Université des Sciences et Technologies de Lille

Institut National de Recherche en Informatique et Automatique (INRIA)

Thèse

présentée par

Ali Khanafer

en vu d'obtenir le grade de Docteur

spécialité : informatique

Algorithmes pour des problèmes de bin packing mono- et multi-objectif

Aziz Moukrim	Université de Technologie de Compiègne	(Rapporteur)
François Vanderbeck	Université de Bordeaux I	(Rapporteur)
Saïd Hanafi	Université de Valenciennes	(Examinateur)
Cláudio Alves	Universidade do Minho	(Examinateur)
El-Ghazali Talbi	Université de Lille I	(Directeur de thèse)
François Clautiaux	Université de Lille I	(Encadrant)

Numéro d'ordre 40363 | Année 2010

Table des matières

Liste des figures

Liste des tableaux

Introduction générale

Cette thèse s'inscrit dans le domaine de l'optimisation combinatoire mono- et multi-objectif. Elle a été réalisée au sein de l'équipe **DOLPHIN** de l'**Institut National de Recherche en Informatique et Automatique de Lille (INRIA Lille)**.

Le problème de bin packing consiste à déterminer le nombre minimum de conteneurs (bins) nécessaires pour ranger un ensemble d'objets. Ce problème $\mathcal{NP}$-complet fait depuis de nombreuses années l'objet de multiples travaux de recherche, théoriques et pratiques. On le retrouve entre autres dans l'industrie de découpe de tissu, de l'acier, de bois et de verre. D'autres applications peuvent être aussi citées comme par exemple l'optimisation du placement de spots publicitaires dans un journal, ou même l'allocation de tâches à un processeur.

La littérature sur le problème de bin packing est riche et les algorithmes et approches de résolution sont très diverses. Cependant, les solutions proposées par ces algorithmes peuvent ne pas être utiles quand on traite des problèmes industriels réels. Ce défaut provient du fait que ces algorithmes ne prennent pas en compte les contraintes supplémentaires imposées par les applications. Dans ce document, nous considérons plusieurs types de contraintes liées à des incompatibilités entre objets. Ces contraintes sont inspirées de celles rencontrées lors d'une collaboration industrielle.

Considérons un problème de packing où certains objets ne peuvent pas être placés dans le même bin. Les solutions obtenues avec cette contrainte peuvent nécessiter un très grand nombre de bins. Dans ce cas, le décideur peut être amené à limiter ce nombre à un certain seuil en relâchant en partie les contraintes de compatibilité. Répondre à cette problématique nous met face à un problème d'optimisation multi-objectif dans lequel on cherche à optimiser à la fois le nombre de bins et le nombre de conflits violés.

Une autre manière de représenter des incompatibilités entre objets est de considérer le cas où ils possèdent des fragilités qui ne doivent pas être dépassées par le poids total des articles au sein d'un bin. Les applications de cette version se trouvent notamment dans le domaine des télécommunications mobiles.

Le sujet de recherche de cette thèse porte sur la résolution d'une variété de problèmes de bin packing. Nous nous intéressons à des bornes inférieures et supérieures pour les trois problèmes suivants : **un problème de bin packing avec conflits** dans lequel des relations de compatibilité sont exprimées entre les couples d'objets ; **un problème de bin packing bi-objectif** dans lequel deux critères sont à minimiser, le

nombre de bins utilisés et le nombre de couples en conflit placés dans le même bin ; **un problème de bin packing avec objets fragiles** dans lequel la somme des tailles des objets placés dans un bin ne dépasse la fragilité d'aucun de ces objets.

Dans cette thèse, nous traitons de problèmes difficiles que nous cherchons à résoudre en pratique. Pour cette raison, nous nous sommes concentrés sur deux aspects : la décomposition en sous-problèmes, et les méthodes approchées (heuristiques et bornes inférieures). Nous avons utilisé deux types de méthodes de décomposition : la *décomposition arborescente* de graphes et celle de programmes linéaires par le biais de la décomposition de Dantzig et Wolfe. Pour ce qui est des heuristiques, nous avons focalisé notre attention sur des méthodes de recherche locale reposant sur le concept *d'oscillation stratégique*, dans laquelle on alterne des phases de construction d'une solution et celles de destruction. Enfin, pour le calcul des bornes inférieures, nos travaux ont reposés sur l'étude de fonctions dites *dual-réalisables* et de leurs extensions, ainsi que sur la relaxation linéaire de programmes en nombres entiers.

Dans le chapitre 1, nous commençons par rappeler le cadre de l'optimisation combinatoire mono- et multi-objectif. Nous décrivons ensuite le problème de bin packing classique, ainsi que les variantes qui seront évoquées dans ce manuscrit. Nous exposons brièvement les modèles et les méthodes dédiés à ces problèmes. Nous présentons de manière détaillée les méthodes heuristiques, ainsi que les prétraitements qui servent à réduire la taille des instances. Nous développons particulièrement la partie concernant les bornes inférieures de la littérature, notamment les travaux basés sur le concept de fonctions dual-réalisables qui nous sera utile dans la suite du manuscrit. Nos problèmes étant étroitement liés aux *problèmes de coloration*, ces derniers sont aussi décrits ainsi que le concept de décomposition arborescente.

Dans le chapitre 2, nous étudions de nouvelles bornes inférieures pour le bin-packing avec conflits. Elles reposent sur le concept de *fonctions dual-réalisables dépendantes de la donnée généralisées* (GDDFF), qui est une généralisation du concept de DDFF proposé par Carlier *et al.* [12] pour prendre en compte les relations de compatibilité entre les objets. Nous proposons une nouvelle famille de DDFF, ainsi que deux familles de GDDFF. Nous proposons aussi des prétraitements qui réduisent la taille du problème. Ils reposent sur la détection des objets en conflit avec tous les autres objets ou de sous-ensembles d'objets pour lesquels une solution optimale peut être calculée. Nos méthodes sont validées numériquement sur les instances de référence [2] et améliorent les résultats pour plusieurs instances pour le cas uni- et bi-dimensionnel.

L'apport du chapitre 3 est une nouvelle approche de résolution pour le problème de bin packing avec conflits basée sur une décomposition arborescente du graphe lié à l'instance. L'idée consiste à diviser le problème en un ensemble de sous-problèmes à traiter indépendamment. Cette méthode permet de prendre en compte la structure du graphe et améliore la performance des algorithmes dédiés au packing. Utiliser la décomposition arborescente n'est pas direct, car certains objets peuvent apparaître dans plusieurs sous-ensembles. Notre approche nécessite donc un soin particulier dans l'affectation des ces objets à un unique cluster (on parle de *séparation de cluster*). Nous proposons pour ce problème des heuristiques rapides, ainsi qu'une méthode

de recherche tabou basée sur une stratégie d'oscillation. Nous concluons le chapitre par des résultats théoriques qui montrent l'efficacité de l'approche en fonction de la largeur de la décomposition arborescente obtenue. Nos méthodes ont été validées expérimentalement avec succès sur des instances issues de la littérature.

Le chapitre 4 est consacré à la variante bi-objectif du problème de bin packing avec conflits, dans laquelle on cherche à minimiser le nombre de bins utilisés et le nombre de couples d'objets en conflit placés dans un même bin. Nous proposons des formulations mathématiques du problème, ainsi que des bornes inférieures et supérieures basées sur la structure spéciale de la frontière pareto. Nous proposons aussi une méthode basée sur la décomposition de Dantzig et Wolfe et la génération de colonnes. Nous avons en particulier proposé plusieurs méthodes heuristiques et exactes pour générer les colonnes efficacement. Une méthode de recherche tabou basée sur une stratégie de recherche oscillatoire est aussi développée. L'efficacité des algorithmes proposés est démontrée par des expérimentations numériques menées sur un jeu d'essai issu de la littérature.

Dans le chapitre 5, nous abordons le problème de bin packing avec objets fragiles. A notre connaissance, cette thèse présente le premier travail qui traite de ce problème d'un point de vue pratique en fournissant des résultats numériques. Nous proposons ainsi un ensemble de bornes inférieures polynomiales basées sur les fonctions dual-réalisables et une méthode de recherche locale à voisinage variable, qui permet d'implémenter une stratégie d'oscillation. Nous proposons aussi une méthode de génération de colonnes pour résoudre ce problème. Pour cette dernière méthode, nous avons étudié trois variantes pour résoudre le problème de pricing associé : deux reposant sur la programmation mathématique et une sur la programmation dynamique. Afin de tester l'efficacité de ces algorithmes, nous avons mis au point un jeu d'essai comportant des instances difficiles. Une analyse fine des résultats est rapportée.

Nous concluons la thèse par un bilan des travaux effectués, ainsi que par les perspectives de recherche liées à cette thèse.

Chapitre 1

Etat de l'art

1.1 Introduction

Le problème de bin packing consiste à déterminer le nombre minimum de conteneurs (*bins*) identiques nécessaires pour ranger un ensemble d'objets. Il a fait l'objet depuis quelques années d'une attention croissante pour deux raisons : outre son intérêt théorique, il a de nombreuses applications dans le domaine industriel, de la logistique, de l'informatique et même de l'édition. On retrouve le problème notamment dans l'industrie du tissu, du métal, mais aussi dans le cadre de la découpe de bois, de verre ou même de placement des spots publicitaires dans les journaux.

Ce problème $\mathcal{NP}$-difficile [51] a été étudié par un grand nombre de chercheurs et est très bien traité dans le cas général par des heuristiques avec garantie de performance et des évaluations par défaut très efficaces. L'intérêt pour d'autres variantes de ce problème est plus récent car elles se prêtent moins bien à des études théoriques. Néanmoins, depuis quelques années, un nombre croissant d'articles traitent de ce sujet et de ses variantes. Dans cette thèse, nous traitons les trois problèmes de bin packing suivants : **un problème de bin packing avec conflits**, **un problème de bin packing bi-objectif** et **un problème de bin packing avec objets fragiles**. La majeure partie de nos contributions concernent des méthodes heuristiques, des prétraitements et des bornes inférieures.

Dans ce chapitre, nous présentons les résultats les plus récents pour les problèmes de bin packing traités dans cette thèse, en nous concentrant sur les heuristiques et les bornes inférieures. Nous présentons aussi des modèles mathématiques pour ces problèmes, qui vont de formulations linéaires compactes à des modèles décomposés selon **la méthode de Dantzig-Wolfe**.

Concernant les **bornes inférieures**, deux types d'approche sont possibles pour ces problèmes : les bornes polynomiales souvent basées sur des calculs rapides en fonctions des tailles des objets (*e.g.* fonctions dual-réalisables (DFF) et leurs extensions), et les bornes basées sur des relaxations de modèles linéaires.

Les **heuristiques** de la littérature sont en général de type glouton et basées sur des procédures basiques. Il existe aussi des procédures de recherche locale.

Les problèmes de packing avec conflits généralisent les problèmes de **coloration de graphe**. Un rapide survol des problèmes de coloration et des différentes approches qui leur sont dédiées sera effectué dans ce chapitre. On peut citer en particulier le

concept de **décomposition arborescente**, qui est un outil intéressant pour résoudre plusieurs problèmes d'optimisation basés sur des graphes. Nous présentons donc dans ce chapitre une méthode de décomposition arborescente qui permet d'exploiter la structure d'un graphe afin d'en tirer de l'information et d'accélérer le processus de résolution. Quelques définitions préliminaires sont alors données afin de mieux comprendre le problème de coloration, de calcul de clique maximale et du calcul d'une décomposition arborescente.

Le reste de ce chapitre est organisé comme suit. Des définitions pour l'optimisation combinatoire mono- et multi-objectif sont données dans la section 1.2. Dans la section 1.3, nous rappelons quelques notions de théorie des graphes dont nous aurons besoin dans la suite de ce document, ainsi que le problème de coloration et ses différentes approches de résolution. Dans les sections 1.4, 1.5 et 1.6 nous présentons respectivement le problème de bin packing, le problème de bin packing avec conflits et le problème de bin packing avec objets fragiles, et nous passons en revue des méthodes de résolution dédiées à ces problèmes.

