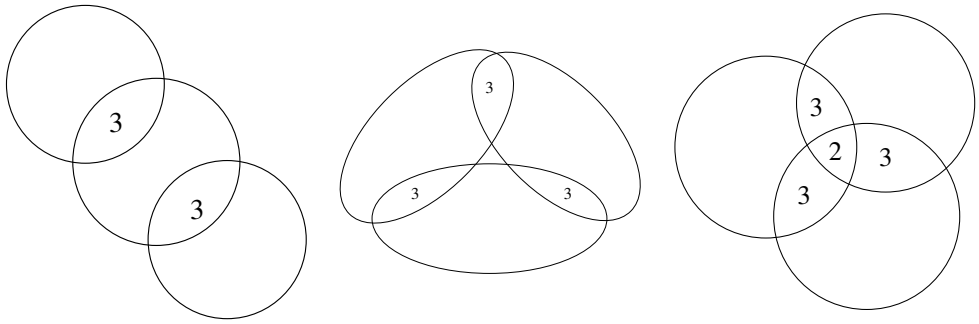
$$(C_{redd}^{\theta_{redd},\lessgtr}) : red(\Phi, \mathcal{T}) \ \lessgtr \ \theta_{redd} \Leftrightarrow \sum_{t \in \mathcal{T}} u_t \ \lessgtr \ \theta_{redd}$$

Une transaction ne peut être redondante $(u_t \neq 1)$ dans l'ensemble de motifs $\Phi$, si elle n'est pas déjà couverte $(y_t = 0)$. Afin d'établir la relation entre les variables booléennes ($u_t$ et $y_t$) et les variables de décision $x_p$, nous introduisons les contraintes suivantes :

$$(C_{x,u}) : 2u_t \leq \sum_{p \in \mathcal{P}} a_{t,p} x_p \leq y_t + |\mathcal{P}| \,.\, u_t, \forall t \in \mathcal{T}$$

Notre définition de la redondance est semblable à celle proposée dans [Shima *et al.*, 2005] mais différente de la redondance proposée dans [Raedt et Zimmermann, 2007], qui est définie sur les paires de motifs plutôt que sur tout l'ensemble de motifs. La dernière définition a été adoptée principalement pour des raisons d'efficacité afin d'avoir un élagage efficace dans l'algorithme *level-wise*.

Afin d'illustrer la différence entre les deux formulations précédentes pour le calcul de la redondance, nous considérons les trois cas de chevauchement des motifs illustrés dans la figure 6.1. Les chiffres qui se trouvent dans la région de chevauchement désignent le nombre de transactions en chevauchement. Ces trois cas pourraient être considérés comme étant semblables avec une contrainte qui mesure le chevauchement maximal entre les paires des motifs, comme définie dans [Raedt et Zimmermann, 2007]. La mesure de redondance globale que nous utilisons




(a) (b) (c)

FIGURE 6.1 – Chevauchement de trois motifs

évalue le cas (a) à 6, le cas (b) à 9, et le cas (c) à 11. Il est à noter que la somme sur les paires de motifs conduit à un chevauchement de 15 pour (c), surestimant ainsi la redondance sur les transactions. Ce contre-exemple montre clairement que notre modélisation de la redondance est plus précise comparée à celle présentée dans [Raedt et Zimmermann, 2007].

**Exemple 6.5.**
Reprenons l'ensemble de motifs donné dans l'exemple 6.4 :

**motif 3 :** $\{E, G\}$ couvrant les transactions $\{t_3, t_4, t_5, t_6, t_7, t_8\}$.

**motif 8 :** $\{A\}$ couvrant les transactions $\{t_1, t_2, t_3, t_4\}$.

**motif 13 :** $\{C, H\}$ couvrant les transactions $\{t_9, t_{10}, t_{11}\}$.

Les transactions $t_3$ et $t_4$ sont redondantes entre les motifs $\{E, G\}$ et $\{A\}$. Vérifions que notre reformulation de la redondance permet d'instancier $u_3$ à 1 et $u_4$ à 1. Puisque les deux transactions sont couvertes alors $(y_3 = 1)$ et $(y_4 = 1)$, selon la contrainte de $(C_{x,u})$, nous avons :

$$\begin{cases} a_{3,2} \cdot x_2 + a_{3,7} \cdot x_7 \leq 1 + 18 \,.\, u_3 \\ a_{3,2} \cdot x_2 + a_{3,7} \cdot x_7 \geq 2 \cdot u_3 \\ a_{4,2} \cdot x_2 + a_{4,7} \cdot x_7 \leq 1 + 18 \,.\, u_4 \\ a_{4,2} \cdot x_2 + a_{4,7} \cdot x_7 \geq 2 \cdot u_4 \end{cases} \begin{array}{c} \rightarrow \\ \\ \rightarrow \\ \\ \end{array} \begin{array}{c} u_3 = 1, \\ \\ u_4 = 1, \\ \\ \end{array}$$

Pour une transaction couverte non redondante, par exemple $t_8$, la variable $u_8$ est égale à 0, car :

$$\begin{cases} a_{8,2} \cdot x_2 \leq 1 + 18 \,.\, u_8 \\ a_{8,2} \cdot x_2 \geq 2 \cdot u_8 \end{cases} \rightarrow \qquad u_8 = 0.$$

Ainsi, la redondance d'un ensemble de motifs peut être quantifiée en utilisant une somme sur les variables $u_t,\ t \in \mathcal{T}$.

### 6.2.5 Contrainte de généralisation $(C^{\Psi}_{gen})$

Afin de donner à l'expert la possibilité de modéliser certaines connaissances qu'il détient sur la nature de l'ensemble de motifs recherché par rapport à une solution connue, nous reformulons les contraintes de généralisation sous forme linéaire. Nous rappelons que $(\Phi \preceq \Psi)$, si et seulement si, pour tout motif $\psi \in \Psi$, il existe un motif $\phi \in \Phi$ tel que :

$$\phi \preceq \psi.$$

Autrement dit, soit $\Psi$ un ensemble de motifs donné, l'ensemble de motifs $\Phi$ est une généralisation de $\Psi$ si, et seulement si, pour tout motif $(\psi \in \Psi)$, il existe, au moins, un motif $(\phi \in \Phi)$ tel que $(\phi \preceq \psi)$. Pour chaque motif $(\psi \in \Psi)$, on note par l'indice $p$ tout motif local ayant une généralisation à un des motifs dans $\Psi$, i.e. $(p \in \mathcal{P} \mid p \preceq \psi)$. Afin que l'ensemble de motifs recherché $\Phi$ soit une généralisation de l'ensemble de motifs donné $\Psi$, il suffit qu'au moins une des variables $x_p$ soit égale à 1 pour tout motif $(\psi \in \Phi)$. La formulation linéaire de cette contrainte est la suivante :

$$(C^{\Psi}_{gen}) : \Phi \preceq \Psi \Leftrightarrow \sum_{\{p:\ \forall p \in \mathcal{P} \mid p \preceq \psi\}} x_p \geq 1, \quad \forall \psi \in \Psi.$$

**Exemple 6.6.**
Soit $\Psi$ un ensemble de motifs connu constitué des motifs $\{A, E\}$, $\{C, G\}$ et $\{C, E, H\}$, avec :

**motif 9 :** $\{A, E\}$ couvrant les transactions $\{t_2, t_3, t_4\}$.
**motif 6 :** $\{C, G\}$ couvrant les transactions $\{t_7, t_8, t_{11}\}$.
**motif 14 :** $\{C, E, H\}$ couvrant les transactions $\{t_9, t_{10}\}$.

Sur les 18 motifs candidats extraits à partir de la base transactionnelle (c.f. la table 3.1), $\Phi = \{\{A\}, \{C\}\}$ est un ensemble de motifs plus général que $\Psi$, avec :

**motif 8 :** $\{A\}$ couvrant les transactions $\{t_1, t_2, t_3, t_4\}$.
**motif 4 :** $\{C\}$ couvrant les transactions $\{t_7, t_8, t_9, t_{10}, t_{11}\}$.

Comme $\{A\} \preceq \{A, E\}$, $\{C\} \preceq \{C, G\}$, $\{C\} \preceq \{C, E, H\}$, les variables binaires $x_8$ et $x_4$ associées aux motifs $\{A\}$ et $\{C\}$ doivent prendre la valeur 1, puisqu'ils sont présents dans l'ensemble de motifs et que les inéquations suivantes sont vérifiées :

$$\begin{cases} x_8 + x_1 \geq 1, \\ x_4 + x_2 \geq 1, \\ x_4 + x_{13} \geq 1. \end{cases}$$

### 6.2.6 Contrainte de spécialisation $(C^{\Psi}_{spe})$

La spécialisation s'exprime à l'aide de la généralisation, mais en inversant la position des deux ensembles de motifs. Toutefois, les expressions linéaires de la spécialisation sont assez différentes puisque l'ensemble de motifs ayant la propriété de spécialisation n'est pas connu.

Soit $\Psi$ un ensemble de motifs donné. L'ensemble de motifs $\Phi$ est plus spécial que $\Psi$, noté $(\Psi \preceq \Phi)$, si et seulement si, pour tout motif $(\phi \in \Phi)$, il existe, au moins, un motif $(\psi \in \Psi)$ tel que $(\psi \preceq \phi)$. Puisque l'ensemble de motifs $\Phi$ n'est pas connu à l'avance, nous considérons, pour la formulation linéaire, tous les motifs candidats plus spéciaux par rapport à au moins un motif $\psi$ dans $\Psi$, i.e. $\{p :\ p \in \mathcal{P} \mid \exists \psi \in \Psi,\ \psi \preceq p\}$. Par conséquent, il faudra que la somme sur les variables $x_p$ soit supérieure ou égale à la taille de l'ensemble de motifs recherché $size(\Phi)$ pour que dernier soit une spécialisation de $\Psi$. La formulation linéaire de cette contrainte est la suivante :

$$(C^{\Psi}_{spe}) : \Psi \preceq \Phi \Leftrightarrow \sum_{\{p:\ p \in \mathcal{P} \mid \exists \psi \in \Psi,\ \psi \preceq p\}} x_p \geq \sum_{p \in \mathcal{P}} x_p$$

**Exemple 6.7.**
Soit $\Psi$ un ensemble de motifs constitué des motifs $\{A\}$, $\{C\}$, $\{C, H\}$ et l'ensemble de motifs recherché $\Phi$ t.q. $(\Psi \preceq \Phi)$. L'ensemble de motif $\Phi$ = $\{\{A, E\}, \{C, G\}, \{C, E, H\}\}$ est solution pour la contrainte $(\Psi \preceq \Phi)$ :
$\Psi$ :

**motif 8** $\{A\}$ couvrant les transactions $\{t_1, t_2, t_3, t_4\}$.

**motif 4** $\{C\}$ couvrant les transactions $\{t_7, t_8, t_9, t_{10}, t_{11}\}$.

**motif 13** $\{C, H\}$ couvrant les transactions $\{t_9, t_{10}, t_{11}\}$.

et $\Phi$ :

**motif 9** $\{A, E\}$ couvrant les transactions $\{t_2, t_3, t_4\}$.

**motif 6** $\{C, G\}$ couvrant les transactions $\{t_7, t_8, t_{11}\}$.

**motif 14** $\{C, E, H\}$ couvrant les transactions $\{t_9, t_{10}\}$.

La taille de l'ensemble de motifs recherché $\Phi$ est $\sum_{p \in \mathcal{P}} x_p = 3$. Nous avons $\{A\} \preceq \{A, E\}$, $\{C\} \preceq \{C, G\}$ et $\{C, H\} \preceq \{C, E, H\}$. Les variables binaires $x_9$, $x_6$ et $x_{14}$ associées aux motifs $\{A, E\}$, $\{C, G\}$ et $\{C, E, H\}$ doivent prendre la valeur 1, puisque $x_9 + x_6 + x_{14} \geq 3$.

### 6.2.7 Contrainte de représentativité $(C_{rep}^{\theta_{rep}, \lessgtr, \mathcal{T}_i})$

Soit $\mathcal{T}_i \subseteq \mathcal{T}$ une base transactionnelle partielle et $\theta_{rep}$ un seuil donné pour la mesure de la représentativité d'un ensemble de motifs $\Phi$. En utilisant la définition de la représentativité donnée dans la section 3.3.2a, nous avons :

$$rep(\Phi, \mathcal{T}_i, \mathcal{T}) \geq \theta_{rep} \Leftrightarrow sup(\Phi, \mathcal{T}_i) \geq \theta_{rep} \times sup(\Phi, \mathcal{T}).$$

Nous introduisons $\mid \mathcal{T}_i \mid$ variables booléennes $(y'_t,\ t \in \mathcal{T}_i)$ portant sur la base de transactions partielle telles que $(y'_t = 1)$ si, et seulement si, il existe au moins un motif $\phi \in \Phi$ tel que le motif $\phi$ couvre la transaction $t$. Comme pour les variables de couverture $y_t$, la contrainte suivante établit la relation entre les variables $(y'_t)$ et les variables $(x_p)$.

$$(C_{x,y'}) : y'_t \leq \sum_{p \in \mathcal{P}} a_{t,p} \cdot x_p \leq |\mathcal{P}| \cdot y'_t, \quad \forall t \in \mathcal{T}_i.$$

En conséquence, la reformulation linéaire de la contrainte de représentativité est la suivante :

$$(C_{rep}^{\theta_{rep}, \lessgtr, \mathcal{T}_i}) : rep(\Phi, \mathcal{T}_i, \mathcal{T}) \lessgtr \theta_{rep} \Leftrightarrow \sum_{t \in \mathcal{T}_i} y'_t \lessgtr \theta_{rep} \times \sum_{t \in \mathcal{T}} y_t$$

## 6.3 Contraintes d'agrégation

Cette section montre comment les contraintes impliquant des agrégats peuvent être modélisées par des expressions linéaires.

### 6.3.1 Contrainte d'agrégation moyenne $(C_{avg}^{\theta_{avg}, \lessgtr})$

Soient $\theta_{avg}$ le seuil sur la fréquence moyenne de l'ensemble de motifs $\Phi$, et $f_p$ la valeur de fréquence du motif local $p$. La contrainte $avg(freq(\Phi, \mathcal{T})) \lessgtr \theta_{avg}$ peut être modélisée par l'expression linéaire suivante :

$$(C_{avg}^{\theta_{avg}, \lessgtr}) : \sum_{p \in \mathcal{C}} f_p \cdot x_p - \theta_{avg} \sum_{p \in \mathcal{C}} x_p \lessgtr 0$$

### 6.3.2 Contrainte d'agrégation somme $(C_{sum}^{\theta_{sum},\lesseqgtr})$

La contrainte $sum(sup(\Phi, \mathcal{T})) \lesseqgtr \theta_{sum}$ peut être modélisée sous forme linéaire :

$$(C_{sum}^{\theta_{sum},\lesseqgtr}) : \sum_{p \in \mathcal{C}} f_p \cdot x_p \lesseqgtr \theta_{sum}$$

### 6.3.3 Contrainte d'agrégation maximum et minimum $(C_{max}^{\theta_{max}}, C_{min}^{\theta_{min}})$

La contrainte de maximisation : $max(freq(\Phi, \mathcal{T})) \leq \theta_{max}$ est modélisée par :

$$f_p \cdot x_p \leq \theta_{max}, \forall p \in \mathcal{C}.$$

Enfin, la contrainte $(C_{min}^{\theta_{min}})$ peut être modélisée de la même manière, mais en utilisant $\geq$ au lieu de $\leq$.

## 6.4 Approximations linéaires des contraintes n-aires difficiles

La reformulation linéaire de la contrainte d'aire fait intervenir un nombre très important de variables et de contraintes : $(|\mathcal{T}| * |\mathcal{I}|)$ variables $q_{i,t}$, $(|\mathcal{T}| * |\mathcal{I}|)$ contraintes $(C_{x,q})$ et une contrainte pour quantifier l'aire $(C_{aire}^{\theta_{aire},\lesseqgtr})$. Cela constitue une limitation forte lorsqu'il s'agit de traiter des bases de données de très grande taille. Pour y remédier, nous proposons d'autres reformulations linéaires approximatives des contraintes originales représentant des bornes supérieures des mesures exactes. D'un point de vue qualitatif, les résultats de nos expérimentations (voir chapitre 7) montrent que la solution du modèle approximatif demeure très proche de celle satisfaisant les contraintes originales.

La nouvelle reformulation se base sur une sommation des aires de motifs. Étant donné $(v_p = aire(p),\ \forall p \in \mathcal{P})$, la seconde reformulation de l'aire d'un ensemble de motifs est donnée comme suit :

$$(C_{aire2}^{\theta_{aire},\lesseqgtr}) : \sum_{p \in \mathcal{P}} v_p x_p \lesseqgtr \theta_{aire}$$

L'approximation de la contrainte d'aire constitue bien une borne supérieure de l'aire exacte d'un ensemble de motifs, puisque les transactions couvertes peuvent être redondantes et donc comptées plusieurs fois dans le calcul de l'aire. Il est à noter que si une contrainte impose que les motifs ne doivent pas se chevaucher et que les motifs locaux sont des motifs fermés, l'approximation linéaire de l'aire correspond à l'aire exacte d'un ensemble de motifs.

**Exemple 6.8.**
Étant donné l'ensemble de motifs suivant :

**motif 1 :** $\{E\}$ couvrant les transactions $\{t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}\}$.
**motif 2 :** $\{G\}$ couvrant les transactions $\{t_3, t_4, t_5, t_6, t_7, t_8, t_{11}\}$.
**motif 13 :** $\{C, H\}$ couvrant les transactions $\{t_9, t_{10}, t_{11}\}$.

L'aire exacte de l'ensemble de motifs est égale à 21, qui est aussi égale, sur cet exemple, à la somme des aires des motifs locaux, i.e. $(9 + 6 + (2 * 3) = 21)$.

## 6.5 Conclusion

L'extraction d'ensembles de motifs constitue une tâche prépondérante en fouille de données permettant, d'une part, de contrôler le nombre parfois excessif de motifs locaux extraits, et d'autre part, de découvrir des corrélations entre ces motifs. Dans ce chapitre, nous avons présenté une approche qui préserve l'aspect déclaratif de l'approche originelle [Raedt et Zimmermann, 2007]. Notre approche tire profit à la fois des algorithmes efficaces d'extraction des motifs locaux, mais également de la puissance de modélisation et de résolution de la PLNE, permettant d'intégrer des critères à optimiser. Cela permet à l'utilisateur d'obtenir le meilleur ensemble de motifs satisfaisant un ensemble de contraintes. Nous avons reformulé des contraintes n-aires très utilisées pour la recherche des ensembles de motifs, allant des contraintes de couverture aux contraintes discriminantes. Bien que beaucoup de contraintes peuvent être reformulées efficacement avec des expressions linéaires, certaines contraintes font intervenir des variables intermédiaires et de nouveaux coefficients et de nouvelle contraintes. Nous avons proposé des expressions linéaires offrant des approximations des contraintes originales.

Dans le chapitre suivant, nous utilisons nos différentes formulations linéaires pour construire des modèles PLNE pour différentes applications et requêtes d'extraction d'ensembles de motifs.

# Chapitre 7

# Programmation linéaire pour l'extraction des ensembles de motifs

## Sommaire


**7.1 Applications de l'extraction d'ensembles de motifs . . . 106**
7.1.1 Clustering conceptuel et ses variantes . . . 106
7.1.2 Problème du tuilage . . . 107
**7.2 Extraction des ensembles de motifs en PLNE . . . 108**
7.2.1 Encodage PLNE des problèmes de clustering . . . 108
7.2.2 Clustering conceptuel dur . . . 108
7.2.3 Co-clustering . . . 109
7.2.4 Soft clustering et soft co-clustering . . . 110
7.2.5 Problème du tuilage . . . 110
7.2.6 Modélisation d'autres types de requêtes . . . 111
**7.3 Évaluation des modèles PLNE . . . 112**
7.3.1 Protocole expérimental . . . 112
7.3.2 Passage à l'échelle . . . 113
7.3.3 Relaxation du nombre $k$ de clusters . . . 115
7.3.4 Qualité des résultats . . . 116
**7.4 Conclusion . . . 121**


Le clustering est l'une des tâches fondamentales en fouille de données. L'identification de groupes de données homogènes est essentielle pour la classification non-supervisée, la réduction de données ou la détection d'anomalies. Le clustering conceptuel [Michalski et Stepp, 1983, Fisher, 1987, Perkowitz et Etzioni, 1999, Pensa *et al.*, 2005] fournit une information additionnelle, la description des clusters par des concepts symboliques. Les approches classiques de clustering conceptuel [Michalski et Stepp, 1983, Fisher, 1987] combinent la formation des clusters et des descriptions pour arriver à un compromis. Des approches plus récentes [Perkowitz et Etzioni, 1999, Pensa *et al.*, 2005] ont choisi de séparer la phase de recherche des descriptions de la phase de clustering, la recherche de description ayant lieu avant ou après la formation des clusters.

Notons que l'extraction d'ensembles de motifs est un problème plus général que le clustering conceptuel. Trouver des mesures d'intérêt pour extraire ces ensembles de motifs pertinents est une tâche difficile. En effet, peu de mesures permettent d'élaguer efficacement l'espace de recherche. On retrouve dans la littérature différents exemples de tâches d'extraction d'ensembles de motifs qui n'exploitent pas la propriété d'anti-monotonie sur les motifs. On distingue, particulièrement, le problème de tuilage [Geerts *et al.*, 2004] qui exploite une mesure d'intérêt

sur les motifs basée sur le concept de tuile et l'aire d'une tuile pour former l'ensemble de motifs. (cf. la section 3.4.2a). Le clustering conceptuel et le problème du tuilage ne sont que deux applications des tâches d'extraction d'ensembles de motifs. En général, les problèmes d'extraction d'ensembles de motifs se distinguent par le type de motifs et le type de contraintes n-aires utilisés. Certaines requêtes sont liées à un domaine particulier et font intervenir de nouvelles contraintes, comme celles présentées dans [Raedt et Zimmermann, 2007].

Vue la taille (très) importante de l'espace de recherche à parcourir pour l'extraction d'ensembles de motifs, la majorité des approches proposées sont de nature heuristique (voir chapitre 3) dont les résultats sont fortement influencés par les conditions d'initialisation, et nécessitent généralement de nombreuses exécutions, entraînant une augmentation des temps CPU. De plus, pour finaliser le choix de l'ensemble de motifs, l'utilisateur devra identifier le résultat pertinent. Les approches exactes ont un problème de passage à l'échelle, qui peut réduire l'intérêt de la déclarativité. Notre approche a deux apports majeurs, directement liés à l'usage de la PLNE : exactitude et flexibilité. Notre utilisation de la PLNE permet de modéliser différents problèmes d'extraction d'ensembles de motifs en un problème d'optimisation linéaire sous contraintes, dont la résolution par l'algorithme de séparation et évaluation produit une solution exacte, offrant ainsi des garanties sur la qualité des clusterings en un temps raisonnable par rapport aux approches de l'état de l'art, notamment les approches heuristiques. Les résultats de nos expérimentations montrent clairement l'intérêt de notre approche, en particulier en matière de qualité.

**Plan du chapitre.** Ce chapitre est organisé en trois sections. Nous introduisons dans la section 7.1 le problème de clustering conceptuel et le problème de tuilage. Nous y présentons, notamment, les notions utilisées pour définir chaque problème, ainsi que leurs différentes variantes. Nous présentons dans la section 7.2 les modèles PLNE utilisés pour chaque problème et pour chaque variante. La section 7.3 est consacrée à l'évaluation et à la discussion des résultats de notre approche et des approches existantes relativement aux critères évoqués précédemment : le passage à l'échelle, la déclarativité et la qualité des résultats.

## 7.1 Applications de l'extraction d'ensembles de motifs

Dans cette section, nous rappelons les deux problèmes d'extraction d'ensembles de motifs principalement traités dans cette thèse, à savoir le problème de clustering conceptuel et le problème de tuilage.

### 7.1.1 Clustering conceptuel et ses variantes

Le clustering conceptuel, introduit dans le chapitre 3, peut prendre différentes formes exprimées par la relation entre les transactions et les motifs fermés. Outre la recherche d'un partitionnement sur l'ensemble des transactions, différentes variantes ont été proposées dans la littérature [Berkhin, 2006] afin de trouver d'autres formes de clustering conceptuel. Parmi ces variantes, on peut citer le soft clustering, le co-clustering, et le soft co-clustering. Ces trois problèmes seront définis dans les sections suivantes.

Pour chaque variante, une fonction d'évaluation $f$ est nécessaire pour optimiser la qualité du clustering. Dans nos expérimentations, nous avons considéré deux critères pour définir la qualité d'un clustering conceptuel :

1. maximiser la somme des tailles des concepts, de façon à favoriser les concepts comportant un grand nombre d'items.
2. minimiser la diversité, de façon à favoriser les concepts ayant une meilleure description des transactions couvertes par chaque cluster.

#### 7.1.1a Clustering conceptuel dur

Le clustering (sens général) consiste à partitionner un ensemble de transactions en groupes relativement homogènes. Le clustering conceptuel est un clustering particulier, qui consiste à attribuer une description distincte à chaque cluster (groupe), c.-à-d. le concept caractérisant l'ensemble des transactions qu'il contient (cf. chapitre 3). Ce problème peut donc être formulé comme suit : Rechercher un ensemble de $k$ clusters, chacun des $k$ clusters étant décrit par un motif fermé $\phi_i,\ (1 \leq i \leq k)$, couvrant toutes les transactions sans aucun chevauchement entre les clusters. Cette version est connue sous le nom de problème de clustering conceptuel dur basé sur les concepts formels.

Des exemples de clustering conceptuel pour $k$=3 sont donnés à la table 3.2 du chapitre 3.

#### 7.1.1b Le co-clustering

Le co-clustering consiste à trouver $k$ clusters couvrant à la fois l'ensemble des transactions et l'ensemble des items, sans aucun chevauchement sur les transactions ni sur les items.

Dans nos exemples de clustering conceptuel figurant à la table 3.2, la solution :

$$s_1 : \{A, B, D\}, \{C, F, G, H\}, \{E\}$$

est un co-clustering, car couvrant tous les items et toutes les transactions sans aucun chevauchement.

#### 7.1.1c Le soft clustering et le soft co-clustering

Le *soft clustering* consiste à relâcher :

- soit la relation de couverture : les transactions ne sont pas toutes couvertes, mais au moins $\delta_t$ transactions doivent être couvertes, avec $\delta_t \leq |\mathcal{T}|(= m)$.
- soit la relation de non-chevauchement : de petits chevauchements sont autorisés, mais chaque transaction $t$ ne peut apparaître dans plus de $\delta_o$ clusters, avec $\delta_o \geq 1$.

Le *soft co-clustering* consiste à relâcher la relation de couverture ou la relation de non-chevauchement, non seulement sur l'ensemble des transactions (comme pour le soft clustering), mais aussi sur l'ensemble des items :

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

**Exemple 7.1.**
Pour $\delta_t$=6, $\delta_o$=1, $\gamma_i$=7, $\gamma_o$=2, $s_r = [\{A, E\}, \{B, E, G\}, \{C, F, G, H\}]$ est un soft co-clustering puisque $D$ est le seul item manquant, les items $E$ et $G$ apparaissent deux fois et $\{t_1, t_7, t_8, t_9, t_{10}\}$ sont les transactions non couvertes.

### 7.1.2 Problème du tuilage

Le problème de tuilage, introduit dans le chapitre 3, consiste à chercher un ensemble fini de $k$ tuiles $\Phi = \{\tau(\phi_1), \ldots, \tau(\phi_k)\}$. Une tuile est dite maximale si elle est représentée par un motif fermé. Différentes variantes du problème de tuilage existent dans la littérature [Geerts *et al.*, 2004]. Nous rappelons brièvement dans les sections suivantes les variantes les plus connues ainsi que les modèles PLNE associés.

#### 7.1.2a k-Tuilage Maximum

Étant donnés une base de transactions $\mathcal{T}$ et un entier positif $k$, le problème de $k$-tuilage maximum consiste à trouver un tuilage composé d'au plus $k$ tuiles qui maximise l'aire sur l'ensemble de transactions.

#### 7.1.2b Tuilage Minimum

Étant donnée une base de données $\mathcal{T}$, le problème de tuilage minimum consiste à trouver un tuilage d'aire égal à la base de transactions, et composé d'un nombre minimal de tuiles.

#### 7.1.2c Top-k Tuiles

Étant donnés une base de données $\mathcal{T}$ et un entier positif $k$, le problème de top-k tuiles consiste à trouver les $k$ tuiles de plus grande aire sur l'ensemble de transactions.

## 7.2 Extraction des ensembles de motifs en PLNE

### 7.2.1 Encodage PLNE des problèmes de clustering

Parmi les différents encodages utilisés en PLNE, on distingue ceux utilisés dans des applications très répandues comme les problèmes de partitionnement, de recouvrement et d'affectation [Hoffman et Padberg, 2009]. En général, l'encodage PLNE utilisé pour ces problèmes repose sur un ensemble de variables booléennes avec des contraintes d'égalité pour les problèmes de partitionnement, des contraintes d'inégalité supérieure pour les problèmes de recouvrement ou des contraintes d'inégalité inférieure pour les problèmes d'affectation. Le programme linéaire pour ces problèmes est généralement défini comme suit :

$$\begin{array}{rl} \texttt{Maximize or Minimize} & c^T x \\ \texttt{Subject to} & Ax\ (\leq, =, or\ \geq)\ b \\ & x_i \in \{0,1\}, i = 1..n \end{array} \tag{7.1}$$

où, $x$ est le vecteur des $n$ variables de décision booléennes, $c$ est le vecteur des coûts associés aux variables de décision $x_j$ et correspond à une mesure donnée. $A$ est la matrice de coefficients de dimension $m \times n$, et $b$ est le vecteur second membre des contraintes de dimension $m \times 1$ . Résoudre un tel encodage PLNE consiste à trouver le meilleur partitionnement, recouvrement ou affectation relativement à une fonction objectif dans l'ensemble des solutions satisfaisables. Nous nous inspirons de ces modèles pour la modélisation de nos différentes requêtes.

Rappelons que $\mathcal{T}$ est un ensemble de $m$ transactions définies sur un ensemble $\mathcal{I}$ de $n$ items, $\mathcal{P}$ l'ensemble des $p$ motifs fermés ou concepts formels (relativement à la mesure de fréquence). Soit $a_{t,p}$ la matrice binaire de dimension $m \times |\mathcal{P}|$ telle que $(a_{t,p} = 1)$ si, et seulement si, $p \subseteq t$, i.e., la transaction $t$ appartient au cluster représenté par le motif fermé $p$. Nous adoptons ces notations dans les sections suivantes.

### 7.2.2 Clustering conceptuel dur

Le problème de clustering conceptuel peut être modélisé en un PLNE (voir la figure 7.1) avec $|\mathcal{P}|$ variables booléennes $(x_p,\ p \in \mathcal{P})$, où $(x_p = 1)$ si, et seulement si, le cluster représenté par le motif fermé $p$ appartient au clustering.

| | $\phi_1$ | $\phi_2$ | $\phi_3$ | $\phi_4$ | $\phi_5$ | $\phi_6$ | $\phi_7$ | $\phi_8$ | $\phi_9$ | $\phi_{10}$ | $\phi_{11}$ | $\phi_{12}$ | $\phi_{13}$ | $\phi_{14}$ | $\phi_{15}$ | $\phi_{16}$ | $\phi_{17}$ | $\phi_{18}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $t_1$ | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $t_2$ | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| $t_3$ | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| $t_4$ | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| $t_5$ | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| $t_6$ | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| $t_7$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| $t_8$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| $t_9$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| $t_{10}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| $t_{11}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |

TABLE 7.1 – Matrice $(a_{t,p})$ associée au jeu de données $\mathcal{T}$ (table 3.1).

Les contraintes (1) stipulent que toute transaction $t$ doit être couverte par un et un seul cluster $p$. La contrainte (2) détermine le nombre $(k = k_0)$ de clusters. La fonction objectif est définie en associant à chaque cluster $(p \in \mathcal{P})$ une valeur $v_p$ qui reflète l'intérêt (à maximiser) ou le coût (à minimiser) du cluster représenté par le motif $p$, par exemple, minimiser la diversité de chaque concept ou maximiser sa taille.

| | |
|---|---|
| `Optimiser` | $\sum_{p\in\mathcal{P}} v_p \,.\, x_p$ |
| `Sous contraintes` | (1) $\sum_{p\in\mathcal{P}} a_{t,p} \,.\, x_p = 1, \quad \forall t \in \mathcal{T}$<br>(2) $\sum_{p\in\mathcal{P}} x_p = k_0$<br>$x_p \in \{0,1\}, p \in \mathcal{P}$ |

FIGURE 7.1 – Modèle PLNE pour le clustering conceptuel (`PSILP-M1`).

**Exemple 7.2.**
Considérons l'ensemble $\mathcal{T}$ des transactions de la Table 3.1. La matrice $(a_{t,p})$ associée à $\mathcal{T}$ est illustrée par la Table 7.1. Si on considère ($v_p$=`size`$(p)$) la valeur attachée au cluster $p$, alors la solution qui maximise la fonction objectif est ($x_2$=$x_{12}$=$x_{15}$=1), toutes les autres variables $x_p$ étant égales à 0 (d'où, $k$=3). Cette solution correspond au clustering $s_1$ (voir la table 3.2).

Contrairement aux approches existantes qui nécessitent de fixer au départ le nombre $k$ de clusters, notre proposition, en revanche, permet de relâcher $k$. La contrainte (2') de la figure 7.2 permet de contrôler le nombre $k$ de clusters en spécifiant une borne inférieure $k_{min}$ et/ou une borne supérieure $k_{max}$. La relaxation de $k$ permet à notre approche une plus grande flexibilité.

### 7.2.3 Co-clustering

Le co-clustering consiste à trouver $k$ clusters couvrant à la fois l'ensemble des transactions et l'ensemble des items, mais sans aucun chevauchement sur les transactions ni sur les items. Soit $w_{i,p}$ la matrice binaire $n \times |\mathcal{P}|$ définie comme suit : $(w_{i,p} = 1)$ si et seulement l'item $i$ appartient au motif fermé $p$. La figure 7.2 donne le modèle PLNE du co-clustering. Les contraintes (3) indiquent que tout item $i$ doit être couvert par exactement un cluster représenté par un motif $p$.

| | |
|---|---|
| `Optimiser` | $\sum_{p\in\mathcal{P}} v_p \,.\, x_p$ |
| `Sous contraintes` | (1) $\sum_{p\in\mathcal{P}} a_{t,p} \,.\, x_p = 1, \quad \forall t \in \mathcal{T}$<br>(2) $k = \sum_{p\in\mathcal{P}} x_p$<br>(2′) $k_{min} \leq k \leq k_{max}$<br>(3) $\sum_{p\in\mathcal{P}} w_{i,p} \,.\, x_p = 1, \quad \forall i \in \mathcal{I}$<br>$k \in \mathbb{N}, \quad x_p \in \{0,1\}, p \in \mathcal{P}$ |

FIGURE 7.2 – Modèle PLNE pour le co-clustering (`PSILP-M3`).

**Exemple 7.3.**
La solution optimale $x_2{=}x_{12}{=}x_{15}{=}1$ et toutes les autres variables $x_p$ égales à 0, correspondant au clustering $s_1$ (voir la table 3.2) est un co-clustering puisque les clusters associés couvrent à la fois $\mathcal{I}$ et $\mathcal{T}$ sans chevauchements.

### 7.2.4 Soft clustering et soft co-clustering

En exploitant les reformulations linéaires des contraintes n-aires introduites dans le chapitre 6, il est possible de modéliser le soft clustering conceptuel, en introduisant $m$ variables booléennes $y_t, (t \in \mathcal{T})$, où $(y_t = 1)$ si, et seulement si, la transaction $t$ appartient à au moins un cluster représenté par un motif $p$. La figure 7.3 décrit le modèle PLNE associé.