1.2 Optimisation combinatoire

Dans cette section, nous rappelons quelques définitions qui portent sur l'optimisation mono- et multi-objectif. Nous décrivons aussi rapidement les différentes approches de résolution des problèmes multi-objectif trouvées dans la littérature.

1.2.1 Optimisation mono-objectif

Résoudre un problème d'optimisation (PO) consiste à trouver une solution qui optimise (*minimise* ou *maximise*) un critère donné. L'*optimum* découvert n'est pas souvent unique, mais au contraire, il existe un ensemble de solutions optimisant ce critère. Formellement, un problème d'optimisation peut être décrit comme suit pour $n \geq 1$:

$$PO = \begin{cases} \text{Optimiser une fonction objectif } f(\overrightarrow{x}) : \mathbb{R}^n \rightarrow \mathbb{R} \\ \text{où } \overrightarrow{x} \in \mathcal{X} \text{ est un vecteur comprenant les variables de décision} \\ \text{et } \mathcal{X} \text{ est un ensemble fermé de } \mathbb{R}^n \\ \text{et } \mathcal{Z} = f(\mathcal{X}) \text{ un ensemble de points réalisables} \end{cases}$$

L'ensemble $\mathcal{X}$, dit *espace décisionnel*, représente l'ensemble des solutions potentielles du problème. Cet ensemble est déterminé à l'aide de contraintes, souvent analytiques, données dans l'énoncé du problème. Le terme *espace de recherche* représente l'ensemble des valeurs pouvant être prises par les variables de décision, et la notion d'*espace réalisable* représente l'ensemble des valeurs des variables satisfaisant les contraintes.

Dans le cadre de cette thèse, nous traitons des problèmes appartenant à la classe des problèmes $\mathcal{NP}$-difficiles.

1.2.2 Optimisation multi-objectif

Dans un contexte d'optimisation multi-objectif, le décideur fait face à un problème d'évaluation de solutions par rapport à un ensemble de critères considérés. Il n'existe pas alors un seul optimum global pour tous les critères. Il faut donc utiliser une nouvelle notion d'optimum. La notion d'optimalité Pareto est la plus commune dans la littérature [98, 37].

Définition 1.1 ***(Solution dominée)*** *Une solution x domine une solution y (notée $x \succeq y$) ssi $\forall i \in \{1, 2, \ldots, n\}$, $f_i(x)$ est meilleure que $f_i(y)$ et $\exists i \in \{1, 2, \ldots, n\}$, $f_i(x)$ est strictement meilleure que $f_i(y)$.*

Définition 1.2 ***(Solution non-dominée)*** *Une solution x est dite solution non-dominée ssi $\forall y \in \mathcal{X}, f(y) \not\succ f(x)$.*

Définition 1.3 ***(Solution Pareto optimale)*** *une solution $x \in \mathcal{X}$ est dite Pareto optimale ssi x est une solution non-dominée.*

Une solution est donc dite Pareto optimale s'il n'existe pas d'autre solution réalisable qui améliore la valeur d'un objectif sans dégrader la valeur d'au moins un autre objectif. Les solution Pareto optimales sont aussi connues sous le nom de solutions efficaces. La solution d'un problème multi-objectif n'est pas donc une solution unique mais un ensemble de solutions dit "ensemble Pareto optimal".

Définition 1.4 ***(Ensemble Pareto optimal)*** *Etant donné un problème d'optimisation multi-objectif, l'ensemble Pareto optimal $\mathcal{X}^*$ est défini comme suit :*

$$\mathcal{X}^* = \{x \in \mathcal{X} \mid \nexists x' \in \mathcal{X}, f(x') \succeq f(x)\} \tag{1.1}$$

Définition 1.5 ***(Frontière Pareto)*** *Etant donné un problème d'optimisation multi-objectif et son ensemble Pareto optimal $\mathcal{X}^*$, la frontière Pareto $\mathcal{Z}^*$ est définie comme suit :*

$$\mathcal{Z}^* = \{f(x) \mid x \in \mathcal{X}^*\} \tag{1.2}$$

Il est difficile, voire impossible, de déterminer une fonction analytique permettant de trouver les coordonnées des solutions Pareto optimales dans l'espace de recherche. La procédure classique consiste à approximer la frontière Pareto en cherchant un nombre suffisant de solutions x et en calculant leur coût $f(x)$ dans l'espace objectif tout en tenant compte des règles de dominance définies plus-haut.

Un grand nombre d'approches existent pour résoudre les problèmes d'optimisation multi-objectif. Certaines utilisent des connaissances du problème pour fixer des préférences sur les critères et ainsi contourner son aspect multicritère. D'autres mettent tous les critères au même niveau d'importance, mais là aussi il existe plusieurs

façons de réaliser une telle opération. Plusieurs ouvrages ou articles de synthèse ont été rédigés, des états de l'art plus complets peuvent être consultés notamment dans [27, 32, 38, 39, 90, 112].

Dans sa thèse, Hammami [58] a dressé un panorama des méthodes d'optimisation multi-objectif. celles-ci sont réparties en trois catégories :

- les méthodes scalaires ;
- les méthodes non-scalaires ;
- les méthodes Pareto.

Nous allons préciser rapidement les spécificités de ces méthodes. Dans une approche scalaire, une fonction objectif est créée de manière à traiter le problème d'optimisation multi-objectif comme un problème d'optimisation mono-objectif, et donc d'utiliser les méthodes déjà existantes pour de tels problèmes. Plusieurs méthodes différentes ont été proposées pour transformer un problème multi-objectif en un problème à un seul objectif : les méthodes d'agrégation, les méthodes avec vecteur cible et les méthodes ϵ-contraintes notamment.

Dans une approche non-scalaire, les méthodes d'optimisation se basent essentiellement sur des recherches à base de population de solutions qui traitent séparément les différents objectifs. L'inconvénient de ces méthodes réside dans le fait qu'elles tendent à générer des solutions largement optimisées pour certains objectifs et beaucoup moins optimisées pour d'autres. Les solutions de compromis peuvent être négligées.

Les approches Pareto utilisent directement la notion de dominance Pareto. D'une manière générale, la plupart des méthodes trouvées dans la littérature appartiennent à la famille des algorithmes évolutionnaires, et la majeure partie de ces méthodes évolutionnaires sont des algorithmes génétiques. Néanmoins, des méthodes non-évolutionnaires ont aussi été proposées. D'une manière générale, les approches Pareto permettent de ne pas favoriser un objectif plutôt qu'un autre. Elles traitent les objectifs d'une façon équitable fournissant ainsi des solutions de meilleur compromis au décideur.

1.3 Graphe, coloration et décomposition

Dans cette section, nous passons en revue l'ensemble des notions de théorie des graphes dont nous aurons besoin dans la suite de ce document, en mettant l'accent sur le problème de coloration de graphes. Nous présentons ensuite la méthode de *décomposition arborescente* qui permet d'exploiter la structure d'un graphe afin d'en déduire de l'information et d'accélérer un processus de résolution. Nous décrivons en particulier comment construire une telle décomposition.

1.3.1 Définitions élémentaires

Etant donné un graphe non orienté $G = (V, E)$, V désignera l'ensemble de ses *sommets* et $E \subseteq V \times V$ l'ensemble de ses *arêtes*. Un graphe $G = (V, E)$ est dit *complet*

si, pour toute paire de sommets u, v de V, $(u, v) \in E$. Un graphe $G = (V, E)$ est dit *biparti* ssi il existe un ensemble $U \subseteq V$ tel que, pour toute arête $(u, v) \in E$, $u \in U$ et $v \in V \setminus U$; G est *biparti complet* si, pour tout $u \in U$ et pour tout $v \in V \setminus U$, $(u, v) \in E$. Etant donné un graphe non orienté $G = (V, E)$, le *degré* de $v \in V$, noté $deg(v)$, est le nombre de sommets auxquels v est connecté par une arête.

1.3.2 Coloration de graphes

La coloration de graphes est un problème très ancien. Son existence remonte à 1852 avec le besoin de coloration des cartes géographiques. A cette époque, le problème consistait à colorer les cartes en affectant des couleurs différentes aux zones géographiques adjacentes. Aujourd'hui, le problème de coloration est d'un intérêt double, pratique (établissement d'emploi de temps, ordonnancement de tâches, allocation de ressources [78, 48, 31, 50]) et théorique (problème de référence de la classe $\mathcal{NP}$-complet).

FIGURE 1.1 – *Un exemple de graphe et une coloration possible de ce graphe.*

Ce problème a été l'objet de nombreux travaux qui se répartissent globalement selon deux approches : premièrement la recherche de classes de graphes pour lesquels le problème est polynomial [48, 30, 52, 19], et deuxièmement la recherche d'algorithmes heuristiques, capables de trouver dans le cas général, en un temps raisonnable, des colorations approchées de bonne qualité. On retrouve notamment des méthodes constructives [9, 78] et les méthodes de recherche locale [61, 14, 36, 47].

Dans cette section, nous présentons plus formellement le problème de coloration et nous décrivons aussi les algorithmes qui nous seront utiles dans la suite de ce manuscrit.

1.3.2.1 Définitions

Quelques notions concernant le problème de coloration de graphes sont rappelées ici.

Une *coloration d'un graphe* $G = (V, E)$ est l'affectation d'une couleur $c(x)$ à chaque sommet $x \in V$ en s'assurant que $c(x) \neq c(y)$ pour toute arête $(x, y) \in E$. Elle peut également être vue comme une partition de V en k blocs, de telle sorte qu'il n'existe pas d'arête entre deux sommets d'un même bloc. Si le nombre de couleurs utilisées est k, la coloration de G est appelée une k-coloration.

Définition 1.6 ***(Problème de k-coloration)*** *Le problème de k-coloration consiste, étant donné un graphe G et un entier k, à répondre à la question suivante : "G est-il colorable avec seulement k couleurs ?"*

Définition 1.7 ***(Problème de coloration)*** *Le problème de coloration de graphes est le problème d'optimisation associé au problème de k-coloration. Il consiste à trouver la valeur minimale de k pour laquelle une k-coloration est possible. Cette valeur est appelée nombre chromatique du graphe et notée $\chi(G)$.*

Dans le cas général, trouver le nombre chromatique d'un graphe est un problème $\mathcal{NP}$-difficile, et il est tout aussi difficile d'estimer ce nombre [51]. Néanmoins, pour certaines familles particulières de graphes le nombre chromatique peut être calculé en temps polynomial. Parmi ces graphes, citons les graphes planaires et les graphes parfaits y-compris les graphes d'intervalles et les graphes triangulés (voir [89] pour plus de détails).

1.3.2.2 Approches de résolution

Un grand nombre de méthodes ont été proposées pour le problème de coloration. Les principales méthodes exactes reposent sur des méthodes de construction arborescente de type branch & bound (voir par exemple [9]). Dans la pratique, ces algorithmes rencontrent des difficultés pour colorer certains graphes à partir d'une taille moyenne. Pour cette raison, des méthodes heuristiques ont été proposées dans le but de trouver en un temps raisonnable des colorations de bonne qualité.

Les premières heuristiques proposées sont les heuristiques gloutonnes [9, 78]. Depuis la fin des années 80, un grand pas a été franchi avec l'introduction des méthodes de recherche locale [14, 61, 36, 95, 35, 47, 67].

Il existe notamment deux approches de nature opposée pour résoudre le problème de coloration. La première est *l'approche de construction* qui consiste, à partir d'une coloration partielle vide, à colorer successivement les différents sommets pour tenter d'aboutir à une coloration complète si possible optimale. La deuxième approche est l'approche de réparation. Celle-ci consiste à construire une coloration complète éventuellement mauvaise, puis à itérer une suite de transformations locales pour aboutir à une coloration de meilleure qualité. L'approche de construction est celle qu'adoptent les méthodes exactes de type branch & bound ainsi que les heuristiques gloutonnes. Certaines méthodes de recherche locale se fondent sur l'approche de réparation, mais certaines autres utilisent l'approche de construction.