Tout d'abord, comme $\sum_{t\in\mathcal{T}} y_t$ donne le nombre de transactions couvertes, la contrainte (5) stipule qu'au moins $\delta_t$ transactions doivent être couvertes. Puis, comme une transaction $t$ appartient à $\sum_{p\in\mathcal{P}} a_{t,p}x_p$ clusters, la contrainte (4) spécifie que toute transaction $t$ doit appartenir à au plus $\delta_o$ clusters. Notons que si $(y_t{=}0)$, alors la transaction $t$ n'est pas couverte et la contrainte (4) est satisfaite. Nous procédons de la même manière pour le soft co-clustering (voir la section 7.1.1b) en introduisant $n$ variables booléennes $z_i, (i \in \mathcal{I})$, où $(z_i = 1)$ si, et seulement si, l'item $i$ appartient à au moins un motif $p$.

| | |
|---|---|
| Optimiser | $\sum_{p\in\mathcal{P}} v_p \,.\, x_p$ |
| Sous contraintes | (4) $y_t \leq \sum_{p\in\mathcal{P}} a_{t,p} \,.\, x_p \leq \delta_o \,.\, y_t, \quad \forall t \in \mathcal{T}, \quad (C_{x,y})$ |
| | (5) $\sum_{t\in\mathcal{T}} y_t \geq \delta_t, \quad (C_{cov}^{\delta_t,\geq})$ |
| | (6) $z_i \leq \sum_{p\in\mathcal{P}} w_{i,p} \,.\, x_p \leq \gamma_o \,.\, z_i, \quad \forall i \in \mathcal{I}, \quad (C_{x,z})$ |
| | (7) $\sum_{i\in\mathcal{I}} z_i \geq \gamma_i, \quad (C_{cov}^{\gamma_i,\geq})$ |
| | $k = \sum_{p\in\mathcal{P}} x_p$ |
| | $k_{min} \leq k \leq k_{max}, \quad (C_{taille}^{k_{min},\geq}), (C_{taille}^{k_{max},\leq})$ |
| | $k \in \mathbb{N},$ |
| | $x_p \in \{0,1\}, \quad p \in \mathcal{P}$ |
| | $y_t \in \{0,1\}, \quad t \in \mathcal{T}$ |
| | $z_i \in \{0,1\}, \quad i \in \mathcal{I}$ |

FIGURE 7.3 – Modèle PLNE pour le soft co-clustering (PSILP-M3) .

Comme la somme $\sum_{i\in\mathcal{I}} z_i$ détermine le nombre d'items couverts, la contrainte (7) stipule qu'au moins $\gamma_i$ items doivent être couverts. Ensuite, comme un item $i$ appartient à $\sum_{p\in\mathcal{P}} w_{i,p}x_p$ clusters, la contrainte (6) exprime que tout item $i$ doit appartenir à au plus $\gamma_o$ clusters. Notons que si $(z_i{=}0)$, alors l'item $i$ n'est pas couvert et la contrainte (6) est satisfaite.

### 7.2.5 Problème du tuilage

Dans le problème de tuilage, le nombre $k$ de tuiles est très souvent borné par une valeur minimale et une valeur maximale, permettant ainsi au processus d'optimisation de déterminer la bonne valeur de $k$. Il faut noter que les tuiles peuvent se chevaucher, cependant un couple de transactions et items $(t, i)$ n'est compté qu'une et une seule fois dans le calcul de l'aire dans la fonction objectif.

Comme le montre la figure 7.4, cet encodage nécessite $(n \times m + 2)$ contraintes et $((n \times m) + \ell + 1)$ variables. Ceci constitue une limitation majeure lorsqu'il s'agit de traiter des bases de données très volumineuses. De ce fait, nous proposons un second modèle PLNE en considérant l'aire comme la somme des aires des tuiles individuelles dans l'ensemble $v_p$ $(p \in \mathcal{P})$, donnant ainsi *une approximation du problème du tuilage*. En effet, chaque couple $(t, i)$ couvert par l'ensemble des tuiles peut potentiellement être compté plusieurs fois.

| | |
|---|---|
| Maximiser | $\sum_{t\in\mathcal{T},i\in\mathcal{I}} q_{t,i}, \quad (C^{0,\uparrow}_{aire})$ |
| Sous contraintes | $q_{t,i} \leq \sum_{p\in\mathcal{P}} cq^p_{t,i} x_p \leq \lvert P\rvert \,.\, q_{t,i}, \forall t \in \mathcal{T}, \forall i \in \mathcal{I}, \quad (C_{x,q})$<br>$k = \sum_{p\in\mathcal{P}} x_p$<br>$k_{min} \leq k \leq k_{max}, \quad (C^{k_{min},\geq}_{taille}), (C^{k_{max},\leq}_{taille})$<br>$k \in \mathbb{N},$<br>$x_p \in \{0,1\}, \quad p \in \mathcal{P}$<br>$q_{i,t} \in \{0,1\}, i \in \mathcal{I} \wedge t \in \mathcal{T}$ |

FIGURE 7.4 – Problème du tuilage en PLNE (PSILP-M4).

L'approximation du problème de tuilage réutilise notre modèle du soft co-clustering (voir la section 7.2.4) où chaque variable correspond à une tuile et les coefficients de la fonction objectif sont instanciés par l'aire de chaque tuile.

À partir de la figure 7.3 l'approximation consiste à trouver un ensemble de tuiles couvrant à la fois l'ensemble des transactions et l'ensemble des items, avec de petits chevauchements sur les transactions et sur les items. En faisant ainsi, nous pouvons contrôler la redondance sur les couples $(t, i)$ comptés plusieurs fois dans le tuilage. Notons que si les seuils de chevauchement sur l'ensemble des transactions et l'ensemble des items sont fixés à 1, la somme des aires des tuiles individuelles est égale à l'aire d'un tuilage sans redondance sur les couples $(t, i)$.

### 7.2.6 Modélisation d'autres types de requêtes

Dans cette section, nous présentons quelques exemples de requêtes visant à illustrer la déclarativité de notre approche et l'utilisation des contraintes pour l'extraction des ensembles de motifs. Nous rappelons que toutes les requêtes formulées en PLNE requièrent une étape préalable d'extraction des motifs locaux. Étant donnée une base de molécules $\mathcal{T}$, constituée d'un ensemble de molécules actives et inactives, nous présentons deux exemples de requêtes complexes et nous montrons comment les modéliser par une combinaison des contraintes présentées dans le chapitre 6.

La requête $(Q_1)$ illustre une tâche très typique pour l'extraction des ensembles de motifs. Elle consiste à chercher des ensembles de motifs qui couvrent au moins $\theta_{active}$ molécules actives. Le support minimum garantit que l'ensemble des motifs est représentatif de l'ensemble de données tandis que la redondance maximale impose que les motifs individuels capturent des caractéristiques propres à certains sous-ensembles.

La requête $(Q_2)$ constitue un raffinement de $Q1$ en introduisant une contrainte de généralité. Par exemple : rechercher une représentation qui résume l'ensemble des sous-graphes extraits sur les données moléculaires, de sorte qu'un utilisateur non spécialisé en fouille, e.g. un chimiste, n'aura qu'un petit ensemble de fragments à évaluer. Dans ce cas, l'utilisateur aura déjà une idée sur l'ensemble des fragments qu'il veut voir, et que le reste soit étoffé. Cela pourrait être réalisé en cherchant un ensemble de motifs qui inclut un motif particulier (ou des motifs syntaxiques), qui soit plus général qu'un autre. Ceci peut s'effectuer en ajoutant une contrainte de généralité présentée dans le chapitre 6.

Les figures 7.5a et 7.5b donnent, respectivement, les modèles PLNE associés aux requêtes $Q1$ et $Q2$.

1. *Maximiser le support* ($z = \sum_{t\in\mathcal{T}} y_t$) sous contraintes de redondance maximale.
2. *Minimiser la redondance* ($z = \sum_{t\in\mathcal{T}} u_t$) sous contraintes de support minimal.

Les contraintes $(C_{x,y})$ et $(C_{x,u})$ portant sur les nouvelles variables $u$ et $y$ peuvent être fusionnées pour former une seule contrainte linéaire, notée $(C_{x,y,u})$ :

$$y_t + u_t \leq \sum_{p\in\mathcal{P}} a_{t,p} x_p \leq y_t + \lvert\mathcal{P}\rvert \,.\, u_t, y_t \geq u_t, \forall t \in \mathcal{T}$$

| Maximiser | $\sum_{t \in \mathcal{T}} y_t$ |
|---|---|
| Sous contraintes | $(C^{\theta,\leq}_{redd})$ |
| | $(C_{x,y,u})$ |
| | $x_p \in \{0,1\}, \forall p \in \mathcal{P}$ |
| | $y_t \in \{0,1\}, u_t \in \{0,1\}, \forall t \in \mathcal{T}$ |

(a) Modèle PLNE pour la requête $Q1$ (PSILP-M5) .

| Minimiser | $\sum_{t \in \mathcal{T}} u_t$ |
|---|---|
| Sous contraintes | $(C^{\theta,\geq}_{cov})$ |
| | $(C_{x,y,u})$ |
| | $x_p \in \{0,1\}, \forall p \in \mathcal{P}$ |
| | $y_t \in \{0,1\}, u_t \in \{0,1\}, \forall t \in \mathcal{T}$ |

(b) Modèle PLNE pour la requête $Q2$ (PSILP-M6) .

FIGURE 7.5 – Modèles PLNE des requêtes plus complexes.

## 7.3 Évaluation des modèles PLNE

Dans cette section, nous nous intéressons à l'évaluation des modèles PLNE de la section précédente. L'évaluation expérimentale est conçue pour répondre à des questions relatives à chaque application.

Pour le clustering conceptuel :

1. Comment (en termes de temps CPU) notre approche se compare-t-elle à l'approche de Guns et al. (KPatternSet) et à l'approche de Mueller et al. ?
2. Comment les clusters résultants se comparent-ils relativement aux différentes fonctions objectif ?
3. Compte tenu de la nature exacte de notre approche, comment les clusters obtenus et leurs descriptions se comparent-ils qualitativement à ceux qui résultent des approches heuristiques existantes ? Les deux approches avec lesquelles nous nous comparons sont celles de Pensa et al. (CDKMeans) et Perkowitz et al. (CM).

Pour le problème du tuilage :

1. Étant données les limitations en mémoire du modèle PLNE associé au problème de $k$-tuilage, comment les tuilages obtenus se comparent-ils qualitativement à ceux qui résultent du modèle approximatif ?
2. Comment les résultats de notre approche exacte se comparent-ils qualitativement avec ceux obtenus par (k-LTM) [Geerts *et al.*, 2004] ?
3. Comment les $k$-tuilages avec chevauchement se comparent-ils qualitativement avec les $k$-tuilages sans chevauchement ?

### 7.3.1 Protocole expérimental

Les expérimentations ont été menées sur les mêmes bases de transactions que celles utilisées dans [Guns *et al.*, 2013] et disponibles dans le dépôt de l'UCI. La table 4.5 du chapitre 4 présente les caractéristiques de ces bases de transactions.

Pour le clustering conceptuel, toutes les expérimentations ont été menées sur la même machine sous Linux x86_64 ayant un processeur AMD Opteron 6282SE avec 2.60 GHz et une mémoire RAM de 512 Go. Une autre machine sous Linux x86_64 a été utilisée pour évaluer les requêtes liées au problème du tuilage [26] possédant un processeur AMD Opteron 6174 avec 2,60 GHz et une mémoire vive de 256 Go. Au niveau de toutes les approches, un *Timeout* de 24 heures a été fixé.

Les expérimentations ont été réalisées avec les contraintes de fermeture comme seules contraintes locales. Nous avons utilisé les motifs fermés pour les concepts formels et aussi pour représenter les tuiles, car les motifs fermés couvrent une aire plus grande que leurs homologues non fermés.

26. A l'exception de l'approche k-LTM qui est implémentée sous Windows, et qui est utilisée pour une comparaison qualitative.

Pour les deux étapes de notre approche, nous avons utilisé :

1. LCM pour extraire l'ensemble des motifs locaux (motifs fermés),
2. CPLEX $v.12.4$ pour résoudre les différents modèles PLNE.

Pour évaluer la qualité d'un clustering, nous avons utilisé la mesure de similarité intra-cluster *(ICS)* et la mesure de dissimilarité inter-clusters *(ICD)*, les deux devant être la plus grande que possible.

La mesure de similarité entre deux transactions $t$ et $t'$ est définie par :

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

En conséquence, la mesure de similarité est définie par :

$$ICS(\phi_1, ..., \phi_k) = \frac{1}{2} \sum_{1 \leq i \leq k} ( \sum_{t,t' \in \phi_i, t \neq t'} s(t,t'))$$

et la mesure de dissimilarité par :

$$ICD(\phi_1, ..., \phi_k) = \sum_{1 \leq i < j \leq k} ( \sum_{t \in \phi_i, t' \in \phi_j} (1 - s(t,t')))$$

Afin d'évaluer les résultats sur le problème de tuilage, nous reprenons le modèle `PSILP-M3`, (modèle plus flexible des contraintes de couverture sur les items et sur les transactions) afin de trouver un ensemble de tuiles et de le comparer avec le modèle original $\texttt{PSILP} - \texttt{M4}$. Comme `PSILP-M3` requiert des seuils pour contrôler les relations de couverture et de chevauchement, nous avons considéré les trois configurations ci-dessous :

1. `PSILP-M3-a` avec une configuration similaire aux résultats de k-LTM, notamment, sur la couverture et le chevauchement.
2. `PSILP-M3-b` avec une couverture totale de l'ensemble de transactions ($\delta_t = m$), un chevauchement maximal de ($\delta_o = |\mathcal{P}|$), et une couverture minimale d'un item ($\gamma_i = 1$) sans chevauchement ($\gamma_o = 1$).
3. `PSILP-M3-c` avec une couverture totale de toutes les transactions ($\delta_t = m$) sans chevauchement ($\delta_o = 1$) et une couverture minimale d'au moins un item ($\gamma_i = 1$) avec chevauchement maximal de ($\gamma_o = |\mathcal{P}|$).

L'une des caractéristiques principales distinguant le problème de $k$-tuilage du reste des variantes est la notion d'aire. Par conséquent, nous utilisons la notion de *recall* d'un ensemble de motifs $\Phi$ afin d'évaluer les différents tuilages résultants des méthodes considérées. Le *recall* est défini comme le ratio entre le nombre de "1", i.e. le nombre de couples (transaction et item) couverts par l'ensemble de motifs, et le nombre total de "1" de la représentation binaire de la base de transactions $\mathcal{T}$. Cette mesure doit être aussi grande que possible :

$$recall(\Phi, \mathcal{T}) = \frac{\sum_{(t,i) \in area(\Phi, \mathcal{T})} \mathcal{T}_{ti}}{\sum_t \sum_i \mathcal{T}_{ti}}.$$

## 7.3.2 Passage à l'échelle

### 7.3.2a Comparaison des temps CPU sur le clustering conceptuel

La figure 7.6 compare les performances de notre approche PLNE (utilisant le modèle de `PSILP-M1`) avec celle basée sur la PPC (KPatternSet) en considérant les différentes bases de transactions sélectionnées, pour différentes valeurs de $k$, et par rapport au même critère d'optimisation sur la taille des motifs locaux sélectionnés dans le clustering conceptuel. Les temps CPU de `PSILP-M1` incluent l'étape de prétraitement.

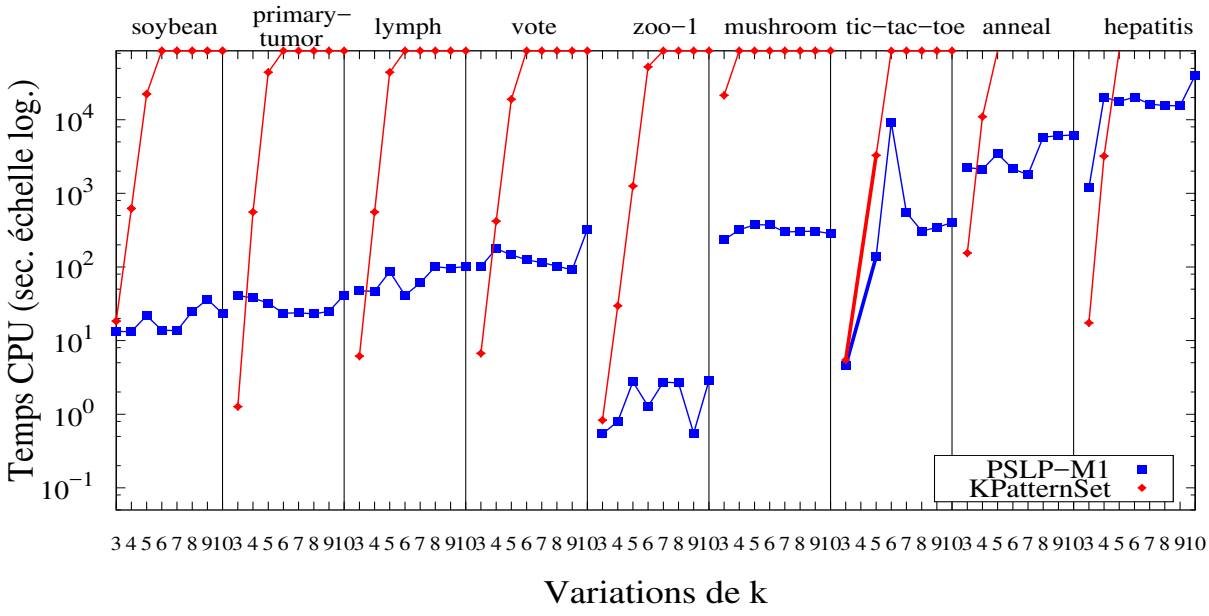



FIGURE 7.6 – Maximiser la taille de la description.

Pour ($k = 3$), KPatternSet domine `PSILP-M1` sauf sur les instances Soybean, Zoo-1 et Mushroom. Mais pour ($k \geq 4$), `PSILP-M1` est plus performant que KPatternSet avec plusieurs ordres de grandeur et ceci sur toutes les instances considérées. Enfin, KPatternSet ne parvient pas à trouver une solution dans le contrat de temps imposé pour ($k > 5$). Sur *Audiology*, les deux approches ne sont pas en mesure de trouver une solution. Ceci est en partie du au nombre élevé de motifs locaux (fermés) $10^6$ en comparaison avec les autres instances où le nombre de motifs locaux varie de $10^3$ jusqu'à $10^5$. En considérant le critère d'optimisation basé sur la minimisation de la diversité des motifs locaux (voir la figure 7.7), on peut formuler les mêmes observations que précédemment. Notons que sur la base *Tic-Tac-Toe*, il n'existe aucun clustering conceptuel réalisable pour ($k = 4$).