1.3.2.2.1 Heuristiques gloutonnes

Les heuristiques gloutonnes sont des méthodes constructives sans retour arrière. Le principe d'une heuristique gloutonne est de colorer tour à tour chacun des sommets du graphe, sans jamais mettre en cause les choix effectués. A chaque étape, l'algorithme choisit un sommet non coloré et une couleur pour colorer ce sommet. Ces deux choix sont déterminés par des critères heuristiques.

On peut distinguer les heuristiques gloutonnes statiques et dynamiques. Les premières commencent par déterminer un ordre sur les sommets du graphe (permutation) et dans un deuxième temps, considèrent les sommets dans cet ordre et attribuent une couleur à chaque sommet. Dans une heuristique dynamique, le nouveau sommet à colorer est choisi dynamiquement, c'est-à-dire lorsque les sommets précédents ont déjà été colorés.

D'une manière générale, les heuristiques gloutonnes sont très rapides mais les résultats peuvent être de mauvaise qualité. Les deux algorithmes gloutons les plus efficaces et les plus connus sont DSatur proposé par Brélaz [9] en 1979, et RLF proposé par Leighton [78] la même année. La méthode DSatur est détaillée dans le paragraphe suivant car elle sera utilisée dans la suite de ce manuscrit.

L'heuristique de saturation [9] est une heuristique gloutonne dynamique. L'algorithme 1 décrit étape par étape le déroulement de l'heuristique DSatur.

Algorithme 1 : Algorithme DSatur

entrée : un graphe $G = (V, E)$
sortie : une couleur $c(i)$ pour tout $i \in V$

```
1 pour i ∈ V faire
2     c(i) ← 0;
3     dsat(i) ← deg(i);
4 tant que il reste un sommet non coloré faire
5     i = sommet non coloré ayant le plus grand dsat;
6     c(i) = plus petite couleur non utilisée par un voisin de i;
7     pour chaque voisin x de i faire
8         dsat(x) ← nombre de voisins colorés de x;
```

A chaque étape de l'algorithme, le nouveau sommet qui doit être coloré est choisi en fonction d'un critère particulier appelé la saturation. La *saturation* d'un sommet non coloré est le nombre de couleurs interdites pour ce sommet, c'est-à-dire le nombre de couleurs utilisées par ses voisins déjà colorés. A chaque étape, l'heuristique choisit le sommet dont la saturation est maximale. Un critère secondaire utilisé pour départager les *ex-aequo* consiste à choisir le sommet de plus grand degré. La couleur choisie est la couleur autorisée de plus petit rang possible. La complexité temporelle de cet algorithme est de l'ordre de $\mathcal{O}(n^2 \times log(n))$.

1.3.2.2.2 La recherche locale

La recherche locale a été utilisée depuis le milieu des années 80 pour traiter le problème de coloration. Les algorithmes proposés diffèrent par la méta-heuristique utilisée (généralement la recherche tabou ou le recuit simulé) mais aussi par la représentation du problème [14, 61, 36, 95]. Une comparaison entre certaines de ces stratégies est proposée dans [36]. L'approche de recherche locale est extrêmement puissante pour le problème et a obtenu de très bons résultats. Il faut cependant noter que les al-

gorithmes les plus efficaces existant aujourd'hui combinent la recherche locale avec d'autres stratégies comme par exemple l'approche évolutionnaire.

Les stratégies utilisées pour représenter le problème par la recherche locale sont nombreuses et variées. Cette diversité est due au fait qu'il n'existe pas de voisinage naturel utilisable pour le problème. En effet, le voisinage naturel qui consiste à changer la couleur d'un sommet quelconque ne convient pas car nous risquons de colorer ce sommet avec une couleur déjà utilisée par l'un des voisins. Il faut donc utiliser des mécanismes plus complexes pour définir le voisinage (*e.g.* chaînes de Kempe, l'approche de pénalisation, l'approche de réparation et l'approche constructive).

La recherche tabou de Herz et de Werra [61] a été améliorée en 1996 par Dorne et Hao [35]. La recherche tabou a été utilisée en 1996 par Fleurent et Ferland [47] pour améliorer la qualité de leur population de solutions dans le cadre d'une méthode évolutionnaire [47]. Parmi les travaux les plus récents sur la recherche tabou on peut citer ceux de González-Verlade et Laguna [67].

1.3.2.2.3 Les méthodes exactes

Il n'existe pas un grand nombre de méthodes exactes efficaces pour résoudre le problème de coloration : l'algorithme de branch & bound basé sur DSatur et proposé par Brélaz [9] est resté la référence. Des méthodes exactes basées sur des stratégies implicites d'énumération ont été proposées dans [77, 106]. Mehrotra et Trick [88] ont proposé un modèle de programmation linéaire qu'ils ont résolu par une méthode de génération de colonnes. Mendez Diaz et Zabala [33, 34] ont proposé un algorithme de branch & cut pour ce problème. Quelques temps après, Lucet *et al.* [82] ont proposé une méthode de résolution exacte du problème de coloration par des méthodes de *décomposition arborescente* (voir section 1.3.3).

1.3.2.3 Variantes

Le problème de *Soft Graph Coloring* (SGC) est une variante du problème de coloration. Etant donné un nombre de couleurs limité, l'objectif est de donner une couleur à chacun des sommets d'un graphe non orienté de manière à minimiser le nombre de conflits, un conflit étant une arête connectant deux sommets de même couleur.

Les algorithmes pour le SGC se basent d'habitude sur une métrique, dite *degré de conflit* [45], pour évaluer la qualité des solutions (colorations). Le degré de conflits $\gamma_{G=(I,E)}$ d'une coloration est égale au poids total des arêtes de conflits divisé par le poids total de toutes les arêtes :

$$\gamma_{G=(I,E)} = \frac{\sum_{(u,v)\in E, c_u=c_v} w_{(u,v)}}{\sum_{(u,v)\in E} w_{(u,v)}} \tag{1.3}$$

où $w_{(u,v)}$ le poids de l'arête (u,v) et c_u la couleur d'un sommet u.

Les méthodes de résolution dédiées au SGC [44, 45] présentent des caractéristiques communes et ont été baptisées méthodes de réparation (*Repair Methods*). Pour

résoudre le SGC, ces méthodes suivent le principe général suivant. A partir d'une solution complète, on cherche itérativement à modifier la valeur d'une variable de manière à faire diminuer le nombre de variables en conflits.

1.3.3 Décomposition arborescente

Dans cette section, nous présentons une méthode qui permet d'exploiter la structure d'un graphe. Dans une première partie, nous donnons quelques définitions préliminaires, puis nous décrivons des méthodes pour calculer une décomposition arborescente.

1.3.3.1 Définitions

Les notions de "cluster"' et de "clique" permettent d'identifier des sous-graphes d'un graphe donné.

Définition 1.8 ***(Cluster, Clique, Clique maximale)*** *Soit $G = (V, E)$ un graphe non orienté. Un sous-ensemble $U \subseteq V$ de l'ensemble des sommets de G est appelé un cluster de G. On dit que le cluster U est une clique si $\forall (u, v) \in U^2 / u \neq v, (u, v) \in E$. Si de plus $\forall x \in V \setminus U, U \cup \{x\}$ n'est pas une clique alors U est une clique maximale de G.*

Les cliques représentent les parties les plus connexes d'un graphe. Le calcul d'une décomposition en clusters permet d'identifier les clusters du graphe qui présentent cette caractéristique.

Définition 1.9 ***(Décomposition en clusters** ou **décomposition arborescente)*** *[101] Une décomposition arborescente d'un graphe $G = (V, E)$ est un couple (C, T) où $T = (N, A)$ est un arbre avec un ensemble de nœuds N et un ensemble d'arêtes A, et $C = \{c_i : i \in N\}$ une famille de sous-ensembles de V tels que :*

1. $\cup_{i \in N} c_i = V$,
2. $\forall (v, w) \in E, \exists i \in N / u \in c_i$ *et* $v \in c_i$,
3. $\forall (i, j, k) \in N^3$, *si* c_k *est sur un chemin de* c_i *à* c_j *alors* $c_i \cap c_j \subset c_k$.

Un exemple de décomposition arborescente est donné dans la figure 1.2. Pour un graphe donné, il y a en général un grand nombre de décompositions arborescentes possibles. D'ailleurs tous les graphes admettent au moins la décomposition triviale composée d'un seul cluster contenant tous les sommets du graphe. Bien sûr, seules les décompositions arborescentes les plus représentatives de la structure du graphe vont nous intéresser. Nous allons pour cela introduire le concept de *largeur* d'une décomposition arborescente.

Définition 1.10 ***(Séparateur)*** *L'intersection de deux ou plusieurs clusters d'une décomposition arborescente est appelée séparateur.*

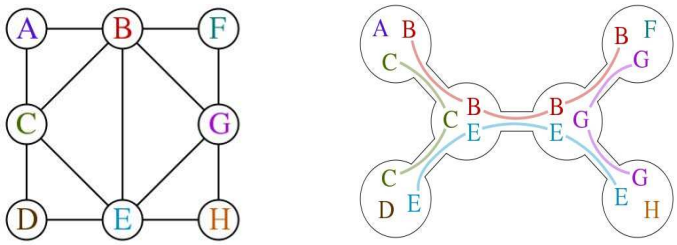


FIGURE 1.2 – *Un graphe G avec huit sommets et une décomposition arborescente comportant six clusters.*

Définition 1.11 ***(Largeur d'une décomposition arborescente)*** *La largeur d'une décomposition arborescente, noté $w(C,T)$ pour une décomposition (C,T) d'un graphe G, est égale à $max_{i\in N}(|c_i|-1)$. La largeur d'arbre d'un graphe est égale à la plus petite largeur sur l'ensemble de ses décompositions arborescentes possibles.*

Une décomposition arborescente est considérée de bonne qualité si sa largeur est proche de la largeur d'arbre du graphe. Toute clique est contenue au moins une fois dans un cluster. La largeur d'arbre d'un graphe est donc minorée par la taille de sa plus grosse clique.

Calculer une décomposition de largeur minimale est un problème $\mathcal{NP}$-difficile [115]. Toutefois, ce problème peut être résolu en temps polynomial si le graphe à décomposer est triangulé. Nous présentons donc dans la section suivante quelques définitions et techniques liées aux graphes triangulés.

1.3.3.2 Triangulation de graphe

Dans cette section, nous passons en revue de quelques notions qui nous intéressent à propos des graphes triangulés et de la triangulation de graphes.

Définition 1.12 ***(Graphe triangulé)*** *On dit qu'un graphe est triangulé si tous ses cycles de longueur 4 ou plus ont au moins une corde, c'est-à-dire une arête connectant deux sommets non consécutifs dans le cycle.*

Définition 1.13 ***(Triangulation)*** *L'opération qui consiste à rendre triangulé un graphe en lui ajoutant de nouvelles arêtes est appelée triangulation.*

La figure 1.3 montre un exemple de triangulation de graphe.

Le problème de "minimum fill-in" qui consiste à trouver une triangulation minimale est un problème $\mathcal{NP}$-complet [115]. Toutefois, en utilisant de bonnes heuristiques, on peut trouver des triangulations qui en sont proches en temps polynômial. Un des moyens de procéder est de calculer pour tout graphe à trianguler son "graphe d'élimination".

Définition 1.14 ***(Graphe d'élimination)*** *Soit $G=(V,E)$ un graphe et notons $|V|=n$. Soit $\alpha=\{1,2,\ldots,n\}\rightarrow V$ un ordre sur les sommets de G. On appelle graphe d'élimination de G selon α le graphe obtenu en prenant successivement chaque sommet $v\in V$ dans l'ordre*

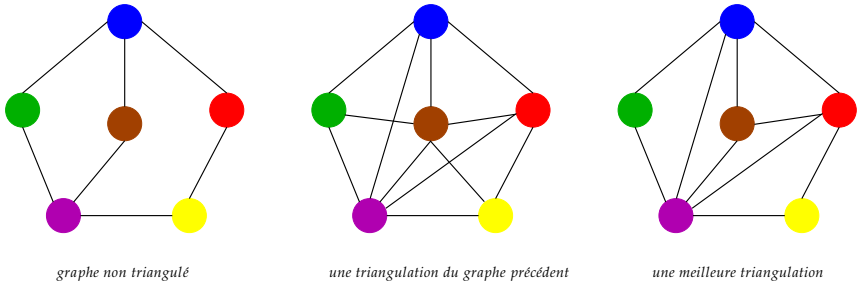


FIGURE 1.3 – *Différentes triangulations d'un même graphe*

α et en ajoutant une arête entre tous les voisins de v situés après v selon α. Les arêtes ainsi ajoutées sont appelées "arêtes de fill-in". α est un ordre d'élimination parfait si aucune arête n'est ajoutée durant la construction du graphe d'élimination.