#### 7.3.2b Comparaison des temps CPU sur le problème de tuilage

Les tableaux 7.2 et 7.3 comparent, en termes de temps de CPU, notre modèle `PSILP-M4` et l'approche KPatternSet sur différentes instances, ceci pour plusieurs valeurs de $k$. Nous rapportons également la valeur du *recall*. Les temps CPU de `PSILP-M4` incluent l'étape de pré-traitement. `PSILP-M4` est plus performant que KPatternSet sur toutes les bases considérées : KPatternSet atteint le *timeout* pour des valeurs de $k \geq 4$. Pour les valeurs de $k$ où le $k$-tuilage optimal est trouvé par l'approche de KPatternSet, `PSILP-M4` est nettement plus rapide que KPatternSet (plusieurs ordres de grandeur).

Comparons maintenant le modèle `PSILP-M4` avec les modèles `PSILP-M3-a` et `PSILP-M3-b`. `PSILP-M4` trouve la solution optimale sur la plupart des bases, exceptées *Mushroom*, *Hepatitis* et *Anneal*. Les résultats obtenus par `PSILP-M3-a` restent relativement proches en termes de *recall*, en particulier pour des valeurs de $k \leq 5$. De plus, `PSILP-M3-a` est plus rapide, en particulier sur les bases *Lymph* et *Vote* (avec un speed-up allant jusqu'à 660).

Enfin, comparons `PSILP-M3-a` et `PSILP-M3-b`. On observe que `PSILP-M3-b` obtient des valeurs de *recall* plus élevées pour *Hepatitis*, *Primary Tumor* et *Vote*. Cela est dû au fait que `PSILP-M3-b` permet un chevauchement entre les tuiles avec une redondance plus élevée (voir la table 7.5). En termes de temps CPU,

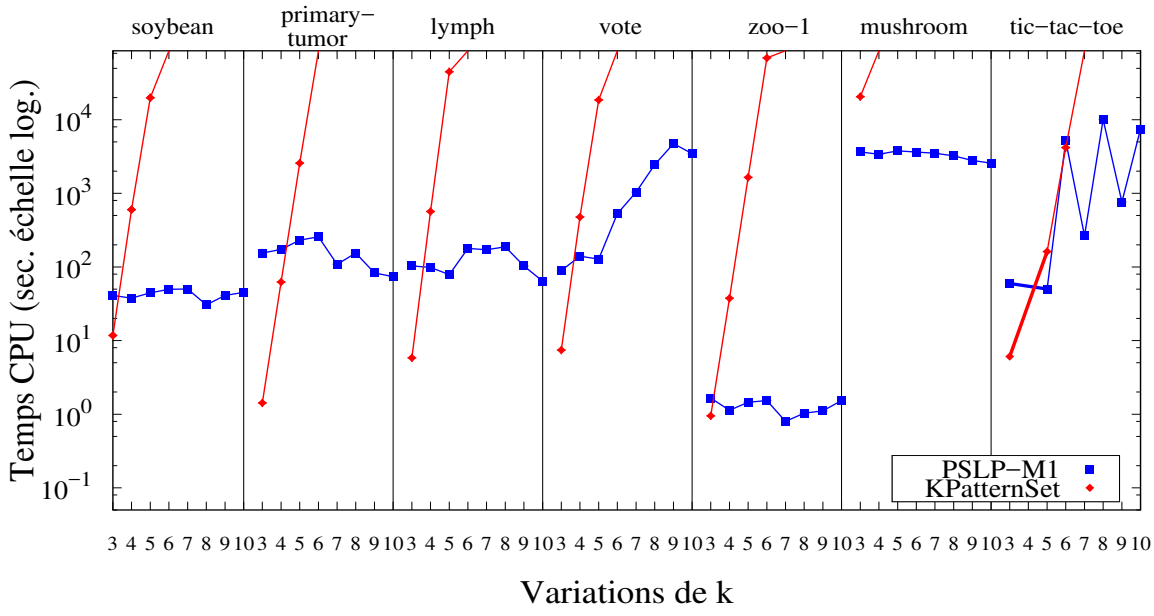



FIGURE 7.7 – Minimiser la diversité des motifs.

`PSILP-M3-b` est plus rapide que `PSILP-M3-a` sur 51 configurations parmi 64, avec un *speed-up* entre 1 et 4 pour 27 configurations, et entre 7 et 25 pour 9 configurations. Sur la base *Mushroom*, aucun tuilage de $k$ tuiles n'existe en utilisant le modèle `PSILP-M3-b`. Ces résultats montrent que `PSILP-M3-a` et `PSILP-M3-b` atteignent de bonnes valeurs de *recall* avec moins d'espace mémoire requis, contrairement au modèle de `PSILP-M4`. Ceci montre l'intérêt d'utiliser `PSILP-M3-b` comme une alternative pratique pour trouver une approximation d'un $k$-tuilage optimal.

### 7.3.3 Relaxation du nombre $k$ de clusters

Pour évaluer l'intérêt de relâcher le nombre de motifs $k$, la table 7.4 compare les performances de `PSILP-M1` avec $k$ relâché (`PSILP-M1-Relâché`) avec `PSILP-M1`, en termes de la meilleure valeur trouvée pour $k$ (Col. 2 vs. Col. 5) et en termes du temps CPU (Col. 4 vs. Col .7). Les valeurs rapportées dans la Col. 5 correspondent à la meilleure valeur de $k$ trouvée parmi l'ensemble $\{3, 4, \ldots, 10\}$ maximisant la taille des descriptions dans l'ensemble de motifs. Les deux modèles obtiennent la même meilleure valeur pour $k$. En effet, il y a un biais vers un $k$ près de $k_{max}$ lorsqu'on utilise la taille de la description comme critère d'optimisation. En comparant les temps CPU, `PSILP-M1-Relâché` est souvent *plus rapide*, notamment sur les deux bases les plus difficiles, *Anneal* et *Hepatitis* (speed-up allant jusqu'à 3.84).

Nous nous sommes également comparés avec l'approche proposée dans [Mueller et Kramer, 2010a] où le nombre $k$ de clusters doit être fixé au préalable, contrairement à notre approche. Pour comparer les deux approches, nous avons implanté la moyenne des similarités internes des clusters basée sur les distances (i.e., $s(x, y)$) ainsi que la fonction objectif utilisée dans leur travail (dénoté `PSILP-M1-Distance`).

La table 7.4 compare `PSILP-M1-Relâché` et `PSILP-M1-Distance`, en termes de la meilleure valeur trouvée pour $k$ et en termes de temps CPU. Les deux approches retrouvent globalement les mêmes meilleures valeurs de $k$ et obtiennent, sur les 7 premières bases des temps CPU comparables. Par contre, `PSILP-M1-Relâché`

| $\mathcal{D}$ | Items | Trans. | $k$ | Recall | | | | | Time (s) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | (1) | (2) | (3) | (4) | (5) | (1) | (2) | (3) | (4) | (5) |
| Zoo-1 | 34 | 101 | 3 | 0.48 | 0.38 | 0.38 | 0.48 | 0.46 | 8.99 | 5.87 | 1.57 | 2,418 | |
| | | | 4 | 0.57 | 0.47 | 0.48 | TO | 0.56 | 14.38 | 4.53 | 3.67 | TO | |
| | | | 5 | 0.63 | 0.54 | 0.52 | TO | 0.62 | 25.11 | 4.49 | 2.99 | TO | |
| | | | 6 | 0.68 | 0.55 | 0.55 | TO | 0.67 | 34.96 | 10.82 | 2.98 | TO | |
| | | | 7 | 0.73 | 0.57 | 0.58 | TO | 0.71 | 22.85 | 5.81 | 3.49 | TO | |
| | | | 8 | 0.78 | 0.68 | 0.61 | TO | 0.75 | 25.56 | 6.96 | 3.16 | TO | |
| | | | 9 | 0.82 | 0.69 | 0.64 | TO | 0.79 | 35.07 | 8.04 | 3.59 | TO | |
| | | | 10 | 0.85 | 0.69 | 0.66 | TO | 0.82 | 26.21 | 6.91 | 3.23 | TO | 0.99 |
| Lymph | 66 | 148 | 3 | 0.43 | 0.42 | 0.35 | TO | 0.42 | 299.91 | 50.92 | 152.54 | TO | |
| | | | 4 | 0.48 | 0.38 | 0.39 | TO | 0.47 | 954.74 | 143.84 | 152.13 | TO | |
| | | | 5 | 0.52 | 0.44 | 0.42 | TO | 0.51 | 1,629 | 97.88 | 152.72 | TO | |
| | | | 6 | 0.55 | 0.37 | 0.45 | TO | 0.54 | 4,261 | 175.01 | 152.52 | TO | |
| | | | 7 | 0.58 | 0.40 | 0.46 | TO | 0.57 | 6,027 | 155.19 | 153.15 | TO | |
| | | | 8 | 0.61 | 0.40 | 0.48 | TO | 0.59 | 5,197 | 47.68 | 146.13 | TO | |
| | | | 9 | 0.63 | 0.40 | 0.49 | TO | 0.61 | 7,313 | 45.51 | 148.06 | TO | |
| | | | 10 | 0.65 | 0.44 | 0.50 | TO | 0.63 | 28,779 | 43.56 | 148.51 | TO | 1321 |
| Primary-tumor | 29 | 336 | 3 | 0.46 | 0.40 | 0.36 | 0.46 | 0.46 | 405.91 | 33.25 | 82.26 | 41,496 | |
| | | | 4 | 0.54 | 0.48 | 0.47 | TO | 0.53 | 680.64 | 74.99 | 88.60 | TO | |
| | | | 5 | 0.59 | 0.47 | 0.55 | TO | 0.57 | 3,807 | 43.93 | 68.65 | TO | |
| | | | 6 | 0.64 | 0.44 | 0.60 | TO | 0.61 | 3,863 | 92.28 | 58.76 | TO | |
| | | | 7 | 0.68 | 0.46 | 0.64 | TO | 0.65 | 6,407 | 154.82 | 61.58 | TO | |
| | | | 8 | 0.71 | 0.50 | 0.67 | TO | 0.68 | 86,429 | 41.75 | 53.93 | TO | |
| | | | 9 | 0.74 | 0.54 | 0.70 | TO | 0.71 | 86,425 | 65.50 | 65.13 | TO | |
| | | | 10 | 0.77 | 0.52 | 0.73 | TO | 0.73 | 86,423 | 43.70 | 58.23 | TO | 4.9 |
| Soybean | 47 | 630 | 3 | 0.44 | 0.42 | 0.33 | 0.44 | 0.43 | 127.94 | 15.31 | 8.35 | 32,485 | |
| | | | 4 | 0.50 | 0.47 | 0.37 | TO | 0.49 | 147.11 | 24.38 | 6.96 | TO | |
| | | | 5 | 0.55 | 0.49 | 0.41 | TO | 0.53 | 141.70 | 22.10 | 6.43 | TO | |
| | | | 6 | 0.59 | 0.47 | 0.43 | TO | 0.58 | 144.27 | 19.25 | 6.08 | TO | |
| | | | 7 | 0.62 | 0.47 | 0.45 | TO | 0.60 | 157.41 | 14.11 | 5.90 | TO | |
| | | | 8 | 0.65 | 0.45 | 0.47 | TO | 0.62 | 358.05 | 16.30 | 9.65 | TO | |
| | | | 9 | 0.67 | 0.46 | 0.48 | TO | 0.64 | 342.79 | 14.65 | 8.83 | TO | |
| | | | 10 | 0.69 | 0.48 | 0.49 | TO | 0.66 | 734.20 | 9.42 | 6.28 | TO | 7 |

TABLE 7.2 – Comparaison des différentes approches (TO : TimeOut ; – : Pas de solution ; (1) : PSILP-M4 ; (2) : PSILP-M3-a ; (3) : PSILP-M3-b ; (4) : KPatternSet ; (5) : k-LTM).

est nettement plus efficace sur les deux bases les plus difficiles, *Anneal* et *Hepatitis*. La figure 7.11 montre que les clusterings trouvés en relâchant $k$ sont proches de ceux trouvés avec un $k$ fixé.

Enfin, dans la table 7.4, on peut remarquer que l'étape de pré-traitement est négligeable par rapport à l'étape de résolution, à l'exception des deux bases *Anneal* et de *Hepatitis* en raison du nombre élevé de motifs locaux (fermés) extraits.

## 7.3.4 Qualité des résultats

### 7.3.4a Analyse qualitative des clusterings conceptuels

Pour évaluer la qualité des clusterings, nous reportons sur les figures 7.8, 7.9, 7.10 et 7.11 les mesures de l'*ICS* et de l'*ICD* pour chaque clustering trouvé par les approches heuristiques et par nos modèles PLNE. Les points correspondent à des clusterings optimaux obtenus pour différentes valeurs de $k$ sur l'ensemble des bases considérées pour l'évaluation. Une couleur spécifique est attribuée à chaque base. Une étiquette représentant la valeur de $k$ est associée à certains points avec la même couleur. Notons que nous n'avons pas reporté sur les figures 7.9 et 7.10 les points qui se trouvent sur le front Pareto puisqu'ils sont incomparables.

| $\mathcal{D}$ | Items | Trans. | $k$ | Recall | | | | | Time (s) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | (1) | (2) | (3) | (4) | (5) | (1) | (2) | (3) | (4) | (5) |
| Tic-Tac-Toe | 24 | 958 | 3 | 0.15 | 0.15 | 0.11 | TO | 0.15 | 24.49 | 8.93 | 5.98 | TO | |
| | | | 4 | 0.20 | 0.20 | 0.16 | TO | 0.20 | 23.67 | 6.11 | 5.05 | TO | |
| | | | 5 | 0.25 | 0.25 | 0.21 | TO | 0.25 | 25.81 | 5.43 | 5.82 | TO | |
| | | | 6 | 0.29 | 0.29 | 0.27 | TO | 0.29 | 24.45 | 5.92 | 4.91 | TO | |
| | | | 7 | 0.33 | 0.33 | 0.33 | TO | 0.33 | 43.17 | 5.53 | 2.70 | TO | |
| | | | 8 | 0.38 | 0.38 | 0.38 | TO | 0.38 | 44.76 | 5.44 | 3.82 | TO | |
| | | | 9 | 0.42 | 0.42 | 0.42 | TO | 0.42 | 42.76 | 4.87 | 3.50 | TO | |
| | | | 10 | 0.46 | 0.46 | 0.46 | TO | 0.46 | 41.98 | 5.16 | 3.35 | TO | 0.1 |
| Vote | 45 | 435 | 3 | 0.32 | 0.30 | 0.06 | TO | 0.31 | 3,020 | 81.75 | 34.36 | TO | |
| | | | 4 | 0.37 | 0.35 | 0.19 | TO | 0.36 | 1,947 | 84.99 | 46.09 | TO | |
| | | | 5 | 0.42 | 0.38 | 0.33 | TO | 0.40 | 4,269 | 732.10 | 49.42 | TO | |
| | | | 6 | 0.46 | 0.39 | 0.40 | TO | 0.44 | 3,651 | 83.48 | 43.23 | TO | |
| | | | 7 | 0.50 | 0.38 | 0.45 | TO | 0.47 | 13,831 | 282.56 | 39.13 | TO | |
| | | | 8 | 0.53 | 0.37 | 0.49 | TO | 0.51 | 8,671 | 819.33 | 33.03 | TO | |
| | | | 9 | 0.56 | 0.42 | 0.53 | TO | 0.54 | 15,152 | 881.76 | 36.10 | TO | |
| | | | 10 | 0.60 | 0.43 | 0.57 | TO | 0.57 | 13,974 | 701.37 | 30.53 | TO | 3.3 |
| Mushroom | 112 | 8124 | 3 | TO | 0.34 | – | TO | 0.34 | TO | 431.82 | – | TO | |
| | | | 4 | TO | 0.40 | – | TO | 0.40 | TO | 667.61 | – | TO | |
| | | | 5 | TO | 0.46 | – | TO | 0.46 | TO | 780.05 | – | TO | |
| | | | 6 | TO | 0.48 | – | TO | 0.50 | TO | 572.54 | – | TO | |
| | | | 7 | TO | 0.49 | – | TO | 0.53 | TO | 502.91 | – | TO | |
| | | | 8 | TO | 0.44 | – | TO | 0.56 | TO | 1,302 | – | TO | |
| | | | 9 | TO | 0.48 | – | TO | 0.58 | TO | 763.20 | – | TO | |
| | | | 10 | TO | 0.51 | – | TO | 0.60 | TO | 479.45 | – | TO | 666.4 |
| hepatitis | 66 | 137 | 3 | TO | 0.27 | 0.25 | TO | 0.28 | TO | 1,261 | 604.16 | TO | |
| | | | 4 | TO | 0.33 | 0.33 | TO | 0.34 | TO | 910.38 | 1,007.34 | TO | |
| | | | 5 | TO | 0.37 | 0.38 | TO | 0.39 | TO | 1,310 | 725.38 | TO | |
| | | | 6 | TO | 0.42 | 0.42 | TO | 0.44 | TO | 887.81 | 781.88 | TO | |
| | | | 7 | TO | 0.43 | 0.46 | TO | 0.48 | TO | 1,327 | 616.90 | TO | |
| | | | 8 | TO | 0.45 | 0.49 | TO | 0.52 | TO | 2,372 | 736.19 | TO | |
| | | | 9 | TO | 0.45 | 0.52 | TO | 0.55 | TO | 3,046 | 625.16 | TO | |
| | | | 10 | TO | 0.45 | 0.54 | TO | 0.57 | TO | 16,295 | 656.40 | TO | 6103 |

TABLE 7.3 – Comparaison des différentes approches (2). (`TO` : TimeOut ; – : no solution ; (1) : `PSILP-M4` ; (2) : `PSILP-M3-a` ; (3) : `PSILP-M3-b` ; (4) : KPatternSet ; (5) : k-LTM).

La figure 7.8 détaille les résultats obtenus pour les deux critères d'optimisation de `PSILP-M1`. À l'exception des bases *Vote* et *Tic-Tac-Toe*, la mesure de la diversité pénalise le critère *ICS* en faveur de l'*ICD* pour atteindre des valeurs plus élevées. Ceci est un indicateur sur le choix des clusters plus équilibrés : l'*ICS* est nécessairement limité par le nombre de transactions par cluster, cependant l'*ICD* augmente s'il n'y a plus de transactions dans d'autres clusters.

La mesure de taille montre un comportement inverse. Ceci est un indicateur de solutions caractérisées par peu de clusters contenant une grande proportion des transactions et d'autres clusters contenant moins de transactions.