Théorème 1.1 *[49] Un graphe est triangulé si et seulement s'il a un ordre d'élimination parfait.*

Ce théorème donne une caractéristique importante des graphes triangulés. On en déduit aussi que tout graphe d'élimination est triangulé.

Pour trianguler un graphe il suffit donc de calculer un graphe d'élimination. La complexité temporelle maximale de ce procédé est de l'ordre de $\mathcal{O}(n \times d^2)$ où n est le nombre de sommets du graphe et d le degré du sommet de plus grand degré. La qualité de la triangulation obtenue dépend uniquement de l'ordre d'élimination α choisi. Trouver l'ordre d'élimination qui aboutira à la meilleure triangulation est un problème $\mathcal{NP}$-complet que l'on peut résoudre avec les méthodes classiques, exactes (*i.e.* branch & bound) ou approchées (*i.e.* recuit simulé, algorithmes génétiques, etc.), plus ou moins coûteuses en temps de calcul.

Nous décrivons en particulier l'algorithme glouton "Maximum cardinality search (MCS)" proposé par Tarjan et Yannakakis [110] en 1984 et qui sera utilisé dans le chapitre 3. L'algorithme 2 décrit les détails d'exécution de MCS.

Algorithme 2 : Heuristique Maximal Cardinality Search (MCS)

donnée : Un graphe $G = (V, E)$ de n sommets.
résultat : Un ordre d'élimination parfait α.

```
1 pour chaque sommet v ∈ V faire
2     label[v] ← 0;
3 pour chaque valeur i allant de n à 1 faire
4     choisir le sommet non numéroté v ayant le plus grand label;
5     α[v] ← i;
6     pour chaque sommet non numéroté w ∈ V tel que (v, w) ∈ E faire
7         label[v] ← label[v] + 1;
8 retourner α;
```

La complexité temporelle de MCS est de l'ordre $\mathcal{O}(|V| + |E|)$. Tarjan et Yannakakis [110] ont montré que MCS trouve un ordre d'élimination parfait si le graphe est triangulé.

1.3.3.3 Calculer une décomposition arborescente

Trouver une décomposition arborescente optimale d'un graphe G est facile si G est triangulé. Nous avons vu dans la section précédente comment trianguler un graphe quelconque. Nous verrons dans cette section comment calculer une décomposition arborescente d'un graphe après l'avoir triangulé.

Pour toute décomposition arborescente d'un graphe non triangulé $G = (V, E)$, il existe une triangulation de G qui admet la même décomposition. En particulier, il existe une triangulation de G qui admet la même décomposition optimale. Trouver la largeur d'arbre de G revient ainsi à trouver une triangulation de G dont la largeur d'arbre est minimale [76].

Pour calculer une décomposition arborescente d'un graphe triangulé G, il suffit de calculer son arbre de jonction, un arbre dont les nœuds sont les éléments de l'ensemble des cliques maximales de G et dans lequel deux nœuds sont reliés entre eux si et seulement si les deux cliques correspondantes ont au moins un sommet de G commun.

Le calcul des cliques maximales est très simple à faire si l'on utilise les constatations de Dechter et Pearl [42] et de Tarjan et Yannakakis [110] qui indiquent que pour un graphe triangulé et son ordre associé, tout sommet forme une clique avec tous ses voisins le suivant dans l'ordre.

Définition 1.15 *[110] Toute clique maximale dans un graphe triangulé $G = (V, E)$ est de la forme $\{v\} \cup X(v)$, pour chaque sommet $v \in V$ où $X(v) = \{w \ / \ (v, w) \in E, \alpha(v) < \alpha(w)\}$.*

Définition 1.16 *[110] Un graphe triangulé contient au plus n cliques maximales.*

Tarjan et Yannakakis [110] ont proposé un algorithme de complexité temporelle $\mathcal{O}(n + m)$ pour énumérer les cliques maximales d'un graphe triangulé G. Cet algorithme se base essentiellement sur un ordre d'élimination parfait calculé par MCS.

1.4 Le problème de bin packing uni- et bi-dimensionnel

Dans cette section, nous définissons le problème de bin packing $\mathcal{BPP}$. Nous décrivons aussi quelques méthodes de résolution dédiées à la variante uni- et bi-dimensionnelle ($\mathcal{BPP}$-1$\mathcal{D}$ et $\mathcal{BPP}$-2$\mathcal{D}$ respectivement). Nous décrivons en particulier des modèles mathématiques, des prétraitements, des bornes inférieures polynomiales et des heuristiques rapides.

Le *problème de bin packing uni-dimensionnel* ($\mathcal{BPP}$-1$\mathcal{D}$) consiste à minimiser le nombre de containers uni-dimensionnel (bins) nécessaires pour ranger une liste d'objets caractérisés par leur longueur. Ce problème est $\mathcal{NP}$-complet [51]. Une instance de

$\mathcal{BPP}$-1$\mathcal{D}$, notée $\langle I, w, W\rangle$, comprend un ensemble $I = \{1, 2, \ldots, n\}$ de n objets et une fonction w qui associe à chaque objet i une valeur w_i qui correspond à sa longueur, et W une valeur positive représentant la taille du bin.

Le *problème de bin packing bi-dimensionnel* ($\mathcal{BPP}$-2$\mathcal{D}$) est une généralisation naturelle de $\mathcal{BPP}$-1$\mathcal{D}$. Il s'agit de minimiser le nombre de grands rectangles (bins) identiques pour ranger une liste d'objets rectangulaires. Les objets doivent être rangés de telle manière que les côtés des rectangles soient parallèles à ceux du bin. On note $\langle I, w, h, W, H, G\rangle$ une instance de $\mathcal{BPP}$-2$\mathcal{D}$, $I = \{1, 2, \ldots, n\}$ est la liste des objets à ranger, et une fonction w (resp. h) qui associe à chaque objet i une valeur w_i (resp. h_i) qui correspond à sa longueur (resp. sa largeur), W et H représentent la longueur et la largeur du bin.

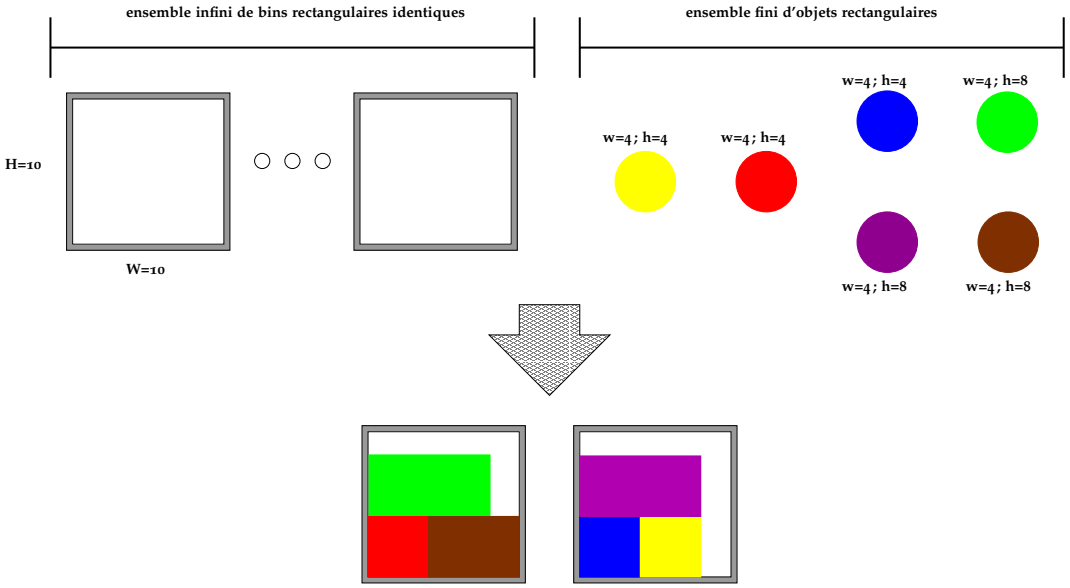


FIGURE 1.4 – *Une instance de $\mathcal{BPP}$-2$\mathcal{D}$ et une solution possible de cette instance.*

Dans le problème qui nous intéresse, les objets ont une orientation fixe et doivent être rangés de telle manière que leurs côtés soient parallèles à ceux du bin. Une des différences fondamentales entre $\mathcal{BPP}$-1$\mathcal{D}$ et $\mathcal{BPP}$-2$\mathcal{D}$ réside dans le *problème de faisabilité* : étant donné un ensemble d'objets et un bin, existe-t-il un placement réalisable pour ces objets dans le bin ? Alors que ce problème est trivial en une dimension (il suffit de sommer la longueur des objets), il est $\mathcal{NP}$-complet en deux dimensions. On a donc deux types de problèmes imbriqués à résoudre : un problème d'affectation et plusieurs problèmes de faisabilité.

1.4.1 Modèles mathématiques

Dans cette section, nous présentons des formulations mathématiques pour le $\mathcal{BPP}$-1$\mathcal{D}$ et le $\mathcal{BPP}$-2$\mathcal{D}$.

1.4.1.1 Cas uni-dimensionnel

En 1960, Kantorovich [70] a introduit le premier modèle de programmation linéaire en nombre entier pour le problème de cutting-stock. Le $\mathcal{BPP}$-1$\mathcal{D}$ peut être modélisé de la même manière. Le modèle suivant vise à minimiser le nombre de bins utilisés par une solution.

$$\begin{aligned}
\min \quad & \sum_{k=1}^{n} y_k && && (1.4)\\
\text{t.q.} \quad & \sum_{k=1}^{n} x_{ik} = 1 && (\forall i \in I) && (1.5)\\
& \sum_{i \in I} w_i \times x_{ik} \leq W \times y_k && (k = 1, \ldots, n) && (1.6)\\
& y_k \in \{0,1\} && (k = 1, \ldots, n) && (1.7)\\
& x_{ik} \in \{0,1\} && (\forall i \in I, k = 1, \ldots, n) && (1.8)
\end{aligned}$$

où y_k est une variable entière qui prend la valeur 1 si le bin k est utilisé et 0 sinon. La variable x_{ik} est aussi une variable entière qui prend la valeur 1 si l'objet i est placé dans le bin k et 0 sinon. Les contraintes (1.5) contrôlent le fait que chaque objet est placé dans un bin. Les contraintes (1.6) dites contraintes de sac-à-dos vérifient que la somme des tailles des objets placés dans un bin ne dépasse pas la taille du bin.

Une relaxation LP de ce modèle donne une estimation par défaut sur la solution optimale du modèle. Martello et Toth [85] ont montré que la valeur de cette borne inférieure pourrait atteindre $\frac{1}{2} \times \mathcal{OPT}$ dans le pire des cas.