Pour nous comparer à CDKMeans et CM [27], nous devons, en premier lieu, identifier un clustering qui servira de point de comparaison. Pour cela :

1. Nous avons exécuté chaque méthode 100 fois pour lisser les effets de l'initialisation aléatoire de k-Means,
2. Nous avons ensuite regroupé le résultat dans des *classes d'équivalence* dans lesquelles les clusterings sont formés approximativement sur les mêmes clusters.
3. Nous avons enfin considéré la plus grande classe d'équivalence et choisi un représentant (un des clusterings de cette classe puisqu'ils ont la même composition) de cette classe. En cas de présence de plusieurs grandes classes d'équivalence, nous avons considéré un représentant de chaque classe.

27. Pour `CM`, nous avons utilisé `SimpleKMeans` et `JRip` dans `Weka`. Nous avons ré-implémenté `CDKMeans`.

| Instance | PSILP-M1 avec $k$ relaxé – Maximiser la taille de la description | | | PSILP-M1 avec $k$ fixé – Maximiser la taille de la description | | |
|---|---|---|---|---|---|---|
| | meilleur $k$ | Temps CPU (s.) (1) | Temps CPU (s.) (2) | meilleur $k$ | Temps CPU (s.) (1) | Temps CPU (s.) (2) |
| Soybean | 10 | 0.74 | 15.52 | 10 | 0.74 | 21.31 |
| Primary-tumor | 10 | 1.98 | 40.62 | 10 | 1.98 | 35.84 |
| Lymph | 10 | 3.38 | 25 | 10 | 3.38 | 93.96 |
| Vote | 10 | 4.42 | 191.12 | 10 | 4.42 | 304.18 |
| Zoo-1 | 10 | 0.07 | 6.22 | 10 | 0.07 | 2.73 |
| Mushroom | 10 | 17.1 | 154.75 | 10 | 17.1 | 163.67 |
| Tic-Tac-Toe | 10 | 0.38 | 420.75 | 10 | 0.38 | 398.86 |
| Anneal | 10 | 236.85 | 1,493.38 | 10 | 236.85 | 5,744.03 |
| Hepatitis | 10 | 113.84 | 20,825.8 | 10 | 113.84 | 40,014.5 |

| Instance | PSILP-M1-Distance Maximiser la moyenne des similarités du contenu de chaque cluster [Mueller et Kramer, 2010a] | | |
|---|---|---|---|
| | meilleur $k$ | Temps CPU (s.) (1) | Temps CPU (s.) (2) |
| Soybean | 10 | 0.74 | 8.52 |
| Primary-tumor | 8 | 1.98 | 35.4 |
| Lymph | 10 | 3.38 | 149.69 |
| Vote | 9 | 4.42 | 66.21 |
| Zoo-1 | 9 | 0.07 | 0.35 |
| Mushroom | 10 | 17.1 | 169.04 |
| Tic-Tac-Toe | 10 | 0.38 | 209.4 |
| Anneal | 10 | 236.85 | 5,744.03 |
| Hepatitis | 10 | 113.84 | 37,745.7 |

TABLE 7.4 – L'impact de la relaxation du nombre de clusters $k$ ((1) étape de prétraitement (2) étape de résolution).

Les expérimentations montrent combien il est difficile de contrôler les résultats des heuristiques : les clusterings représentatifs de l'approche CM ne couvrent jamais toutes les transactions de la base, et ont souvent des clusters qui se chevauchent. La figure 7.9 montre que les clusterings calculés par CM ont un désavantage qualitatif par rapport aux clusterings trouvés par PSILP-M1 (les points de CM sont toujours dominés par ceux de PSILP-M1). Afin d'avoir une comparaison équitable, nous avons exécuté le modèle PSILP-M3 avec des configurations permettant la même quantité de non-couverture et de chevauchement. Si plusieurs grandes classes d'équivalence apparaissent, nous avons choisi le clustering qui maximise les deux critères l'*ICS* et l'*ICD* tout en utilisant sa couverture comme contrainte pour PSILP-M3. La figure 7.10 montre que notre approche peut avoir des résultats proches de ceux obtenus par les heuristiques, renforçant ainsi l'aspect de la déclarativité.

CDKMeans est encore plus difficile à contrôler par l'utilisateur : lorsque tous les motifs fermés sont utilisés, l'algorithme attribue, généralement, toutes les transactions à un ou deux clusters, peu importe la valeur de $k$. L'utilisation de la contrainte de fréquence améliore un peu cette situation, même si $|\mathcal{C}| \leq k$ pour $k \geq 4$. De plus, les clusterings ne parviennent pas, dans certains cas, à se stabiliser finissant ainsi en 100 différents clusterings. Finalement, la représentation particulière des clusters dans CDKMeans peut entraîner des clusters avec des descriptions sans aucune transaction assignée.

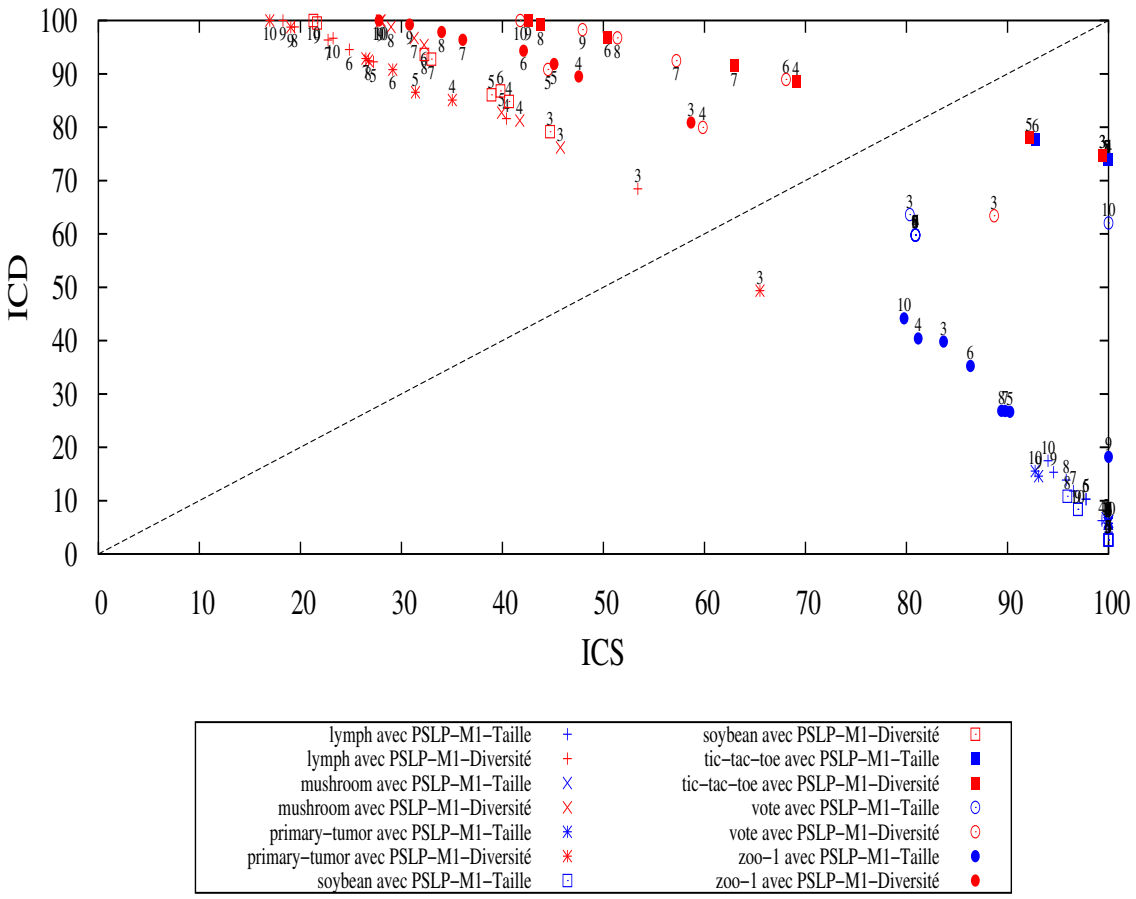



FIGURE 7.8 – Comparaison de la qualité des clusterings résultants de `PSILP-M1` : Diversité vs. Taille de la description.

#### 7.3.4b Analyse qualitative des tuilages

Avant d'aborder la comparaison qualitative entre `PSILP` et l'algorithme k-LTM, nous décrivons trois points dans lesquels k-LTM est différent :

1. k-LTM utilise une heuristique (meilleure rapidité mais aucune garantie d'optimalité de la solution).
2. k-LTM est basé sur une extraction itérative, il exploite des informations sur les tuiles déjà trouvées pour compléter le tuilage.
3. Le $(k+1)$-tuilage est un sur-ensemble du $k$-tuilage, i.e. différentes solutions peuvent être trouvées.

Les tables 7.2, 7.3 et 7.5 montrent que `PSILP-M4` présente de meilleures valeurs de *recall* que ceux obtenus par k-LTM, mais nécessitant plus de temps CPU pour trouver l'optimum. Les temps CPU de k-LTM sont indiqués dans la dernière colonne [28]. Pour la base la plus difficile, *Anneal*, les deux méthodes sont incapables de trouver des solutions. k-LTM a un léger avantage par rapport à `PSILP-M3-b` sur trois bases.

28. Ces temps CPU sont donnés à titre indicatif. Notons que l'implémentation de k-LTM est disponible uniquement sous Windows, toutes les autres approches sont sous Linux.

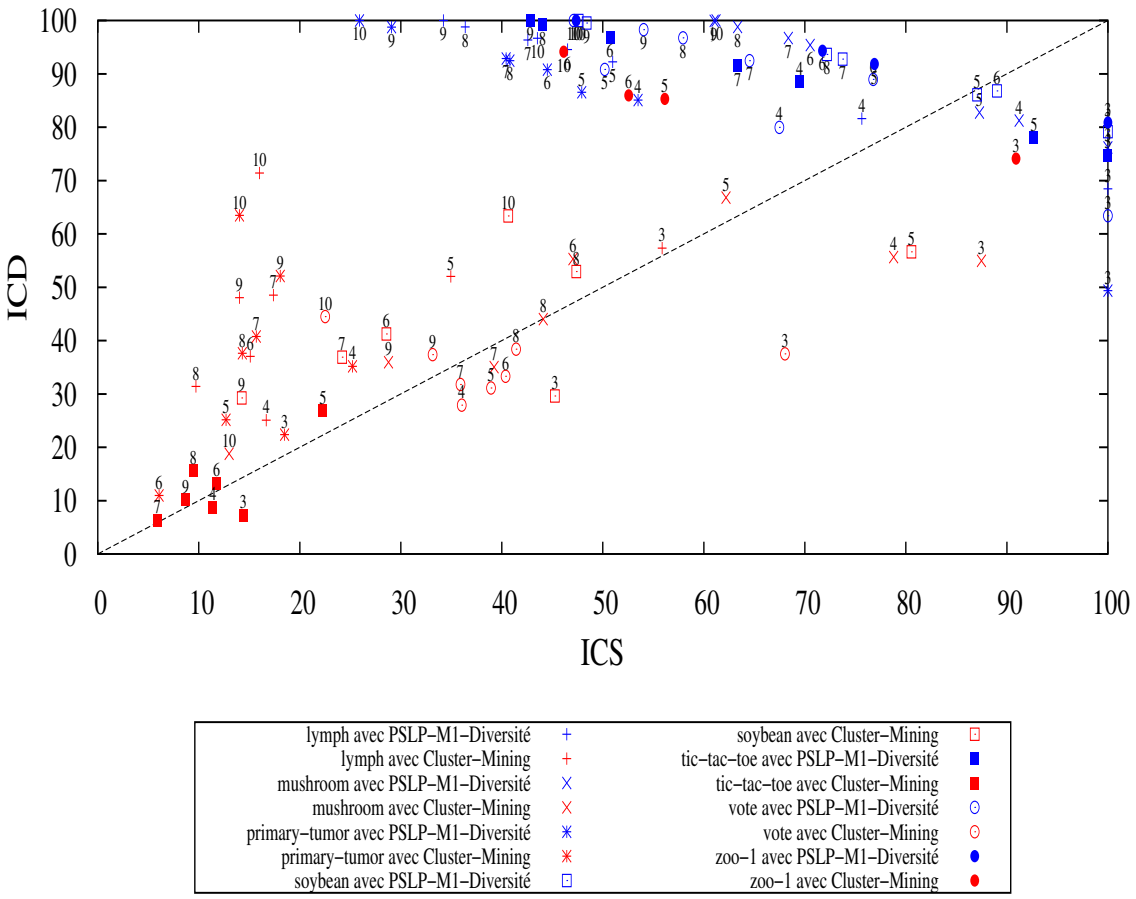



FIGURE 7.9 – Comparaison de la qualité des clusterings résultants de `PSILP-M1` avec la mesure de diversité vs. Cluster Mining (CM).

Nos dernières expérimentations comparent `PSILP-M3-c` avec `PSILP-M3-b` et k-LTM. Rappelons que `PSILP-M3-c` permet d'extraire $k$ tuiles sans chevauchement. La table 7.5 montre quatre situations distinctes de *recall* et de redondance pour les trois approches.

— La colonne 4 reporte la redondance dans le $k$-tuilage mesurée par $red(\Phi, \mathcal{T})/sup(\Phi, \mathcal{T})$.

— La colonne 5 (resp. 6) reporte le pourcentage d'items redondants.

En général, `PSILP-M3-b` et k-LTM atteignent des valeurs de *recall* plus élevées que `PSILP-M3-c`. Ceci n'est pas surprenant puisque le tuilage trouvé par `PSILP-M3-b` consiste en des tuiles larges qui se chevauchent sur l'ensemble des transactions contrairement à `PSILP-M3-c` (voir colonne 4). `PSILP-M3-c` comble cette lacune en couvrant plus d'items que les autres approches, et recherche des tuiles se chevauchant plus sur les items. Cet ajustement de caractéristiques résultantes est la force des approches déclaratives.

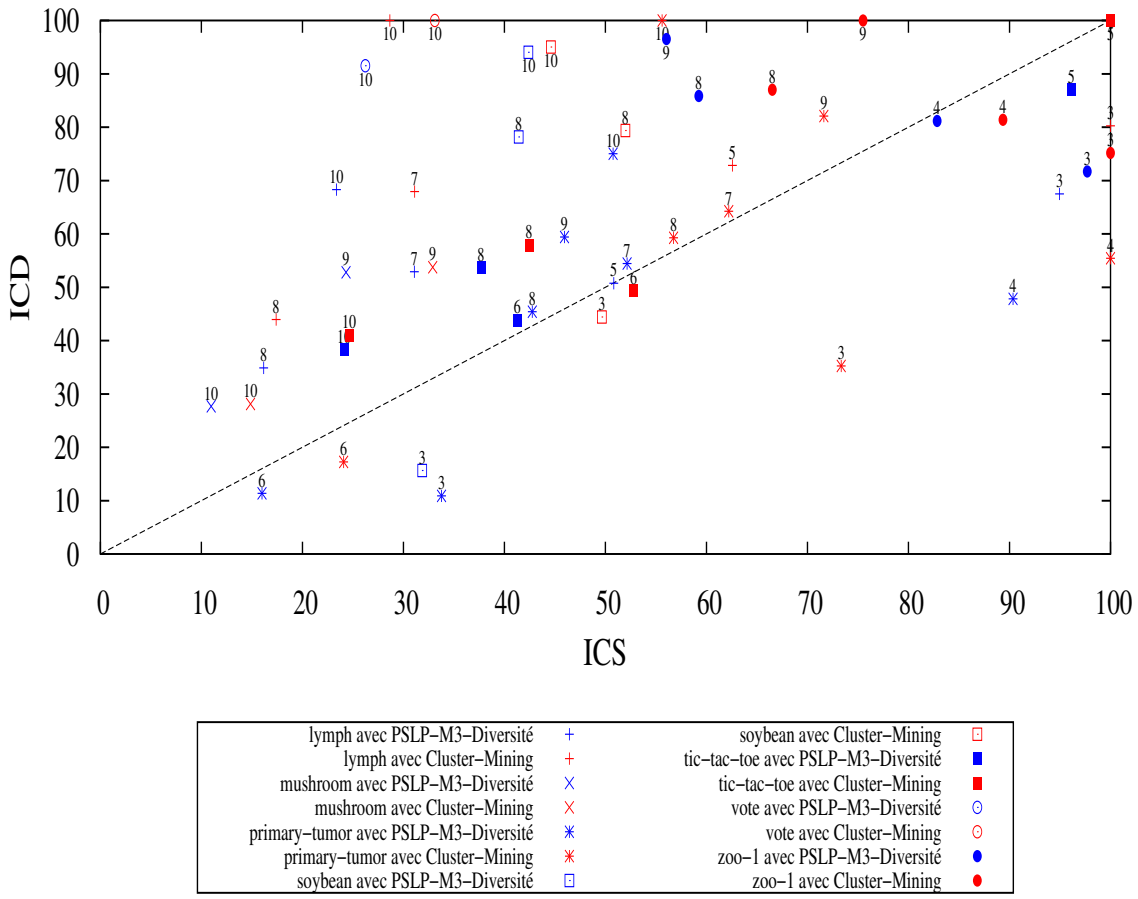



FIGURE 7.10 – Comparaison de la qualité des clusterings résultants de `PSILP-M3` avec la mesure de diversité vs. Cluster Mining (CM).

## 7.4 Conclusion

Dans ce chapitre, nous avons comparé notre approche PLNE pour l'extraction des ensembles de motifs avec des approches heuristiques et des approches déclaratives exactes. Notre méthode permet à l'utilisateur d'extraire des ensembles de motifs qui optimisent un citère donné tout en satisfaisant un ensemble de contraintes. Le solveur PLNE garantit que l'ensemble de motifs extrait est optimal relativement à la fonction objectif et aux contraintes définies par l'utilisateur. Les expérimentations réalisées sur différentes bases de transactions ont montré l'efficacité de notre approche sur deux applications déjà étudiées pour l'extraction des ensembles de motifs (clustering conceptuel et tuilage).

Notre approche est suffisamment générique pour pouvoir modéliser de nombreux problèmes tels que le clustering (cas général), la classification ou la synthèse des motifs. La flexibilité de notre approche est clairement une étape majeure vers le développement de systèmes interactifs en fouille de données. Nous pouvons également améliorer le processus des résolution des différents modèles PLNE en exploitant les techniques de génération de colonnes afin de pouvoir traiter des bases de transactions de très grande taille (cf les perspectives développées au Chapitre 8).