Le modèle compact présenté ci-dessus peut être décomposé par la méthode de décomposition de Dantzig-Wolfe [29]. Gilmore et Gomory [54, 55] ont étudié l'idée qui consiste à énumérer tous les sous-ensembles d'objets qui peuvent être rangés dans un bin. On note un tel sous-ensemble *pattern* et on représente un pattern p par une colonne $(a_{1p}, \ldots, a_{ip}, \ldots, a_{np})^T$, où a_{ip} prend la valeur 1 si l'objet i est présent dans le pattern p et 0 sinon. Soit P l'ensemble de patterns valides, *i.e.* l'ensemble des patterns p pour lesquels

$$\sum_{i=1}^{n} w_i \times a_{ip} \leq W. \tag{1.9}$$

Soit z_p une variable booléenne qui prend la valeur 1 si le pattern p est utilisé et 0 sinon ($p \in P$). Le $\mathcal{BPP}$-1$\mathcal{D}$ peut être modélisé sous forme d'un modèle de partitionnement d'ensemble comme suit :

$$\begin{aligned}
\min \quad & \sum_{p \in P} z_p && && (1.10)\\
\text{t.q.} \quad & \sum_{p \in P} a_{ip} z_p = 1 && (\forall i \in I) && (1.11)\\
& z_p \in \{0,1\} && (\forall p \in P) && (1.12)
\end{aligned}$$

Les contraintes (1.11) imposent que chaque objet i soit placé dans un bin. L'inconvénient de ce méthode réside dans le fait que le nombre de patterns possibles est

exponentiel. Résoudre la relaxation linéaire du modèle (1.10)-(1.12) peut se révéler difficile. Pour ce faire, on utilise la méthode de génération de colonnes introduite par Dantzig et Wolfe [29] . On initialise le modèle avec un sous-ensemble de colonnes uniquement, et on le résout à l'optimalité. On parle de problème maitre. Lorsque cela est fait, le problème est de déterminer une colonne valide qui pourrait être ajoutée au problème maitre et réduire la valeur de la fonction objectif. La phase consistant à calculer une telle colonne est appelé étape de pricing. Dans le cas du bin-packing, ce problème est difficile : il s'agit du problème classique de sac à dos.

1.4.1.2 Cas bi-dimensionnel

Un modèle linéaire pour le $\mathcal{BPP}$-$2\mathcal{D}$ peut être établi en utilisant les techniques de modélisation proposées par Onodera *et al.* [97] et Chen *et al.* [18].

Pour résoudre le $\mathcal{BPP}$-$2\mathcal{D}$, Gilmore et Gomory [56] ont adapté leur modèle (1.10)-(1.12) proposé dans [54, 55] pour le $\mathcal{BPP}$-$1\mathcal{D}$. La faiblesse de cette modélisation réside dans le très grand nombre de colonnes. Gilmore et Gomory [56] ont présenté des méthodes pour générer dynamiquement les colonnes en utilisant une résolution pseudo-polynomiale de problèmes de sac-à-dos en une dimension $\mathcal{KP}$-$1\mathcal{D}$.

Il existe d'autres manières de modéliser le problème qui peuvent être mieux adaptées à la méthode exacte utilisée pour résoudre le $\mathcal{BPP}$-$2\mathcal{D}$. Parmi ces modélisations on cite la "modélisation par une liste de coordonnées", la "modélisation sous forme de graphe d'intervalles" et la "modélisation basée sur la permutation des objets" (voir [21] pour plus de détails).

1.4.2 Prétraitements

Le problème de bin packing bi-dimensionnel $\mathcal{BPP}$-$2\mathcal{D}$ est difficile à résoudre de manière exacte lorsque la taille du problème devient grande. Pour cette raison, il est intéressant de réduire la taille du problème en effectuant des rangements optimaux d'objets. Les prétraitements que nous décrivons ci-dessous ont un intérêt différent. Celui de Martello et Vigo [86] permet de réduire la taille de l'instance, tandis que celui de Boschetti et Mingozzi [7] permet d'augmenter la taille de certains objets.

1.4.2.1 Le prétraitement de Martello et Vigo

Martello et Vigo [86] proposent deux prétraitements qui peuvent aussi s'appliquer en cours de méthode quand on a placé un objet i de grande taille dans un bin. L'idée est de trouver le *meilleur* ensemble d'objets que l'on peut placer avec i. Les auteurs proposent des méthodes élémentaires pour les cas où un ou deux objets au maximum peuvent être placés avec i.

Pour un objet donné i, soit $Q^i = \{j \in I : h_i + h_j \leq H\} \cup \{j \in I : w_i + w_j \leq W\}$, l'ensemble des objets compatibles avec i. Si $Q^i = \emptyset$, le prétraitement effectué est immédiat, dans le cas contraire, les auteurs proposent de calculer une borne supérieure k pour le nombre d'objets qui peuvent être rangés avec i.

1. Si $Q^i = \emptyset$, alors on ne peut ranger aucun objet avec i. Cet objet forme donc un sous-problème trivial et peut être ranger seul dans un bin.
2. Si $Q^i \neq \emptyset$, on considère j le plus grand objet de Q^i. Un prétraitement est appliqué dans deux cas de figure :
   (a) si $k = 1$, et que j a une plus grande hauteur et une plus grande largeur que tous les autres objets de Q^i
   (b) si $k = 2$ et tous les couples de pièces de Q^i différentes de j peuvent être rangés dans une surface égale à celle de j.

Dans les deux cas, i et j sont placés dans le même bin et ne sont plus examinés par la suite. Cette méthode peut être généralisée à de plus grandes valeurs de k, mais le coût algorithmique devient rapidement trop élevé.

1.4.2.2 Le prétraitement de Boschetti et Mingozzi

Le prétraitement qui suit est proposé par Boschetti et Mingozzi [7]. Pour un objet donné j, on peut calculer la largeur minimum obligatoirement perdue lorsque cet objet est placé dans un bin. Si on note w_j^* cette valeur, w_j peut être modifiée de la manière suivante

$$w_j \leftarrow w_j + (W - w_j^*). \tag{1.13}$$

La valeur w_j^* est la valeur optimale du problème sac-à-dos suivant

$$w_j^* = w_j + max \left\{ \sum_{i \in A - \{j\}} w_i \xi_i : \sum_{i \in A - \{j\}} w_i \xi_i \leq W - w_j, \xi_i \in \{0,1\} \right\}. \tag{1.14}$$

Une méthode pseudo-polynomiale classique peut calculer w_j^* en $\mathcal{O}(W \times n)$. La procédure de réduction dépend de l'ordre dans lequel les objets j sont considérés. Un critère heuristique peut être utilisé pour choisir cet ordre. Par exemple, il est intéressant d'accroître la largeur des objets qui ont une grande hauteur afin d'obtenir une plus grande surface totale. Une procédure similaire est obtenue en considérant la hauteur à la place de la largeur. Lorsqu'un objet a été traité, tous les objets qui faisaient partie de la solution du problème *subset-sum* ne peuvent pas être modifiés.

1.4.3 Bornes inférieures polynomiales

Dans cette section, nous rappelons les dernières méthodes utilisées pour calculer des bornes inférieures polynomiales pour le problème de bin packing uni- et bi-dimensionnel.

La borne continue est la borne la plus simple pour le $\mathcal{BPP}$. Elle est calculée en sommant les tailles des objets et en divisant le résultat par la taille d'un bin

$$LB_0 = \left\lceil \frac{\sum_{i \in I} w_i}{W} \right\rceil. \tag{1.15}$$

Cette borne fournit de bons résultats lorsque les tailles des objets sont petites ou uniformément réparties sur $[0, W]$ [5], mais peut se révéler très mauvaise dans les autres cas. Martello et Vigo [86] ont montré qu'elle pouvait atteindre $\frac{1}{2} \times \mathcal{OPT}(I)$ pour le $\mathcal{BPP}$-1$\mathcal{D}$.

Dans la suite, nous présentons le concept de *fonctions dual-réalisables* qui sont utilisées pour calculer des bornes inférieures pour le $\mathcal{BPP}$ meilleures que la borne continue. D'autres bornes inférieures qui ne se basent pas sur des DFF ont été aussi proposées pour le $\mathcal{BPP}$. Dans [65], Jarboui *et al.* ont proposé des bornes inférieures qui se basent sur des estimations du nombre d'objets qui peuvent être placés dans un bin.

1.4.3.1 Fonctions dual-réalisables (DFF)

Le concept de fonctions dual-réalisables (DFF) a été proposé par Johnson [68] et utilisé entre autres par Lueker [83], Chao *et al.* [16], Fekete et Schepers [43], Carlier *et al.* [12] et Crainic *et al.* [28] pour calculer des évaluations par défaut pour le $\mathcal{BPP}$. Un survol complet et détaillé de la littérature sur les DFF appliquées aux problèmes de cutting et packing a été établi par Clautiaux *et al.* [20].

Ces fonctions ont la propriété suivante : pour tout ensemble de réels, si leur somme est inférieure à un, alors cette propriété reste vraie après transformation.

Definition 1.4.1 *$g : [0, 1] \to [0, 1]$ est une "fonction dual-réalisable (DFF)" si pour tout ensemble fini S de réels, on a*

$$\sum_{x \in S} x \leq 1 \Rightarrow \sum_{x \in S} g(x) \leq 1 \tag{1.16}$$

Une manière d'améliorer la borne continue LB_0 est de modifier la taille des objets en utilisant une DFF. L'intérêt de la méthode est d'augmenter la taille de certains objets, en réduisant la taille d'autres objets de manière à obtenir une surface totale plus importante que dans l'instance originale. Les méthodes utilisant les DFF considèrent toujours un bin de longueur unitaire et des objets dont la longueur est un réel compris entre 0 et 1. Pour pouvoir utiliser ces fonctions, il faut donc projeter les dimensions dans l'intervalle $[0, 1]$. En utilisant des DFF sur une instance de $\mathcal{BPP}$, toute borne inférieure pour l'instance obtenue est aussi une borne inférieure pour l'instance initiale.

Dans [20], Clautiaux *et al.* ont montré que les fonctions f_0 et f_2 proposées par Fekete et Schepers [43] et Carlier *et al.* [12] donnent les meilleurs résultats en moyenne. Ce sont donc ces fonctions qui sont utilisées dans ce manuscrit.

La fonction f_0[43] Cette fonction repose sur l'observation suivante : si aucun objet ne peut être rangé avec un objet i, la taille de i peut être augmentée jusqu'à la taille du bin. D'une manière analogue, pour une valeur donnée k ($1 \leq k \leq \frac{C}{2}$), la taille

de tous les objets de taille strictement supérieure à $C-k$ peut être augmentée si tout objet plus petit que k est supprimé. Nous notons f_0^k la famille de DFF correspondante.

$$f_0^k : [0,C] \to [0,C]$$
$$x \mapsto \begin{cases} C \text{ si } x > C-k \\ x \text{ si } C-k \geq x \geq k \\ 0 \text{ sinon} \end{cases}$$

La fonction f_2 Cette fonction repose sur des techniques d'arrondis. C'est une amélioration de celle qui est utilisée implicitement par Boschetti et Mingozzi [7] dans leur bornes inférieures.

Soit f_2^k la fonction définie de la manière suivante :

$$f_2^k : [0,C] \to \left[0, 2 \times \left\lfloor \frac{C}{k} \right\rfloor\right]$$
$$x \mapsto \begin{cases} 2 \times \left(\left\lfloor \frac{C}{k} \right\rfloor - \left\lfloor \frac{C-x}{k} \right\rfloor \right) & \text{si } x > \frac{1}{2}C \\ \left\lfloor \frac{C}{k} \right\rfloor & \text{si } x = \frac{1}{2}C \\ 2 \times \left\lfloor \frac{x}{k} \right\rfloor & \text{si } \frac{1}{2}C > x \end{cases}$$

1.4.3.2 DFF dépendantes de la donnée (DDFF)

Les DFF sont définies indépendamment de l'instance. Carlier *et al.* [12] ont défini une autre classe de fonctions qui dépend de la taille des objets dans l'instance considérée.

Definition 1.4.2 *Soit $I = \{1,\dots,n\}$, $w_1, w_2, \dots, w_n$ n valeurs entières et C un entier tel que $W \geq w_i$ pour $i = 1,\dots,n$. Une "fonction dual-réalisable dépendante de la donnée (DDFF)" f associée à W et $w_1, w_2, \dots, w_n$ est une application discrète de $[0,W]$ dans $[0,W']$ telle que*

$$\forall I_1 \subset I, \sum_{i \in I_1} w_i \leq W \Rightarrow \sum_{i \in I_1} f(w_i) \leq f(W) = W'$$

Pour une instance donnée D, une DDFF dépendante de D peut être utilisée comme une DFF. Cette classe de fonctions prend en compte le nombre d'occurences de chaque valeur, et peut ainsi conduire à des évaluations qui améliorent strictement celles obtenues avec les DFF.