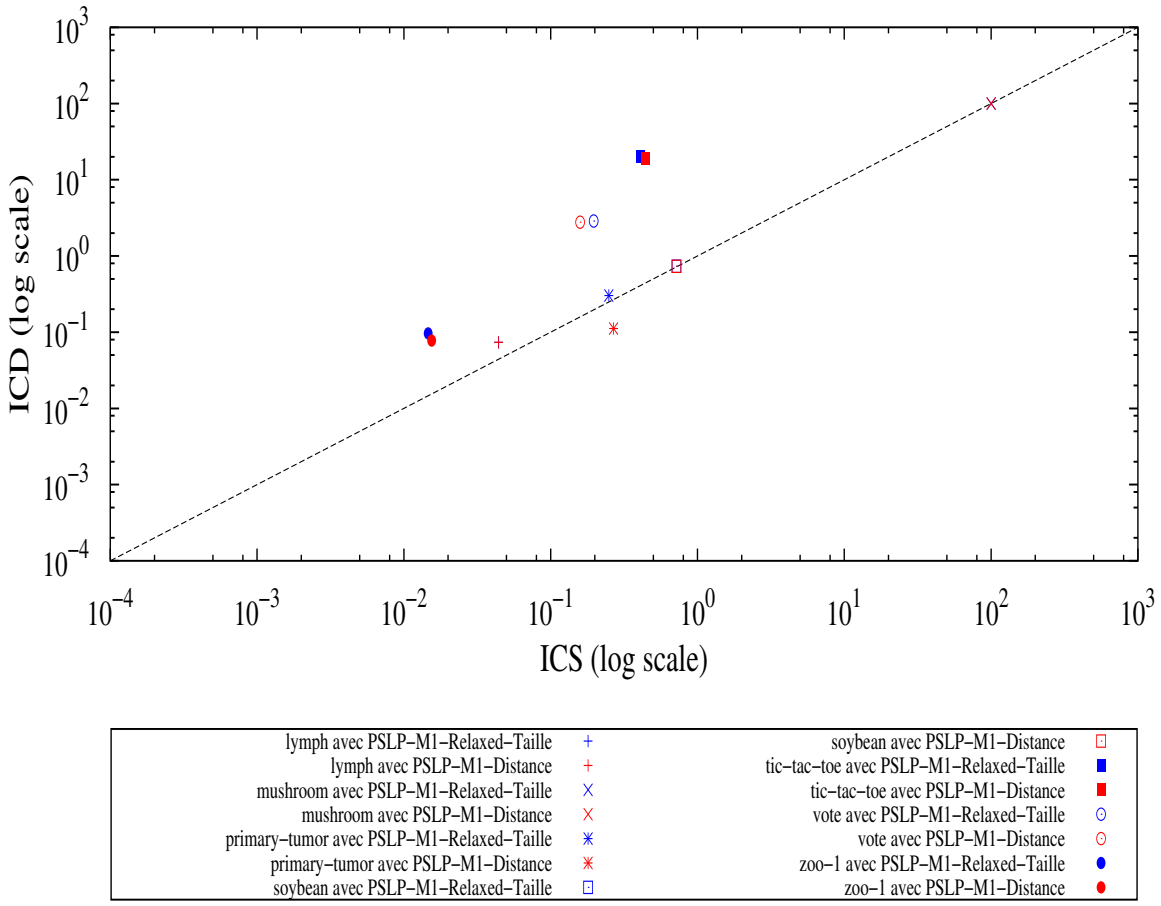



FIGURE 7.11 – Comparaison de la qualité des clusterings résultants de `PSILP-M1-Relâché` vs. `PSILP-M1-Distance` en optimisant la taille de la descritpions des motifs.

| $\mathcal{D}$ | $k$ | **Recall** | | **Redundant Trans.** | | | **Redundant Items** | | | **Coverage Items** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | (3) | (3') | (3) | (3') | (5) | (3) | (3') | (5) | (3) | (3') | (5) |
| **Mushroom** | 3 | - | 0.23 | - | 0.00 | 0.55 | - | 0.50 | 0.31 | - | 0.08 | 0.16 |
| | 4 | - | 0.31 | - | 0.00 | 0.64 | - | 0.32 | 0.28 | - | 0.16 | 0.18 |
| | 5 | - | 0.36 | - | 0.00 | 0.81 | - | 0.42 | 0.29 | - | 0.22 | 0.24 |
| | 6 | - | 0.42 | - | 0.00 | 0.91 | - | 0.44 | 0.41 | - | 0.21 | 0.25 |
| | 7 | - | 0.46 | - | 0.00 | 0.92 | - | 0.59 | 0.55 | - | 0.24 | 0.25 |
| | 8 | - | 0.51 | - | 0.00 | 0.93 | - | 0.66 | 0.53 | - | 0.24 | 0.26 |
| | 9 | - | 0.54 | - | 0.00 | 0.93 | - | 0.69 | 0.51 | - | 0.24 | 0.27 |
| | 10 | - | 0.56 | - | 0.00 | 0.93 | - | 0.53 | 0.50 | - | 0.32 | 0.28 |
| **Soybean** | 3 | 0.33 | 0.28 | 0.85 | 0.00 | 0.59 | 0.00 | 0.30 | 0.41 | 0.16 | 0.20 | 0.25 |
| | 4 | 0.37 | 0.34 | 0.90 | 0.00 | 0.64 | 0.00 | 0.54 | 0.37 | 0.20 | 0.26 | 0.34 |
| | 5 | 0.41 | 0.40 | 0.93 | 0.00 | 0.74 | 0.00 | 0.56 | 0.47 | 0.22 | 0.36 | 0.36 |
| | 6 | 0.43 | 0.44 | 0.96 | 0.00 | 0.85 | 0.00 | 0.57 | 0.44 | 0.24 | 0.46 | 0.38 |
| | 7 | 0.45 | 0.48 | 0.98 | 0.00 | 0.85 | 0.00 | 0.52 | 0.52 | 0.26 | 0.42 | 0.40 |
| | 8 | 0.47 | 0.52 | 1.00 | 0.00 | 0.90 | 0.00 | 0.50 | 0.57 | 0.26 | 0.52 | 0.40 |
| | 9 | 0.48 | 0.54 | 1.00 | 0.00 | 0.93 | 0.00 | 0.62 | 0.55 | 0.28 | 0.52 | 0.42 |
| | 10 | 0.49 | 0.66 | 1.00 | 0.00 | 0.95 | 0.00 | 0.69 | 0.52 | 0.28 | 0.52 | 0.44 |
| **Primary-tumor** | 3 | 0.36 | 0.15 | 0.87 | 0.00 | 0.60 | 0.00 | 0.33 | 0.25 | 0.23 | 0.19 | 0.41 |
| | 4 | 0.47 | 0.22 | 0.90 | 0.00 | 0.71 | 0.00 | 0.38 | 0.38 | 0.35 | 0.26 | 0.44 |
| | 5 | 0.55 | 0.28 | 0.94 | 0.00 | 0.82 | 0.00 | 0.33 | 0.42 | 0.42 | 0.39 | 0.48 |
| | 6 | 0.60 | 0.33 | 0.96 | 0.00 | 0.84 | 0.00 | 0.50 | 0.53 | 0.48 | 0.39 | 0.51 |
| | 7 | 0.64 | 0.38 | 0.98 | 0.00 | 0.89 | 0.00 | 0.44 | 0.62 | 0.52 | 0.52 | 0.55 |
| | 8 | 0.67 | 0.42 | 1.00 | 0.00 | 0.91 | 0.00 | 0.42 | 0.75 | 0.52 | 0.61 | 0.55 |
| | 9 | 0.70 | 0.45 | 1.00 | 0.00 | 0.92 | 0.00 | 0.45 | 0.87 | 0.52 | 0.65 | 0.55 |
| | 10 | 0.73 | 0.48 | 1.00 | 0.00 | 0.95 | 0.00 | 0.45 | 0.77 | 0.58 | 0.71 | 0.62 |
| **Anneal** | 3 | - | 0.36 | - | 0.00 | TO | - | 0.47 | TO | - | 0.34 | TO |
| | 4 | - | 0.46 | - | 0.00 | TO | - | 0.64 | TO | - | 0.35 | TO |
| | 5 | - | 0.52 | - | 0.00 | TO | - | 0.74 | TO | - | 0.37 | TO |
| | 6 | - | 0.57 | - | 0.00 | TO | - | 0.67 | TO | - | 0.46 | TO |
| | 7 | - | 0.59 | - | 0.00 | TO | - | 0.84 | TO | - | 0.47 | TO |
| | 8 | - | 0.62 | - | 0.00 | TO | - | 0.71 | TO | - | 0.56 | TO |
| | 9 | - | 0.64 | - | 0.00 | TO | - | 0.78 | TO | - | 0.59 | TO |
| | 10 | - | 0.65 | - | 0.00 | TO | - | 0.82 | TO | - | 0.59 | TO |

TABLE 7.5 – Comparaison Qualitative des tuilages. ( (3) : `PSILP-M3-b` ; (3') : `PSILP-M3-c` ; (5) : k-LTM).

# Chapitre 8

# Conclusion et perspectives

## Sommaire


**8.1 Conclusions** . . . . . **125**
8.1.1 Méthode parallèle coopérative exploitant la décomposition arborescente dans VNS . 126
8.1.2 Stratégies parallèles répliquées pour DGVNS . . . . . 126
8.1.3 Clustering Conceptuel en PLNE . . . . . 126
8.1.4 PLNE pour l'extraction des ensembles de motifs . . . . . 127
**8.2 Perspectives** . . . . . **127**


## 8.1 Conclusions

Cette thèse est l'une des toutes premières à s'intéresser à l'apport du parallélisme sur la décomposition arborescente au sein de VNS. Notre motivation repose sur les constats suivants :

— La décomposition arborescente permet de décomposer le graphe de contraintes d'un problème en clusters formant un graphe acyclique. Les clusters associés à la décomposition arborescente offrent une "carte" des régions pertinentes de l'espace de recherche pouvant être explorées en parallèle par différents processus.

— Bien que la méthode séquentielle DGVNS soit efficace sur des instances de taille modérée, beaucoup d'instances de grande taille nécessitent des temps de calcul plus importants à cause de la faible diversification de DGVNS.

— L'exploitation du parallélisme pour booster la résolution des problèmes d'optimisation est un domaine de recherche très actif. Le modèle *maître-travailleurs* est un modèle très utilisé dans le domaine des méta-heuristiques parallèles. Il a été appliqué avec succès en mode de communication asynchrone dans VNS, offrant une bonne efficacité par rapport à la version séquentielle.

Nous avons aussi travaillé sur la modélisation et la résolution exacte de problématiques complexes liées à l'extraction des ensembles de motifs. Cette thèse est l'une des premières qui s'intéresse à l'utilisation de la PLNE dans un schéma de résolution en deux étapes pour résoudre des problèmes d'extraction des ensembles de motifs, et plus particulièrement le problème du clustering conceptuel et le problème du tuilage.

Nos travaux ont été motivés par les trois préoccupations suivantes :

**Déclarativité.** Offrir un cadre permettant la définition/modélisation de nouvelles contraintes pour traiter de nouveaux problèmes.

**Passage à l'échelle.** Avoir un cadre pratique efficace permettant de traiter des bases de données à l'optimalité, considérées jusque-là comme difficiles par les approches déclaratives existantes.

**Optimalité.** Trouver des solutions optimales optimisant un critère donné afin d'améliorer la qualité des ensembles de motifs par rapport à ceux trouvés par les approches heuristiques.

Sur ces deux volets de thèse, nos principales contributions sont les suivantes :

### 8.1.1 Méthode parallèle coopérative exploitant la décomposition arborescente dans VNS

Nous avons proposé une première stratégie de parallélisation de `DGVNS`, appelée `CPDGVNS` (Cooperative Parallel `DGVNS`), qui consiste à explorer, en parallèle, tous les clusters fournis par une décomposition arborescente . `CPDGVNS` repose sur une architecture *maître-travailleurs*, où le processus maître mémorise, met à jour, et communique la meilleure solution courante, alors que les processus travailleurs gèrent, individuellement, l'exploration des clusters. Les processus travailleurs coopèrent par une communication asynchrone en échangeant des informations sur la meilleure solution trouvée. Ce schéma de communication permet, d'une part, de garantir une gestion locale des clusters, et d'autre part, d'initialiser l'exploration de ces clusters avec des solution différentes, favorisant ainsi une meilleure diversification. Les expérimentations ont été effectuées sur plusieurs instances de trois problèmes réels, et la méthode `CPDGVNS` a montré son efficacité par rapport à la méthode séquentielle.
*Publications associées : [Ouali* et al.*, 2014a, Ouali* et al.*, 2014b].*

### 8.1.2 Stratégies parallèles répliquées pour DGVNS

La principale limitation de `CPDGVNS` est que les travailleurs doivent effectuer un certain nombre d'itérations avant de partager leur meilleure solution avec le maître. Cela rend la coopération avec le maître moins fréquente, et limite donc la diversification de l'exploration de l'espace de recherche par les travailleurs. Pour améliorer la diversification dans `CPDGVNS`, nous avons proposé deux nouvelles stratégies de parallélisation (1) *Replicated Asynchronous DGVNS* (`RADGVNS`) et (2) *Replicated Synchronous DGVNS* (`RSDGVNS`) permettant la production rapide de solutions intermédiaires pour alimenter l'échange d'informations entre les travailleurs :

— `RADGVNS`. Comme pour `CPDGVNS`, les processus travailleurs coopèrent d'une façon asynchrone en échangeant des informations sur la meilleure solution courante. Toutefois, et contrairement à `CPDGVNS`, chaque processus travailleur exécute une seule itération, en complétant la solution partielle de son cluster.

— `RSDGVNS`. Pour étudier l'impact de la synchronisation, nous avons proposé sur la même architecture une autre version dont la communication entre le processus maître et les processus travailleurs est synchrone. Dans `RSDGVNS`, le processus maître attend la fin de tous les processus travailleurs afin d'évaluer le prochain mouvement.

*Publication associée : [Ouali* et al.*, 2015].*

### 8.1.3 Clustering Conceptuel en PLNE

Nous avons proposé une approche hybride combinant la PLNE et la fouille de concepts. Notre approche est complète et tire profit du cadre général de la PLNE (en procurant un haut niveau de flexibilité et d'expressivité) et des heuristiques spécialisées pour l'exploration et l'extraction de données (pour améliorer les temps de calcul). Les expérimentations menées ont montré l'apport de notre proposition par rapport (i) aux approches à base de contraintes et (ii) aux heuristiques spécialisées. *Publication associée : [Ouali* et al.*, 2016].*

### 8.1.4 PLNE pour l'extraction des ensembles de motifs

L'extraction des ensembles de motifs est une tâche difficile et les rares approches existantes utilisent des stratégies heuristiques. Nous avons généralisé l'approche proposée pour le clustering conceptuel afin de définir de nouvelles contraintes, chacune étant modélisée par des expressions linéaires. Nous avons étudié les limitations de la PLNE pour la modélisation et la résolution de nouvelles requêtes, en particulier sur le problème de tuilage. Nous avons alors proposé des formulations approximatives de certaines contraintes, comme la contrainte d'aire d'un ensemble de motifs. Cela a permis le passage à l'échelle, tout en obtenant des ensembles de motifs de qualité très proche de celle des contraintes originales. Les expérimentations ont montré que le cadre de la PLNE avec une démarche en deux étapes est une approche très efficace pour l'extraction des ensembles de motifs par rapport aux approches existantes. *Publication associée : [Ouali* et al.*, 2017].*

## 8.2 Perspectives

Nous détaillons, dans cette section, plusieurs perspectives à court et à moyen terme.

**Vers un cadre unifié pour les approches parallèles** Il serait intéressant de disposer d'un cadre unifié permettant l'hybridation entre la stratégie RADGVNS et la stratégie RSDGVNS. Notre idée est d'adapter le niveau de synchronisation lors de la recherche et selon l'amélioration de la solution courante. Nous pensons que ce cadre unifié pourrait améliorer sensiblement nos résultats actuels et permettre d'attaquer des instantes plus difficiles issues des problèmes de conception des protéines.

**Vers des méthodes de recherche locale complètes.** Une distinction habituelle des méthodes d'optimisation combinatoire s'effectue entre méthode complète et méthode incomplète. La plupart des méthodes complètes s'appuient sur l'exploration d'un arbre de recherche, alors que les méthodes incomplètes s'appuient principalement sur une recherche locale passant d'une solution à une autre solution avec une légère modification de cette dernière. Nous étudions actuellement une variante itérative de DGVNS capable de prouver l'optimalité quand la diversification est maximale avec un arbre de recherche non borné. Notre objectif est d'obtenir un meilleur compromis entre preuve d'optimalité et comportement *anytime* de la méthode (qualité des solutions vs. temps de calcul).

**Génération de colonnes.** Bien que notre approche pour l'extraction des ensembles de motifs ait donné de bons résultats sur des bases de données de tailles conséquentes, elle est toutefois limitée par le nombre de motifs locaux obtenus à l'issue de la première phase. En effet, la modélisation PLNE peut entraîner une consommation en mémoire trop importante, dès lors que le nombre de motifs extraits devient très grand. Une solution à ce problème serait d'utiliser la génération de colonnes. Tout d'abord, la résolution par génération de colonnes se fait sur un modèle maître ayant un nombre restreint de colonnes (dans notre cas motifs locaux) bien plus petit que le modèle complet. Par conséquent, la consommation mémoire des modèles sera plus faible comparée au modèle complet. De plus, durant la résolution, nous pouvons avoir une approximation de la distance entre la solution courante et la solution optimale grâce à une borne inférieure calculée sur le problème dual. Il serait alors possible de stopper la résolution quand la solution courante aurait atteint une qualité donnée (sans avoir à rechercher ni à prouver l'optimalité).

**Modélisation linéaire d'autres types de contraintes.** Dans cette thèse, nous avons considéré quelques problèmes d'extraction d'ensembles de motifs ayant des contraintes n-aires les plus répandues. Nous envisageons de considérer d'autres contraintes plus complexes. Nous souhaitons aussi prendre en considération la dimension multicritères dans nos modèles PLNE.

# Bibliographie


[Agrawal *et al.*, 1993] AGRAWAL, R., IMIELIŃSKI, T. et SWAMI, A. (1993). Mining association rules between sets of items in large databases. *SIGMOD Rec.*, 22(2):207–216. 50

[Alba *et al.*, 2013] ALBA, E., LUQUE, G. et NESMACHNOW, S. (2013). Parallel metaheuristics : recent advances and new trends. *ITOR*, 20(1):1–48. 30, 32

[Aloise *et al.*, 2012] ALOISE, D., HANSEN, P. et LIBERTI, L. (2012). An improved column generation algorithm for minimum sum-of-squares clustering. *Math. Prog.*, 131(1-2):195–220. 61

[Aupetit et Slimane, 2016] AUPETIT, S. et SLIMANE, M. (2016). *Nature Inspires New Algorithms*, pages 263–286. Springer International Publishing, Cham. 30

[Babaki *et al.*, 2014] BABAKI, B., GUNS, T. et NIJSSEN, S. (2014). Constrained clustering using column generation. *In CPAIOR'2014*, page Forthcoming. xiii, 61

[Bayardo, 1998] BAYARDO, Jr., R. J. (1998). Efficiently mining long patterns from databases. *SIGMOD Rec.*, 27(2):85–93. 51, 52

[Bensana *et al.*, 1999] BENSANA, E., LEMAITRE, M. et VERFAILLIE, G. (1999). Earth observation satellite management. *Constraints*, 4(3):293–299. 19, 67

[Berkhin, 2006] BERKHIN, P. (2006). A survey of clustering data mining techniques. *In Grouping Multidimensional Data - Recent Advances in Clustering*, pages 25–71. 59, 106

[Bessière et Cordier, 1993] BESSIÈRE, C. et CORDIER, M. (1993). Arc-consistency and arc-consistency again. *In* FIKES, R. et LEHNERT, W. G., éditeurs : *Proceedings of the 11th National Conference on Artificial Intelligence. Washington, DC, USA, July 11-15, 1993.*, pages 108–113. AAAI Press / The MIT Press. 13

[Bessière et Régin, 1996] BESSIÈRE, C. et RÉGIN, J. (1996). MAC and combined heuristics : Two reasons to forsake FC (and cbj ?) on hard problems. *In* FREUDER, E. C., éditeur : *Proceedings of the Second International Conference on Principles and Practice of Constraint Programming, Cambridge, Massachusetts, USA, August 19-22, 1996*, volume 1118 de *Lecture Notes in Computer Science*, pages 61–75. Springer. 11

[Bessière et Régin, 1997] BESSIÈRE, C. et RÉGIN, J. (1997). Arc consistency for general constraint networks : Preliminary results. *In Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI 97, Nagoya, Japan, August 23-29, 1997, 2 Volumes*, pages 398–404. Morgan Kaufmann. 13

[Bessière et Régin, 2001] BESSIÈRE, C. et RÉGIN, J. (2001). Refining the basic constraint propagation algorithm. *In* NEBEL, B., éditeur : *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001*, pages 309–315. Morgan Kaufmann. 13

[Blum *et al.*, 2011] BLUM, C., PUCHINGER, J., RAIDL, G. R. et ROLI, A. (2011). Hybrid metaheuristics in combinatorial optimization : A survey. *Applied Soft Computing*, 11(6):4135 – 4151. 30, 32

[Boddy et Dean, 1994] BODDY, M. et DEAN, T. L. (1994). Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, 67(2):245 – 285. 43

[Boulicaut et Bykowski, 2000] BOULICAUT, J. et BYKOWSKI, A. (2000). Frequent closures as a concise representation for binary data mining. *In* TERANO, T., LIU, H. et CHEN, A. L. P., éditeurs : *Knowledge Discovery and Data Mining, Current Issues and New Applications, 4th Pacific-Asia Conference, PADKK 2000, Kyoto, Japan,*

*April 18-20, 2000, Proceedings*, volume 1805 de *Lecture Notes in Computer Science*, pages 62–73. Springer. 53

[Boulicaut *et al.*, 2003] BOULICAUT, J.-F., BYKOWSKI, A. et RIGOTTI, C. (2003). Free-sets : A condensed representation of boolean data for the approximation of frequency queries. *Data Mining and Knowledge Discovery*, 7(1):5–22. 51, 53

[Brimberg *et al.*, 2006] BRIMBERG, J., UROŠEVIĆ, D. et MLADENOVIĆ, N. (2006). Variable neighborhood search for the vertex weighted k-cardinality tree problem. *European Journal of Operational Research*, 171(1):74 – 84. 34

[Burdick *et al.*, 2001] BURDICK, D., CALIMLIM, M. et GEHRKE, J. (2001). Mafia : A maximal frequent itemset algorithm for transactional databases. *In Proceedings of the 17th International Conference on Data Engineering*, pages 443–452, Washington, DC, USA. IEEE Computer Society. 51

[Cabon *et al.*, 1999] CABON, B., de GIVRY, S., LOBJOIS, L., SCHIEX, T. et WARNERS, J. (1999). Radio link frequency assignment. *Constraints*, 4(1):79–89. 19, 64

[Calders et Goethals, 2002] CALDERS, T. et GOETHALS, B. (2002). Mining all non-derivable frequent itemsets. *In* ELOMAA, T., MANNILA, H. et TOIVONEN, H., éditeurs : *Principles of Data Mining and Knowledge Discovery, 6th European Conference, PKDD 2002, Helsinki, Finland, August 19-23, 2002, Proceedings*, volume 2431 de *Lecture Notes in Computer Science*, pages 74–85. Springer. 51

[Calders et Goethals, 2007] CALDERS, T. et GOETHALS, B. (2007). Non-derivable itemset mining. *Data Min. Knowl. Discov.*, 14(1):171–206. 51

[Cambazard, 2006] CAMBAZARD, H. (2006). *Résolution de problèmes combinatoires par des approches fondées sur la notion d'explication. (Explanation-based algorithms in Constraint Programming)*. Thèse de doctorat, University of Nantes, France. 11

[Chu *et al.*, 2008] CHU, G., STUCKEY, P. J. et HARWOOD, A. (2008). Pminisat : A parallelization of minisat 2.0. Rapport technique. 75

[Colorni *et al.*, 1992] COLORNI, A., DORIGO, M. et MANIEZZO, V. (1992). An investigation of some properties of an "ant algorithm". *In* MÄNNER, R. et MANDERICK, B., éditeurs : *Parallel Problem Solving from Nature 2, PPSN-II, Brussels, Belgium, September 28-30, 1992*, pages 515–526. Elsevier. 32

[Cooper, 2003] COOPER, M. C. (2003). Reduction operations in fuzzy or valued constraint satisfaction. *Fuzzy Sets and Systems*, 134(3):311–342. 16, 18

[Cooper *et al.*, 2008] COOPER, M. C., de GIVRY, S., SÁNCHEZ, M., SCHIEX, T. et ZYTNICKI, M. (2008). Virtual arc consistency for weighted CSP. *In* FOX, D. et GOMES, C. P., éditeurs : *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pages 253–258. AAAI Press. 16

[Crainic et Nourredine, 2005] CRAINIC, T. et NOURREDINE (2005). Parallel meta-heuristics applications. *In* ALBA, E., éditeur : *Parallel Metaheuristics : A New Class of Algorithms*, chapitre 19, pages 447–494. John Wiley & Sons, Hoboken. 75

[Crainic *et al.*, 2004] CRAINIC, T. G., GENDREAU, M., HANSEN, P. et MLADENOVIĆ, N. (2004). Cooperative parallel variable neighborhood search for the p-median. *Journal of Heuristics*, 10(3):293–314. 40, 41

[Crainic et Toulouse, 1998] CRAINIC, T. G. et TOULOUSE, M. (1998). *Parallel Metaheuristics*, pages 205–251. Springer US, Boston, MA. 39, 75

[Dao *et al.*, 2013a] DAO, T., DUONG, K. et VRAIN, C. (2013a). A declarative framework for constrained clustering. *In ECML PKDD*, volume 8190 de *LNCS*, pages 419–434. 61

[Dao *et al.*, 2013b] DAO, T.-B., DUONG, K.-C. et VRAIN, C. (2013b). A declarative framework for constrained clustering. *In ECML/PKDD (3)*, pages 419–434. xii

[Davidson *et al.*, 2010] DAVIDSON, I., RAVI, S. et SHAMIS, L. (2010). A SAT-based framework for efficient constrained clustering. *In SDM*, pages 94–105. xiii

[de Carvalho, 2002] de CARVALHO, J. V. (2002). A note on branch-and-price algorithms for the one-dimensional cutting stock problems. *Computational Optimization and Applications*, 21(3):339–340. 9

[de Givry *et al.*, 2005a] de GIVRY, S., HERAS, F., ZYTNICKI, M. et LARROSA, J. (2005a). Existential arc consistency : Getting closer to full arc consistency in weighted csps. *In* KAELBLING, L. P. et SAFFIOTTI, A., éditeurs : *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*, pages 84–89. Professional Book Center. 15

[de Givry *et al.*, 2005b] de GIVRY, S., HERAS, F., ZYTNICKI, M. et LARROSA, J. (2005b). Existential arc consistency : Getting closer to full arc consistency in weighted csps. *In* KAELBLING, L. P. et SAFFIOTTI, A., éditeurs : *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*, pages 84–89. Professional Book Center. 16, 18, 19

[de Givry *et al.*, 2006] de GIVRY, S., SCHIEX, T. et VERFAILLIE, G. (2006). Exploiting tree decomposition and soft local consistency in weighted CSP. *In Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA*, pages 22–27. AAAI Press. xi, 76

[Debruyne et Bessière, 2001] DEBRUYNE, R. et BESSIÈRE, C. (2001). Domain filtering consistencies. *J. Artif. Intell. Res. (JAIR)*, 14:205–230. 11

[Dechter, 1990] DECHTER, R. (1990). Enhancement schemes for constraint processing : Backjumping, learning, and cutset decomposition. *Artif. Intell.*, 41(3):273–312. 11

[Dechter, 2003] DECHTER, R. (2003). *Constraint processing*. Elsevier Morgan Kaufmann. 11

[Dechter et Meiri, 1989] DECHTER, R. et MEIRI, I. (1989). Experimental evaluation of preprocessing techniques in constraint satisfaction problems. *In* SRIDHARAN, N. S., éditeur : *Proceedings of the 11th International Joint Conference on Artificial Intelligence. Detroit, MI, USA, August 1989*, pages 271–277. Morgan Kaufmann. 11

[Dechter et Pearl, 1989] DECHTER, R. et PEARL, J. (1989). Tree clustering for constraint networks. *Artificial Intelligence*, 38(3):353 – 366. xi, 76

[Dolan et Moré, 2002] DOLAN, E. D. et MORÉ, J. J. (2002). Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213. 43

[Dorigo et Stützle, 2004] DORIGO, M. et STÜTZLE, T. (2004). *Ant Colony Optimization*. Bradford Company, Scituate, MA, USA. 31

[Fargier et Lang, 1993] FARGIER, H. et LANG, J. (1993). *Uncertainty in constraint satisfaction problems : A probabilistic approach*, pages 97–104. Springer Berlin Heidelberg, Berlin, Heidelberg. 13

[Farley, 1990] FARLEY, A. A. (1990). A note on bounding a class of linear programming problems, including cutting stock problems. *Oper. Res.*, 38(5):922–923. 9

[Fisher, 1987] FISHER, D. H. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2(2):139–172. 105

[Fontaine, 2013] FONTAINE, M. (2013). *Apport de la décomposition arborescente pour les méthodes de type VNS. (Exploiting Tree Decomposition for Guiding Neighborhoods Exploration for VNS)*. Thèse de doctorat, University of Caen Normandy, France. xiii, xv, 22, 24, 38, 39, 41, 63, 76

[Fontaine *et al.*, 2013] FONTAINE, M., LOUDNI, S. et BOIZUMAULT, P. (2013). Exploiting tree decomposition for guiding neighborhoods exploration for VNS. *RAIRO - Operations Research*, 47(2):91–123. xi, 76

[Freuder et Wallace, 1992] FREUDER, E. C. et WALLACE, R. J. (1992). Partial constraint satisfaction. *Artif. Intell.*, 58(1-3):21–70. 13

[Frost et Dechter, 1995] FROST, D. et DECHTER, R. (1995). Look-ahead value ordering for constraint satisfaction problems. *In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*, pages 572–578. Morgan Kaufmann. 11

[Gamrath *et al.*, 2015] GAMRATH, G., KOCH, T., MARTIN, A., MILTENBERGER, M. et WENINGER, D. (2015). Progress in presolving for mixed integer programming. *Mathematical Programming Computation*, 7(4):367–398. 8

[Ganter et Wille, 1997a] GANTER, B. et WILLE, R. (1997a). Applied lattice theory : Formal concept analysis. *In In General Lattice Theory, G. Grätzer editor, Birkhäuser*. Preprints. 52

[Ganter et Wille, 1997b] GANTER, B. et WILLE, R. (1997b). *Formal Concept Analysis : Mathematical Foundations*. Springer-Verlag. 58

[García-López *et al.*, 2002] GARCÍA-LÓPEZ, F., MELIÁN-BATISTA, B., MORENO-PÉREZ, J. A. et MORENO-VEGA, J. M. (2002). The parallel variable neighborhood search for the p-median problem. *Journal of Heuristics*, 8(3):375–388. 40, 41

[Gaschnig, 1979] GASCHNIG, J. G. (1979). *Performance Measurement and Analysis of Certain Search Algorithms.* Thèse de doctorat, Pittsburgh, PA, USA. AAI7925014. 11

[Geerts *et al.*, 2004] GEERTS, F., GOETHALS, B. et MIELIKÄINEN, T. (2004). Tiling databases. *In DS'04, Padova, Italy, October 2-5, 2004, Proceedings*, pages 278–289. 56, 60, 61, 105, 107, 112

[Geng et Hamilton, 2006] GENG, L. et HAMILTON, H. J. (2006). Interestingness measures for data mining : A survey. *ACM Comput. Surv.*, 38(3). 48

[Ginsberg, 1993] GINSBERG, M. L. (1993). Dynamic backtracking. *J. Artif. Intell. Res. (JAIR)*, 1:25–46. 11

[Glover, 1996] GLOVER, F. (1996). Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics*, 65(1):223 – 253. 30

[Glover et Laguna, 1997] GLOVER, F. et LAGUNA, M. (1997). *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA. 30

[Goldfarb et Todd, 1989] GOLDFARB, D. et TODD, M. J. (1989). Optimization. chapitre Linear Programming, pages 73–170. Elsevier North-Holland, Inc., New York, NY, USA. 7

[Golomb et Baumert, 1965] GOLOMB, S. W. et BAUMERT, L. D. (1965). Backtrack programming. *J. ACM*, 12(4):516–524. 10

[Gomory, 1958] GOMORY, R. E. (1958). Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Society*, 64:275–278. 8

[Gottlob *et al.*, 2012] GOTTLOB, G., LEE, S. T., VALIANT, G. et VALIANT, P. (2012). Size and treewidth bounds for conjunctive queries. *J. ACM*, 59(3):16 :1–16 :35. xi, 76

[Griva *et al.*, 2009] GRIVA, I., NASH, S. et SOFER, A. (2009). *Linear and Nonlinear Optimization : Second Edition*. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104). 6, 7

[Guns *et al.*, 2011] GUNS, T., NIJSSEN, S. et RAEDT, L. D. (2011). Itemset mining : A constraint programming perspective. *Artif. Intell.*, 175(12-13):1951–1983. xii

[Guns *et al.*, 2013] GUNS, T., NIJSSEN, S. et RAEDT, L. D. (2013). k-pattern set mining under constraints. *IEEE Trans. Knowl. Data Eng.*, 25(2):402–418. xii, xv, 61, 112

[Hand, 2002] HAND, D. J. (2002). *Pattern Detection and Discovery*, pages 1–12. Springer Berlin Heidelberg, Berlin, Heidelberg. 46

[Hansen *et al.*, 2007] HANSEN, P., BRIMBERG, J., UROŠEVIĆ, D. et MLADENOVIĆ, N. (2007). Primal-dual variable neighborhood search for the simple plant-location problem. *INFORMS Journal on Computing*, 19(4):552–564. 35

[Hansen et Mladenović, 1999] HANSEN, P. et MLADENOVIĆ, N. (1999). *An Introduction to Variable Neighborhood Search*, pages 433–458. Springer US, Boston, MA. 33

[Hansen et Mladenovic, 2001] HANSEN, P. et MLADENOVIC, N. (2001). Variable neighborhood search : Principles and applications. *European Journal of Operational Research*, 130(3):449–467. 33

[Hansen et Mladenović, 2001] HANSEN, P. et MLADENOVIĆ, N. (2001). Variable neighbourhood decomposition search. *joh*, 7(4):335–350. 34, 35

[Hansen et Mladenović, 2006] HANSEN, P. et MLADENOVIĆ, N. (2006). First vs. best improvement : An empirical study. *Discrete Applied Mathematics*, 154(5):802 – 817. {IV} ALIO/EURO Workshop on Applied Combinatorial OptimizationIV ALIO/EURO Workshop on Applied Combinatorial Optimization. 33

[Haralick et Elliott, 1980] HARALICK, R. M. et ELLIOTT, G. L. (1980). Increasing tree search efficiency for constraint satisfaction problems. *Artif. Intell.*, 14(3):263–313. 11

[Harvey et Ginsberg, 1995] HARVEY, W. D. et GINSBERG, M. L. (1995). Limited discrepancy search. *In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*, pages 607–615. Morgan Kaufmann. 11

[Hirschhorn et Daly, 2005] HIRSCHHORN, J. N. et DALY, M. J. (2005). Genome-wide association studies for common diseases and complex traits. *Nature Reviews Genetics*, 6(2):95–108. 19, 68

[Hoffman et Padberg, 2009] HOFFMAN, K. et PADBERG, M. (2009). *Set covering, packing and partitioning problems Set Covering, Packing and Partitioning Problems*, pages 3482–3486. Springer US, Boston, MA. 108

[Hutter *et al.*, 2012] HUTTER, F., XU, L., HOOS, H. H. et LEYTON-BROWN, K. (2012). Algorithm runtime prediction : The state of the art. *CoRR*, abs/1211.0906. 75

[Jussien *et al.*, 2000] JUSSIEN, N., DEBRUYNE, R. et BOIZUMAULT, P. (2000). Maintaining arc-consistency within dynamic backtracking. *In* DECHTER, R., 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 249–261. Springer. 11

[Jégou et Terrioux, 2003] JÉGOU, P. et TERRIOUX, C. (2003). Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, 146(1):43 – 75. xi, 24, 76

[Karmarkar, 1984] KARMARKAR, N. (1984). A new polynomial-time algorithm for linear programming. *In Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, STOC '84, pages 302–311, New York, NY, USA. ACM. 7