Carlier *et al.* [12] ont proposé la fonction f_1 pour concrétiser leur concept de DDFF. Soit f_1^k une nouvelle fonction définie pour un paramètre donné k, $1 \leq k \leq \frac{1}{2}C$, et une liste d'entiers $c_1, c_2, \dots, c_n$ ($I = \{1,\dots,n\}$). Nous introduisons l'ensemble $J = \{i \in I : \frac{1}{2}C \geq c_i \geq k\}$ et $M_C(X,J)$ la valeur optimale pour le problème de sac-à-dos en une dimension ($\mathcal{KP}$-1$\mathcal{D}$) induit par l'ensemble J et la taille X. Cette valeur est égale au nombre maximum d'objets qui peuvent être contenus ensemble dans un container de taille X. Le problème peut être résolu en temps linéaire si les objets sont triés par

ordre croissant de taille. Cette fonction a été utilisée implicitement par Boschetti et Mingozzi [7].

$$f_1^k : [0, C] \to [0, M_C(C, J)]$$
$$x \mapsto \begin{cases} M_C(C, J) - M_C(C - x, J) & \text{si } x > \frac{1}{2}C \\ 1 & \text{si } \frac{1}{2}C \geq x \geq k \\ 0 & \text{sinon} \end{cases}$$

Il est important de préciser que f_1 n'est pas une DFF car $M_C(C, J)$ est dépendant des valeurs $c_1, \ldots, c_n$. Appliquer f_1 sur une autre occurence de problème ne permet pas automatiquement d'obtenir des bornes inférieures.

1.4.3.3 Application au cas bi-dimensionnel

Si la borne continue peut être utilisée en une dimension lorsque l'on cherche une évaluation rapide, elle est beaucoup moins efficace pour $\mathcal{BPP}$-$2\mathcal{D}$. Dans le pire des cas, pour $\mathcal{BPP}$-$2\mathcal{D}$, la borne continue LB_0

$$LB_0 = \left\lceil \frac{\sum_{i \in I} w_i \times h_i}{W \times H} \right\rceil \tag{1.17}$$

peut atteindre un quart de l'optimum. Pour réaliser la faiblesse de cette borne, il suffit de considérer le cas où tous les objets sont de taille $(\frac{W}{2} + \epsilon, \frac{H}{2} + \epsilon)$ pour ϵ petit.

Carlier *et al.* [12] ont proposé des nouvelles bornes inférieures pour le $\mathcal{BPP}$-$2\mathcal{D}$. Elles sont basées sur les fonctions f_0, f_1 et f_2 décrites plus haut. Ils montrent aussi que les familles f_0, f_1 et f_2 peuvent être combinées pour obtenir des bornes qui dominent strictement celles de la littérature.

1.4.4 Méthodes heuristiques

Dans cette section, nous présentons des heuristiques utilisées pour résoudre le problème de bin packing uni- et bi-dimensionnel. Nous nous limitons aux méthodes qui fournissent de bons résultats en pratique, même si elles n'ont pas de garantie de performance.

1.4.4.1 Cas uni-dimensionnel

Pour une synthèse très complète des travaux réalisés pour le cas uni-dimensionnel, se référer aux travaux de Coffman *et al.* [25].

Next Fit Dans cette stratégie on ne considère qu'un bin ouvert à la fois. Les objets sont rangés successivement dans le bin courant tant qu'il y a de la place pour l'objet i en cours, sinon ce bin est fermé et un nouveau bin est ouvert. Une heuristique qui adopte une stratégie next-fit a l'avantage d'avoir une complexité temporelle linéaire en fonction du nombre d'objets à placer. Par contre, le fait de ne considérer qu'un seul bin à la fois cause beaucoup de perte d'espaces exploitables.

First Fit Initialement un seul bin est considéré. Quand il n'y a plus de place dans le premier bin pour ranger l'objet en cours, un deuxième bin est alors ouvert mais sans fermer le premier. Dans une étape intermédiaire où on dispose de k bins ouverts numérotés de 1 à k selon l'ordre de leur première utilisation, l'objet en cours i est rangé dans le bin de plus faible numéro qui peut le contenir. Dans le cas où aucun bin ne peut contenir i, un nouveau bin $k+1$ est alors créé sans fermer les autres. L'ordre selon lequel on traite les objets est crucial pour la qualité de la solution.

Best Fit D'une façon similaire à celle de first-fit, la stratégie best-fit garde les bins toujours ouverts. Cependant, le choix du bin dans lequel l'objet i en cours va être placé dépend des valeurs des *gaps* (espaces libres) présentes dans les bins. Ainsi, i sera placé dans le bin de plus petit gap pouvant le contenir. Les heuristiques best-fit et first-fit peuvent être implantées en $\mathcal{O}(n \times log(n))$ en utilisant une structure de données appropriée [68].

Worst Fit Il s'agit d'une variante du best-fit, suivant laquelle l'objet en cours est placé dans le bin de plus grand gap pouvant le contenir. Cette stratégie peut produire de meilleures solutions que best-fit pour certaines instances.

Dans les heuristiques ci-dessus, les objets sont traités selon un ordre donné. Le nom de l'heuristique dans ce cas s'accorde avec le stratégie de tri adoptée. On parle par exemple de "First Fit Decreasing Height" quand on applique l'heuristique "First Fit" sur une liste d'objets triés par ordre décroissant des hauteurs.

1.4.4.2 Cas bi-dimensionnel

Contrairement à $\mathcal{BPP}$-1$\mathcal{D}$ pour lequel on trouve essentiellement des algorithmes avec garantie de performance (voir [24, 23, 26]), les méthodes proposées pour $\mathcal{BPP}$-2$\mathcal{D}$ sont en général des heuristiques.

Dans cette section, nous ne présentons que les heuristiques qui fournissent de bons résultats en pratique, même si elles n'ont pas de garantie de performance. Elles peuvent être divisées en deux catégories : les méthodes en une phase et les méthodes en deux phases.

1.4.4.2.1 Méthodes en deux phases

Les méthodes en deux phases fonctionnent de la manière suivante. Dans un premier temps, on cherche une solution pour le problème de strip-packing suivant : l'ensemble des objets est l'ensemble original I, et la largeur du bin est égale à W. On détermine une solution pour le problème $\mathcal{BPP}$-2$\mathcal{D}$ original en découpant la bande obtenue en bins de hauteur H. Les méthodes qui suivent diffèrent par les algorithmes utilisés dans les deux phases. Dans la majorité des cas, le premier algorithme range les objets par *niveaux*. Lorsqu'un objet i est rangé à gauche, on cherche à ranger les objets suivants dans la bande horizontale située à droite de i. Cette méthode permet de ne gérer que des bandes de largeur W et de hauteur variable dans la deuxième étape, et donc de pouvoir le traiter comme un $\mathcal{BPP}$-1$\mathcal{D}$.

Certaines algorithmes utilisent des heuristiques dédiées au $\mathcal{BPP}$-1$\mathcal{D}$ (*i.e.* best fit, voir section 1.4.4.1) au cours de la première phase. Parmi ces algorithmes nous citons la "méthode Hybrid Best Fit "(HBF)" proposée par Berkey et Wang en 1987 [6].

D'autres méthodes dont on cite "la méthode Floor-Ceiling (FC)" proposée par Lodi *et al.* [81] en 1998 se permettent, lorsqu'un niveau devient plein, de le *retourner*, échangeant le haut et le bas, la gauche et la droite, puis de recommencer à ranger des objets à gauche.

Certaines méthodes se basent aussi sur la transformation du problème en un problème de sac-à-dos comme "la méthode Knapsack packing (KP)" de Lodi *et al.* [81]. Les niveaux créés sont remplis au maximum à l'aide de l'algorithme de résolution du sac-à-dos uni-dimensionnel $\mathcal{KP}$-1$\mathcal{D}$.

1.4.4.2.2 Méthodes en une phase

Les méthodes en une phase consistent à ranger les objets itérativement dans les bins. Deux règles sont à définir : l'ordre dans lequel les objets sont examinés, le bin et la position dans laquelle on cherche à les placer en priorité.

L'heuristique *bottom-left* (BL) a été proposée par Coffman *et al.* [23] pour le $\mathcal{BPP}$-2$\mathcal{D}$. Elle est basée sur la stratégie *first-fit* (voir section 1.4.4.1). Elle appartient à la famille d'heuristiques préservant la condition de stabilité de type *bottom-left*. Un rectangle préserve la condition de stabilité bottom-left ssi il est placé dans le surface vide la plus basse et la plus à gauche (voir figure 1.5). A chaque itération de l'algorithme, chaque bin garde une liste de surfaces maximales, au sens de l'inclusion, vides (rectangles à bord pointillé dans la figure 1.5). L'heuristique BL procède en placant l'objet courant dans la surface la plus en bas et plus à gauche dans le premier bin qui peut l'accueillir. Le processus est ensuite itéré tant qu'il y a des objets non encore placés. Chazelle [17] en a proposé une implantation efficace.

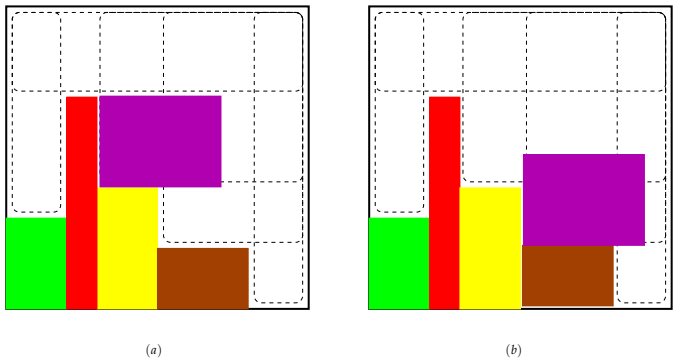


FIGURE 1.5 – *L'objet violet est placé dans une position bottom-left stable dans le bin* (b) *mais pas dans le bin* (a).

Berkey et Wang [6] ont proposé eux aussi une généralisation de l'algorithme First-Fit qu'ils nomment "Finite First Fit (FFF)".

L'heuristique "Alternate Directions (AD)" [81] se base sur le même principe que la méthode FC qui fonctionne en deux phases. L'algorithme s'initialise avec un nombre

de bins égale à une borne inférieure sur le nombre de bins nécessaires. Elle commence par placer en bas de ces bins des objets selon la règle BFD (*Best Fit Decreasing*). Les objets restants sont rangés un par un en bandes allant alternativement de gauche à droite et de droite à gauche. Lorsqu'un objet ne peut être placé dans aucune direction dans le bin courant, on l'affecte au prochain bin initialisé ou on construit un nouveau bin.

Boschetti et Mingozzi [8] proposent l'heuristique "HBP". L'idée est de relancer plusieurs fois un algorithme qui traite les objets dans un ordre fixé au début de chaque relance. On ne considère qu'un seul bin à la fois. Quand on ne peut plus placer aucun objet dans le bin courant, on le *ferme*, et il ne sera plus reconsidéré. Dans ce cas, on *ouvre* un nouveau bin et on répète l'opération tant que tous les objets ne sont pas rangés.

El Hayek *et al.* [60] ont proposé l'heuristique dite "Item Maximal Area (IMA)" qui, au contraire des heuristiques classiques qui consistent à affecter les objets aux bins dans un ordre prédéfini, ne nécessite aucun pré-ordre des objets. A chaque étape de rangement d'un objet, on cherche le meilleur couple $(objet, bin)$ parmi tous les couples réalisables. Le choix du couple est géré par un certain critère qui dépend des caractéristiques des objets et des surfaces considérées.

1.4.5 Bin packing multi-objectif

La littérature sur le bin packing multi-objectif est encore en cours de croissance. Nous présentons dans cette section brièvement les principaux travaux existants dans ce domaine. Amiouny *et al.* [108] ont considéré une variante de bin packing dans laquelle on cherche à rapprocher le centre de gravité des objets placés dans un bin d'un point de repère. Liu *et al.* [80] ont considéré un problème de bin packing bi-dimensionnel multi-objectif dans lequel les deux fonctions objectives consistent à minimiser l'espace perdu dans les bins et l'équilibrage des contenus des bins. Les auteurs ont proposé un algorithme à base d'essaim de particules qui est capable de proposer une variété de solutions pour satisfaire les spécifications des clients en terme d'espace perdu et d'équilibrage. Sathe *et al.* [103] ont considéré une variante bi-objectif d'un problème de bin packing soumis à des contraintes additionnelles. Ce problème est issue d'une application dans l'industrie métallurgique automobile. Ils proposent des modèles mathématiques pour le problème et des méthodes de résolution à base de techniques de clustering et d'algorithmes génétiques.

1.5 Le problème de bin packing avec conflits

Le problème de bin packing avec conflits ($\mathcal{BPP}$-$\mathcal{C}$) est une généralisation du $\mathcal{BPP}$, où on dispose en plus d'un graphe $G = (I, E)$ dit graphe de conflits. Chaque sommet $i \in I$ représente un objet i et chaque conflit entre deux objets (i, j) est indiqué par une arête $(i, j) \in E$. Un conflit entre deux objets est une restriction interdisant de ranger

ces deux objets dans le même bin. Ce problème $\mathcal{NP}$-difficile a été introduit par Jansen et Öhring [64] en 1997.

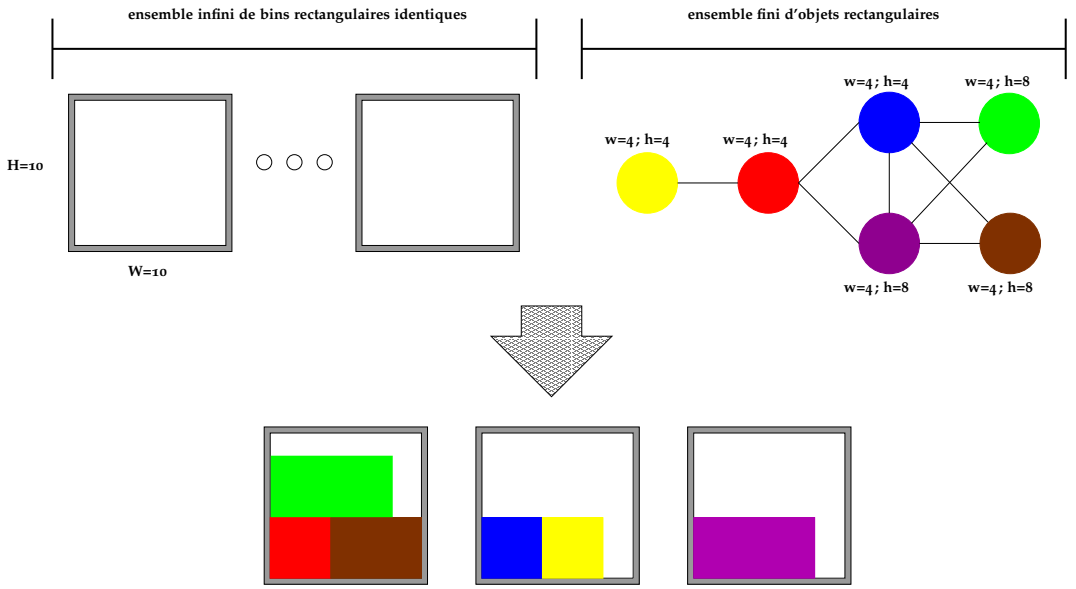


FIGURE 1.6 – *Une instance de $\mathcal{BPPC}$-2$\mathcal{D}$ et d'une solution possible de cette instance.*

Dans cette section, nous décrivons en détail les travaux de [53] et de [2] qui présentent des méthodes exactes et approchées pour le problème.

1.5.1 Modèles mathématiques

Nous présentons dans cette section un modèle linéaire pour le $\mathcal{BPPC}$-1$\mathcal{D}$.

Deux modélisations ont été présentées dans la littérature pour le $\mathcal{BPPC}$-1$\mathcal{D}$. Dans la première, Gendreau *et al.* [53] ont repris le modèle (1.4)–(1.7) dédié au $\mathcal{BPP}$-1$\mathcal{D}$ et ont ajouté les contraintes suivantes afin de gérer les conflits entre les objets

$$x_{ik} + x_{jk} \leq 1 \quad (\forall (i,j) \in E, k = 1, \ldots, n). \tag{1.18}$$

Quelques temps après, Fernandes Muritiba *et al.* [2] ont proposé un nouveau modèle pour le $\mathcal{BPPC}$-1$\mathcal{D}$. Cette fois ci, les auteurs ont repris le même modèle (1.4)–(1.7) et ont ajouté les contraintes suivantes

$$x_{ik} + x_{jk} \leq y_k \quad (\forall (i,j) \in E, k = 1, \ldots, n). \tag{1.19}$$

Dans leur article [2], Fernandes Muritiba *et al.* indiquent que les contraintes (1.19), appropriées aux contraintes de conflits classiques du "vertex coloring problem", sont plus efficaces que les contraintes de Gendreau *et al.* [53].

1.5.2 Bornes inférieures dédiées au bin packing avec conflits

Dans cette section, nous rappelons les résultats existants pour le problème de bin packing uni-dimensionnel avec conflits $\mathcal{BPPC}$-1$\mathcal{D}$. A notre connaissance, aucune borne inférieure pour le $\mathcal{BPPC}$-2$\mathcal{D}$ n'a déjà été proposée.

1.5.2.1 Des bornes à base de cliques

La borne inférieure LB_{MC} à été proposée par Gendreau *et al.* [53]. L'idée consiste à trouver une clique maximale dans le graphe $G = (V, E)$ étendu en ajoutant à E tout couple d'objets (i, j) tel que $(w_i + w_j > W)$. La taille de la clique trouvée est bien évidemment une borne inférieure pour le $\mathcal{BPP}$-$\mathcal{C}$.

Fernandes Muritiba *et al.* [2] ont proposé LB^{imp}_{MC} comme une amélioration de LB_{MC}. Les auteurs disent que malgré q'une clique maximale dans G est, par construction, incluse dans une clique maximale dans $G^{'}$, calculer une clique maximale en utilisant l'heuristique de Johnson directement sur le graphe $G^{'}$ peut donner des résultats non envisageables. Pour justifier, ils disent que les objets de grosses tailles ont systématiquement des degrés élevés dans $G^{'}$, et sont d'habitude parmi les premiers à être inclus dans la clique même quand ils n'appartiennent pas à des cliques maximales de $G^{'}$. L'amélioration proposée consiste à calculer une clique maximale K par l'heuristique de Johnson sur le graphe G, à ajouter à E les arcs (i, j) tel que $(w_i + w_j > W)$ pour obtenir le graphe $G^{'}$, à élargir si possible K en sélectionnant les nœuds de $V \setminus K$ par ordre décroissant des degrés et la clique $K^{'}$ obtenue est une clique maximale dans $G^{'}$ et contient forcément K.

1.5.2.2 Des bornes à base d'un modèle de transport

La borne inférieure LB_{CP} a été introduite par Gendreau *et al.* [53]. Elle consiste à calculer une clique maximale K dans le graphe de conflits G par l'heuristique de Johnson. Ensuite, chaque objet de la clique K est placé dans un bin différent et les objets de $I \setminus K$ sont distribués, complètement ou partiellement (on autorise le découpage des objets), dans ces bins en résolvant un problème de transport. Tous les objets et portions d'objets restants sont placés dans un bin fictif (*dummy*).

La figure 1.7 montre un exemple du modèle de transport de [53, 2] pour une instance $I = \{i_1, i_2, i_3, i_4, i_5\}$ et un graphe de conflits $G = (I, E)$ où $E = \{(i_1, i_2), (i_2, i_3), (i_1, i_5), (i_3, i_4)\}$. Soit $Q = \{i_4, i_5\}$ une grande clique dans G.

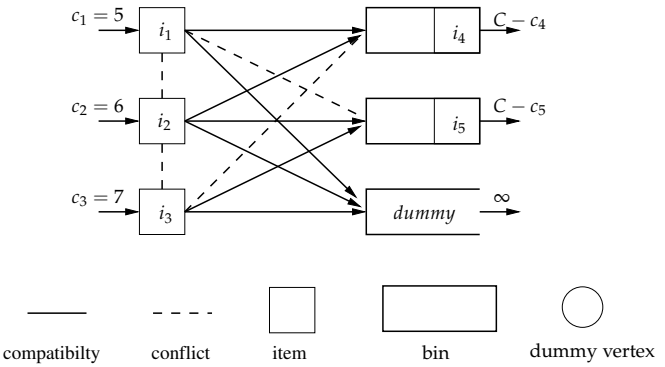


FIGURE 1.7 – *Un exemple du modèle de transport de [53, 2].*

Pour évaluer cette borne, on choisit un niveau de poids l et on divise l'ensemble I en trois sous-ensembles :

$$I_l = \{i \in I, l \leq w_i \leq W - w_h\} \quad (1.20)$$
$$I_l^{'} = \{i \in I, w_i < l\} \quad (1.21)$$
$$I_l^{''} = \{i \in I, l > W - w_h\} \quad (1.22)$$

où $h = \text{argmin}_{i \in K}\{w_i\}$. Pour un niveau de poids l, la borne inférieure LB_{CP} est la suivante

$$LB_{CP} = LB_{MC} + \left\lceil \left(\sum_{i \in I_l} x_{i0} + \sum_{i \in I_l^{''}} w_i + z_l \right) \Big/ W \right\rceil \quad (1.23)$$

où x_{i0} est la portion de l'objet i placée dans le bin fictif et

$$z_l = \max \left\{ 0, \sum_{i \in I_l^{'}} w_i - \left(\sum_{k \in K} r_k - \sum_{i \in I_l} \sum_{k \in K_l \setminus \{0\}} x_{ik} \right) \right\} \quad (1.24)$$

où r_k est la capacité résiduelle du bin k.

Puisque LB_{CP} est une borne inférieure valide pour tout seuil $l \in \{w_i, w_i \leq W - w_h\}$, on peut déduire que

$$\max_{l \in \{w_i, w_i \leq W - w_h\}} \left\{ LB_{CP} \right\} \quad (1.25)$$

est une borne inférieure valide pour le $\mathcal{BPPC}$-1$\mathcal{D}$.

Quelques temps après, la borne inférieure LB_{CP}^{imp} a été proposée par Fernandes Muritiba *et al.* [2] comme une amélioration de LB_{CP}. Cette amélioration consiste à remplacer LB_{MC} par LB_{MC}^{imp} dans (1.23).

1.5.2.3 Une borne à base d'une relaxation pondérée

Dans cette borne, Fernandes Muritiba *et al.* [2] considèrent une relaxation pondérée des contraintes (1.19) en faisant la somme des $n \times |E|$ contraintes pour obtenir un nouvel ensemble composé des n contraintes suivantes :

$$\sum_{i=1}^{n} \delta_i \times x_{ih} \leq |E| \times y_h \qquad h = 1, 2, \ldots, n \quad (1.26)$$

où δ_i représente le degré du nœud i dans $G^{'}$. Le modèle (1.4)-(1.19) et (1.26) peut être vu comme un modèle du problème dit "two-vector packing problem (TVPP)". Ce dernier est une généralisation du problème de bin packing classique où chaque objet et chaque bin est identifié par une taille et une longeur.

Par conséquence, toute borne inférieure pour le TVPP est une borne inférieure valide pour le $\mathcal{BPP}$-$\mathcal{C}$. Les auteurs utilisent la borne inférieure LB_H de Caprara et Toth [11] dont la complexité temporelle est de l'ordre de $\mathcal{O}(n^4 \times log(n))$. L'idée de LB_H est la suivante, on décompose I en trois sous-ensembles $N_1, N_2,$ and N_3, tel que $\forall (i,j) \in E, i \in N_1$ et $j \in N_2$ et N_3 contient les objets restants. Des fonctions spécifiques sont ensuite appliquées séparément sur N_1 et N_2 afin d'obtenir des bornes inférieures qui seront additionnées pour donner une borne inférieure pour l'instance initiale. Les résultats numériques de cette borne inférieure pour le TVPP ne sont battus que par une seule borne inférieure basée sur une méthode génération de colonnes et proposée par les mêmes auteurs (voir section 1.5.2.5).