[Kennedy et Eberhart, 1995] KENNEDY, J. et EBERHART, R. C. (1995). Particle swarm optimization. *In Proceedings of the IEEE International Conference on Neural Networks*, pages 1942–1948. 31

[Khachiyan, 1979] KHACHIYAN, L. G. (1979). A polynomial algorithm in linear programming. *Doklady Akademii Nauk SSSR*, 244:1093–1096. 7

[Kirkpatrick *et al.*, 1983] KIRKPATRICK, S., GELATT, C. D. et VECCHI, M. P. (1983). Optimization by simulated annealing. *SCIENCE*, 220(4598):671–680. 30

[Kitching et Bacchus, 2009] KITCHING, M. et BACCHUS, F. (2009). Exploiting decomposition on constraint problems with high tree-width. *In Proceedings of the 21st International Jont Conference on Artifical Intelligence*, IJCAI'09, pages 525–531, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc. xi

[Klee et Minty, 1972] KLEE, V. et MINTY, G. J. (1972). How good is the simplex algorithm ? *In* SHISHA, O., éditeur : *Inequalities*, volume III, pages 159–175. Academic Press, New York. 7

[Korf, 1996] KORF, R. E. (1996). Improved limited discrepancy search. *In* CLANCEY, W. J. et WELD, D. S., éditeurs : *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, August 4-8, 1996, Volume 1.*, pages 286–291. AAAI Press / The MIT Press. 12

[Kotthoff, 2012] KOTTHOFF, L. (2012). Algorithm selection for combinatorial search problems : A survey. *CoRR*, abs/1210.7959. 75

[Larrosa, 2002] LARROSA, J. (2002). Node and arc consistency in weighted csp. *In Eighteenth National Conference on Artificial Intelligence*, pages 48–53, Menlo Park, CA, USA. American Association for Artificial Intelligence. 13, 16, 17

[Larrosa et Schiex, 2003a] LARROSA, J. et SCHIEX, T. (2003a). In the quest of the best form of local consistency for weighted CSP. *In* GOTTLOB, G. et WALSH, T., éditeurs : *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 239–244. Morgan Kaufmann. 15

[Larrosa et Schiex, 2003b] LARROSA, J. et SCHIEX, T. (2003b). In the quest of the best form of local consistency for weighted CSP. *In* GOTTLOB, G. et WALSH, T., éditeurs : *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 239–244. Morgan Kaufmann. 16, 18

[Larrosa et Schiex, 2004] LARROSA, J. et SCHIEX, T. (2004). Solving weighted CSP by maintaining arc consistency. *Artif. Intell.*, 159(1-2):1–26. 15

[Lazić *et al.*, 2010] LAZIĆ, J., HANAFI, S., MLADENOVIĆ, N. et UROŠEVIĆ, D. (2010). Variable neighbourhood decomposition search for 0–1 mixed integer programs. *Computers & Operations Research*, 37(6):1055 – 1067. 35

[Lee et Leung, 2009] LEE, J. H. M. et LEUNG, K. L. (2009). Towards efficient consistency enforcement for global constraints in weighted constraint satisfaction. *In Proceedings of the 21st International Jont Conference on Artifical Intelligence*, IJCAI'09, pages 559–565, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc. 19

[Lin et Kedem, 1998] LIN, D.-I. et KEDEM, Z. M. (1998). Pincer search : A new algorithm for discovering the maximum frequent set. *In Proceedings of the 6th International Conference on Extending Database Technology : Advances in Database Technology*, EDBT '98, pages 105–119, London, UK, UK. Springer-Verlag. 51

[Lin et Kernighan, 1973] LIN, S. et KERNIGHAN, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Oper. Res.*, 21(2):498–516. 30

[Loudni et Boizumault, 2008] LOUDNI, S. et BOIZUMAULT, P. (2008). Combining VNS with constraint programming for solving anytime optimization problems. *European Journal of Operational Research*, 191:705–735. 37

[Loudni *et al.*, 2013] LOUDNI, S., FONTAINE, M. et BOIZUMAULT, P. (2013). Intensification/diversification in decomposition guided VNS. *In* BLESA, M. J., BLUM, C., FESTA, P., ROLI, A. et SAMPELS, M., éditeurs : *Hybrid Metaheuristics - 8th International Workshop, HM 2013, Ischia, Italy, May 23-25, 2013. Proceedings*, volume 7919 de *Lecture Notes in Computer Science*, pages 22–36. Springer. 38

[Mackworth, 1977] MACKWORTH, A. K. (1977). Consistency in networks of relations. *Artif. Intell.*, 8(1):99–118. 12

[Mannila et Toivonen, 1996] MANNILA, H. et TOIVONEN, H. (1996). Multiple uses of frequent sets and condensed representations (extended abstract). *In* SIMOUDIS, E., HAN, J. et FAYYAD, U. M., éditeurs : *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA*, pages 189–194. AAAI Press. 46, 47, 50, 51

[Marchand *et al.*, 2002] MARCHAND, H., MARTIN, A., WEISMANTEL, R. et WOLSEY, L. (2002). Cutting planes in integer and mixed integer programming. *Discrete Applied Mathematics*, 123(1–3):397 – 446. 8

[Marinescu et Dechter, 2009] MARINESCU, R. et DECHTER, R. (2009). And/or branch-and-bound search for combinatorial optimization in graphical models. *Artificial Intelligence*, 173(16):1457 – 1491. xi, 24, 25, 76

[Métivier *et al.*, 2012] MÉTIVIER, J.-P., BOIZUMAULT, P., CRÉMILLEUX, B., KHIARI, M. et LOUDNI, S. (2012). Constrained clustering using sat. *In IDA*, pages 207–218. xii, 61

[Michalski et Stepp, 1983] MICHALSKI, R. S. et STEPP, R. E. (1983). Learning from observation : Conceptual clustering. *In Machine Learning*, pages 331–363. Springer. 105

[Mitchell, 1998] MITCHELL, M. (1998). *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA. 31

[Mitchell, 1982] MITCHELL, T. M. (1982). Generalization as search. *Artificial Intelligence*, 18(2):203–226. 49, 50

[Mladenovic et Hansen, 1997] MLADENOVIC, N. et HANSEN, P. (1997). Variable neighborhood search. *Computers & OR*, 24(11):1097–1100. xi, 32, 76

[Mladenović *et al.*, 2005] MLADENOVIĆ, N., PLASTRIA, F. et UROŠEVIĆ, D. (2005). Reformulation descent applied to circle packing problems. *Computers & Operations Research*, 32(9):2419 – 2434. 35

[Mohr et Henderson, 1986] MOHR, R. et HENDERSON, T. C. (1986). Arc and path consistency revisited. *Artif. Intell.*, 28(2):225–233. 12

[Mueller et Kramer, 2010a] MUELLER, M. et KRAMER, S. (2010a). Integer linear programming models for constrained clustering. *In Discovery Science*, pages 159–173. xiii, 115, 118

[Mueller et Kramer, 2010b] MUELLER, M. et KRAMER, S. (2010b). Integer linear programming models for constrained clustering. *In DS 2010*, pages 159–173. 61

[Nemhauser et Wolsey, 1988] NEMHAUSER, G. L. et WOLSEY, L. A. (1988). *Integer and Combinatorial Optimization*. Wiley-Interscience, New York, NY, USA. 3

[Ouali *et al.*, 2016] OUALI, A., LOUDNI, S., LEBBAH, Y., BOIZUMAULT, P., ZIMMERMANN, A. et LOUKIL, L. (2016). Efficiently finding conceptual clustering models with integer linear programming. *In* KAMBHAMPATI, S., éditeur : *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 647–654. IJCAI/AAAI Press. xiv, 126

[Ouali *et al.*, 2014a] OUALI, A., LOUDNI, S., LOUKIL, L., BOIZUMAULT, P. et LEBBAH, Y. (2014a). *Cooperative Parallel Decomposition Guided VNS for Solving Weighted CSP*, pages 100–114. Springer International Publishing, Cham. xiii, 76, 78, 126

[Ouali *et al.*, 2015] OUALI, A., LOUDNI, S., LOUKIL, L., BOIZUMAULT, P. et LEBBAH, Y. (2015). Replicated parallel strategies for decomposition guided vns. *Electronic Notes in Discrete Mathematics*, 47:93 – 100. xiii, 76, 78, 126

[Ouali *et al.*, 2014b] OUALI, A., LOUDNI, S., LOUKIL, L. et LEBBAH, Y. (2014b). Une approche parallèle coopérative exploitant la décomposition arborescente dans VNS. *In 11ème Colloque sur l'Optimisation et les Systèmes d'Information et les Systèmes d'Information (COSI 2014)*, pages P1–12, Béjaia, Algeria. xiii, 76, 126

[Ouali *et al.*, 2017] OUALI, A., ZIMMERMANN, A., LOUDNI, S., LEBBAH, Y., CREMILLEUX, B., BOIZUMAULT, P. et LOUKIL, L. (2017). *Integer Linear Programming for Pattern Set Mining ; with an Application to Tiling*, pages 286–299. Springer International Publishing, Cham. xiv, 127

[Papadimitriou et Steiglitz, 1982] PAPADIMITRIOU, C. H. et STEIGLITZ, K. (1982). *Combinatorial Optimization : Algorithms and Complexity*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA. 3

[Pardo *et al.*, 2013] PARDO, E. G., MLADENOVIĆ, N., PANTRIGO, J. J. et DUARTE, A. (2013). Variable formulation search for the cutwidth minimization problem. *Applied Soft Computing*, 13(5):2242 – 2252. 36

[Pasquier *et al.*, 1999] PASQUIER, N., BASTIDE, Y., TAOUIL, R. et LAKHAL, L. (1999). Discovering frequent closed itemsets for association rules. *In* BEERI, C. et BUNEMAN, P., éditeurs : *Database Theory - ICDT '99, 7th International Conference, Jerusalem, Israel, January 10-12, 1999, Proceedings.*, volume 1540 de *Lecture Notes in Computer Science*, pages 398–416. Springer. 51

[Pei *et al.*, 2000] PEI, J., HAN, J. et MAO, R. (2000). Closet : An efficient algorithm for mining frequent closed itemsets. *In ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 21–30. 51

[Pensa *et al.*, 2005] PENSA, R. G., ROBARDET, C. et BOULICAUT, J. (2005). A bi-clustering framework for categorical data. *In PKDD 2005*, pages 643–650. 60, 105

[Perkowitz et Etzioni, 1999] PERKOWITZ, M. et ETZIONI, O. (1999). Adaptive web sites : Conceptual cluster mining. *In IJCAI 99*, pages 264–269. 61, 105

[Prosser, 1993] PROSSER, P. (1993). Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9:268–299. 11

[Raedt et Zimmermann, 2007] RAEDT, L. D. et ZIMMERMANN, A. (2007). Constraint-based pattern set mining. *In Proceedings of the Seventh SIAM International Conference on Data Mining, April 26-28, 2007, Minneapolis, Minnesota, USA*, pages 237–248. SIAM. xii, 55, 56, 57, 58, 61, 62, 92, 98, 99, 103, 106

[Refalo, 2004] REFALO, P. (2004). Impact-based search strategies for constraint programming. *In* WALLACE, M., éditeur : *Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings*, volume 3258 de *Lecture Notes in Computer Science*, pages 557–571. Springer. 11

[Rish et Dechter, 2000] RISH, I. et DECHTER, R. (2000). Resolution versus search : Two strategies for sat. *Journal of Automated Reasoning*, 24(1):225–275. xi, 76

[Robertson et Seymour, 1986] ROBERTSON, N. et SEYMOUR, P. D. (1986). Graph minors. II. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322. xi, 19, 20, 76

[Rosenfeld *et al.*, 1976] ROSENFELD, A., HUMMEL, R. et ZUCKER, S. (1976). Scene Labeling by Relaxation Operations. *IEEE Trans. on Systems, Man, and Cybernetics*, 6(6):173–184. 13

[Rossi *et al.*, 2006] ROSSI, F., BEEK, P. v. et WALSH, T. (2006). *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA. 9

[Ruttkay, 1994] RUTTKAY, Z. (1994). Fuzzy constraint satisfaction. *In In Proc. 3rd IEEE International Conference on Fuzzy Systems*, pages 1263–1268. 13

[Sabin et Freuder, 1994] SABIN, D. et FREUDER, E. C. (1994). Contradicting conventional wisdom in constraint satisfaction. *In ECAI*, pages 125–129. 11

[Sánchez *et al.*, 2009] SÁNCHEZ, M., ALLOUCHE, D., de GIVRY, S. et SCHIEX, T. (2009). Russian doll search with tree decomposition. *In* BOUTILIER, C., éditeur : *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 603–608. xi, 76

[Sánchez *et al.*, 2008] SÁNCHEZ, M., de GIVRY, S. et SCHIEX, T. (2008). Mendelian error detection in complex pedigrees using weighted constraint satisfaction techniques. *Constraints*, 13(1-2):130–154. 19

[Schiex, 2000] SCHIEX, T. (2000). Arc consistency for soft constraints. *In* DECHTER, R., 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 411–424. Springer. 16, 18

[Schiex, 2013] SCHIEX, T. (2013). Possibilistic constraint satisfaction problems or "how to handle soft constraints ?". *CoRR*, abs/1303.5427. 13

[Schiex *et al.*, 1995] SCHIEX, T., FARGIER, H. et VERFAILLIE, G. (1995). Valued constraint satisfaction problems : Hard and easy problems. *In Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'95, pages 631–637, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc. 13

[Schiex et Verfaillie, 1994] SCHIEX, T. et VERFAILLIE, G. (1994). Nogood recording for static and dynamic constraint satisfaction problems. *International Journal on Artificial Intelligence Tools*, 3(2):187–208. 11

[Sevkli et Sevilgen, 2008] SEVKLI, Z. et SEVILGEN, F. E. (2008). A hybrid particle swarm optimization algorithm for function optimization. *In Proceedings of the 2008 Conference on Applications of Evolutionary Computing*, Evo'08, pages 585–595, Berlin, Heidelberg. Springer-Verlag. 33

[Shapiro et Haralick, 1981] SHAPIRO, L. G. et HARALICK, R. M. (1981). Structural descriptions and inexact matching. *IEEE Trans. Pattern Anal. Mach. Intell.*, 3(5):504–519. 13

[Shaw, 1998] SHAW, P. (1998). *Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems*, pages 417–431. Springer Berlin Heidelberg, Berlin, Heidelberg. 36, 37

[Shima *et al.*, 2005] SHIMA, Y., HIRATA, K. et HARAO, M. (2005). *Extraction of Frequent Few-Overlapped Monotone DNF Formulas with Depth-First Pruning*, pages 50–60. Springer Berlin Heidelberg, Berlin, Heidelberg. 98

[Smith et Grant, 1998] SMITH, B. M. et GRANT, S. A. (1998). Trying harder to fail first. *In ECAI*, pages 249–253. 11

[Stumme *et al.*, 2000] STUMME, G., TAOUIL, R., BASTIDE, Y., PASQUIER, N. et LAKHAL, L. (2000). Fast computation of concept lattices using data mining techniques. *In* BOUZEGHOUB, M., KLUSCH, M., NUTT, W. et SATTLER, U., éditeurs : *Proceedings of the 7th International Workshop on Knowledge Representation meets Databases (KRDB 2000), Berlin, Germany, August 21, 2000*, volume 29 de *CEUR Workshop Proceedings*, pages 129–139. CEUR-WS.org. 51

[Talbi, 2009] TALBI, E. (2009). *Metaheuristics - From Design to Implementation*. Wiley. xi, 29, 30

[Talbi, 2013] TALBI, E., éditeur (2013). *Hybrid Metaheuristics*, volume 434 de *Studies in Computational Intelligence*. Springer. 32

[Uno *et al.*, 2003] UNO, T., ASAI, T., UCHIDA, Y. et ARIMURA, H. (2003). LCM : an efficient algorithm for enumerating frequent closed item sets. *In* GOETHALS, B. et ZAKI, M. J., éditeurs : *FIMI '03, Frequent Itemset Mining Implementations, Proceedings of the ICDM 2003 Workshop on Frequent Itemset Mining Implementations, 19 December 2003, Melbourne, Florida, USA*, volume 90 de *CEUR Workshop Proceedings*. CEUR-WS.org. 54

[Uno *et al.*, 2004] UNO, T., KIYOMI, M. et ARIMURA, H. (2004). LCM ver. 2 : Efficient mining algorithms for frequent/closed/maximal itemsets. *In* JR., R. J. B., GOETHALS, B. et H, M. J., éditeurs : *FIMI '04, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, Brighton, UK, November 1, 2004*, volume 126 de *CEUR Workshop Proceedings*. CEUR-WS.org. 53, 54

[van Benthem, 1995] van BENTHEM, H. (1995). GRAPH : Generating radiolink frequency assignment problems heuristically. 67

[Wolsey, 1998] WOLSEY, L. A. (1998). *Integer Programming*. Wiley-Interscience. 8

[Xiao *et al.*, 2011] XIAO, Y., KAKU, I., ZHAO, Q. et ZHANG, R. (2011). A reduced variable neighborhood search algorithm for uncapacitated multilevel lot-sizing problems. *European Journal of Operational Research*, 214(2): 223 – 231. 33

[Yannakakis, 1981] YANNAKAKIS, M. (1981). Computing the minimum fill-in is np-complete. *SIAM Journal on Algebraic Discrete Methods*, 2(1):77–79. 21

[Zaki et Hsiao, 2002] ZAKI, M. J. et HSIAO, C. (2002). CHARM : an efficient algorithm for closed itemset mining. *In* GROSSMAN, R. L., HAN, J., KUMAR, V., MANNILA, H. et MOTWANI, R., éditeurs : *Proceedings of the Second SIAM International Conference on Data Mining, Arlington, VA, USA, April 11-13, 2002*, pages 457–473. SIAM. 51, 54

[Zaki *et al.*, 1997] ZAKI, M. J., PARTHASARATHY, S., OGIHARA, M. et LI, W. (1997). New algorithms for fast discovery of association rules. *In* HECKERMAN, D., MANNILA, H. et PREGIBON, D., éditeurs : *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD-97), Newport Beach, California, USA, August 14-17, 1997*, pages 283–286. AAAI Press. 51

[Zimmermann, 2009] ZIMMERMANN, A. (2009). *Mining Sets of Patterns (Zoeken naar verzamelingen van patronen)*. Thèse de doctorat, Katholieke Universiteit Leuven, Belgium. 55