La borne LB_{TVPP} consiste à relâcher le modèle de $\mathcal{BPP}$-$\mathcal{C}$ en un modèle de TVPP comme décrit là-haut, et ensuite appliquer la borne inférieure LB_H.

1.5.2.4 Une borne à base de matching

Considérons un sous-ensemble d'objets $S \subseteq I$ où un bin donné ne peut contenir que deux objets de S au maximum. Le nombre minimum de bins nécessaires pour stocker tous les éléments de S peut être calculé, en temps polynomial, en regroupant deux à deux autant d'objets que possible, puis affecter un bin à chacune de ces paires d'objets et un bin à chacun des objets restants. La valeur de la solution optimale correspond à la cardinalité d'un matching maximum M sur le graphe $\widehat{G_S}$ induit par S plus le nombre d'objets restants. De cette façon, on obtient la borne inférieure suivante

$$LB_{match} = |M| + (|S| - 2|M|) = |S| - |M|. \tag{1.27}$$

Le choix du sous-sensemble S n'est pas trivial. Dans un premier temps, il faut que S soit maximal au sens de l'inclusion par rapport à la propriété qui dit que deux objets au plus peuvent être mis dans un même bin tandis qu'il existe au moins trois objets qui peuvent être placés dans un même bin pour chaque sous-ensemble $S \cup \{j\}, j \in I \setminus S$. On ne cherche pas le sous-ensemble S de cardinalité maximale parmi tous ceux qui respecte cette propriété. En effet, les résultats expérimentaux prouvent que la taille de S ne garanti pas une meilleure valeur pour L_{match}.

Plusieurs sous-ensembles S sont calculés par un algorithme glouton. La procédure essaie de mettre dans S des objets de grande taille ($w_i > W/2$) et des objets appartenant à une clique maximale K. Pour ce faire, un paramètre β est défini ($0 \leq \beta \leq W/2$) de telle sorte qu'il existe au moins un objet de taille égale à β. Ensuite, pour chaque valeur de β, un sous-ensemble S_β est initialisé par des objets de K dont la taille est supérieure ou égale à β et rempli ensuite par des objets de $I \setminus S_\beta$ qui sont en conflit avec tous les couples d'objets de S_β. Les objets sont triés par ordre

décroissant de leur taille. Pour chaque sous-ensemble, l'algorithme 3 évalue la valeur de la borne inférieure LB_{match} et retourne la valeur maximale.

Algorithme 3 : Borne inférieure à base de matching pour le $\mathcal{BPPC}$-1$\mathcal{D}$

```
1  LB_match = 0;
2  Calculer une clique maximale K;
3  Trier les objets par ordre décroissant de leur taille;
4  pour chaque β tel que 0 ≤ β ≤ W/2 et ∃i ∈ I où w_i = β faire
5      S_β = {i ∈ K : w_i ≥ β};
6      pour chaque i ∈ I \ S_β faire
7          si S_β ∪ {i} ne contient aucun sous-ensemble de trois objets pouvant être
           placés dans le même bin alors
8              S_β = S_β ∪ {i}
9      si |S_β| > LB_match alors
10         Calculer un matching maximum M_β sur S_β par l'algorithme d'Edmond;
11         si LB_match < |S_β| − |M_β| alors
12             LB_match = |S_β| − |M_β|;
13 retourner LB_match;
```

L'instruction de la ligne 7 utilise la procédure CHECK COMPATIBILITY de Caprara and Toth (2001) qui nécessite un temps de $O(|S_\beta| log(S_\beta))$. A la ligne 10, un matching maximum est calculé par l'implémentation en $O(n^3)$ de l'algorithme d'Edmond (1965) proposée par Gabow (1976). Le nombre de valeurs distinctes de β étant borné par n, la complexité de l'algoritme [1] est en $O(n^4)$.

1.5.2.5 Une borne à base de génération de colonnes

Afin d'obtenir une meilleure borne inférieure, Fernandes Muritiba *et al.* [2] modélisent le $\mathcal{BPPC}$-1$\mathcal{D}$ sous forme d'un problème de *couverture d'ensemble* (SC) connu par son efficacité pour résoudre le problème de bin packing (voir [114, 93]) et le problème de *couverture de sommet* (VCP) (voir [88, 84]).

Ils utilisent le modèle (1.10)-(1.12) de Gilmore et Gomory [54, 55].

Pour résoudre le problème maître, ils l'initialisent avec un sous-ensembles de patterns $P^{'} \in P$ et le résolvent à l'optimalité. Ils obtiennent alors les valeurs π_i^*, $i \in I$, des variables duales associées aux contraintes (1.11). Pour détecter les contraintes duales violées, en d'autres termes les variables (patterns) à ajouter au problème maître, ils résolvent le sous-problème de pricing suivant :

$$\max \sum_{i=1}^{n} \pi_i^* \times \theta_i \tag{1.28}$$

$$\sum_{i=1}^{n} w_i \times \theta_i \leq W \tag{1.29}$$

$$\theta_i + \theta_j \leq 1 \qquad (i,j) \in E \tag{1.30}$$

$$\theta_i \in \{0,1\} \qquad i = 1,2,\ldots,n \tag{1.31}$$

Le modèle (1.28)-(1.31) peut être considéré comme un problème de sac-à-dos binaire avec conflits (KPC), avec un vecteur de profits π^* et des contraintes de conflits imposées par les contraintes (1.30).

Pour générer des colonnes, ils essaient dans un premier temps de résoudre le pricing par une heuristique gloutonne ; et si cette dernière ne réussit pas, le modèle (1.28)-(1.31) est résolu par CPLEX. L'idée de l'heuristique gloutonne est simple : elle considère les objets selon un certain ordre et les insère dans le sac un par un tant qu'il y a de la place. Les objets sont triés par ordre décroissant de $\pi_i^*/w_i^{'}$ où

$$w_i^{'} = \alpha \times \frac{w_i}{\overline{w}} + (1-\alpha) \times \frac{\delta_i}{\overline{\delta}} \tag{1.32}$$

et faisant varier α dans $\{0.0, 0.1, 0.2, \ldots, 1.0\}$ (voir section 1.5.3), ce qui donne un total de 11 différents ordres. Les colonnes à coût réduit obtenues sont ensuite ajoutées à l'ensemble $P^{'}$.

Le modèle (1.28)-(1.31) a déjà été utilisé par Hifi et Micharfy [62] qui ont étudié le problème de sac-à-dos disjonctif. Ils ont proposé une procédure de réduction, trois algorithmes exacts ainsi que des bornes inférieures et supérieures. Pferschy et Schauer [99] ont aussi traité ce problème en se basant sur le même modèle. Ils ont proposé un algorithme de programmation dynamique pour le cas des graphes de type arbre et les graphes dont la largeur d'arbre est bornée par une constante (*i.e.* graphe série-parallèle). Récemment, Sadykov et Vanderbeck [102] ont considéré aussi le même problème, même cette fois ci sur des graphes d'intervalles et ont proposé un algorithme de programmation dynamique de complexité pseudo-polynomiale.

1.5.3 Méthodes heuristiques

Nombreux sont les algorithmes d'approximation qui ont été proposés pour le $\mathcal{BPP}$-$\mathcal{C}$ [64, 40, 63, 96, 87, 41]. Ces algorithmes nécessitent des instances de $\mathcal{BPP}$-$\mathcal{C}$ dont le graphe de conflits appartient à des familles de graphes spécifiques (*e.g.* graphes planaires, graphes parfaits). Ces travaux n'entrent pas dans le cadre de la thèse, nous nous concentrons plutôt sur les heuristiques.

Dans décrivons dans cette section les principales méthodes heuristiques proposées pour le $\mathcal{BPP}$-$\mathcal{C}$. Les six premiers algorithmes ont été proposés par Gendreau *et al.* [53] pour calculer des bornes supérieures pour le $\mathcal{BPPC}$-1$\mathcal{D}$. La première méthode (H_1)

est une simple adaption d'une heuristique de bin packing largement utilisée et proposée par Coffman *et al.* [23]. Les trois suivantes (*H_2*, *H_3*, *H_4*) sont basées sur des techniques de coloration de graphe. Les deux dernières (*H_5*, *H_6*) sont basées essentiellement sur le calcul de cliques sur le graphe de conflit et le graphe de compatibilité. Nous présentons ensuite deux méthodes, H_6^{imp} et *U_AF(α)*, proposées par Fernandes Muritiba *et al.* [2] pour résoudre le $\mathcal{BPPC}$-1$\mathcal{D}$.

H_1 Cette heuristique est une adaptation directe de l'heuristique *first fit decreasing (FFD)* de Coffman *et al.* [23]. Dans un premier temps, l'ensemble des objets est trié par ordre décroissant de taille. Suivant cet ordre, chaque objet est ensuite placé dans le premier bin qui peut l'acceuillir et dans lequel il n'existe pas d'objets en conflit avec lui. Dans le cas où tous les objets sont compatibles entre eux, la complexité temporelle de FFD est $\mathcal{O}(n \times log(n))$. Le nombre de conflits étant de l'ordre de $\mathcal{O}(n^2)$, la complexité de *H_1* l'est aussi.

H_2 Cette heuristique, similaire à celle de Jansen et Öhring [64], utilise l'heuristique de coloration de graphe (DSatur) pour diviser l'ensemble d'objets en sous-ensembles disjoints d'objets compatibles entre eux. Ensuite, chaque sous-ensemble est résolu par FFD comme étant une instance de $\mathcal{BPP}$-1$\mathcal{D}$. La complexité temporelle de *H_2* est de l'ordre de $\mathcal{O}(n^2 \times log(n))$ étant donné que la procédure de coloration est de cette complexité.

H_3 Cette heuristique est similaire à *H_2*. La seule différence consiste à traiter à part un ensemble Z d'objets ayant un degré de conflit $d_c \in \{0, 1\}$. L'heuristique *H_2* est ensuite appliquée sur l'ensemble $I \setminus Z$. L'ensemble Z est enfin placé dans les bins existants, dans de nouveaux bins si nécessaire, en utilisant l'heuristique *H_1*. La complexité temporelle de *H_3* est égale à celle de *H_2*.

H_4 Cette heuristique commence par colorer les sommets de G par DSatur (voir section 1.3.2.2.1). Ensuite elle extrait de I le plus grand sous-ensemble P de sommets ayant la même couleur. Si la cardinalité de cet ensemble est égale à 1, elle place chaque objet seul dans un bin et aboutit à une solution de n bins, sinon, elle résout P comme une instance de $\mathcal{BPP}$-1$\mathcal{D}$ à l'aide de l'algorithme FFD (voir section 1.4.4.1). Elle identifie ensuite l'objet $h \in P$ ayant la plus petite taille et définit l'ensemble Q de tous les objets appartenant aux bins, sauf celui le plus rempli, dont la capacité résiduelle est plus grande que w_h. Elle met à jour $I = (I \setminus P) \cup Q$, si $I = \emptyset$, sinon elle réitère tout depuis le début. La complexité temporelle de DSatur est de l'ordre de $\mathcal{O}(n^2 \times log(n))$ et le l'heuristique est itérée au plus n fois. La complexité temporelle de *H_4* est donc de l'ordre de $\mathcal{O}(n^3 \times log(n))$.

H_5 Dans un premier temps, cette heuristique calcule une grande clique D dans le graphe de compatibilité $\overline{G}$. Si cette clique est de taille 1, elle place chaque objet de I dans un bin tout seul et on aboutit à une solution de n bins, sinon, elle résout D comme étant une instance de $\mathcal{BPP}$-1$\mathcal{D}$ par FFD. Elle identifie ensuite l'objet $h \in P$ ayant la plus petite taille et définie l'ensemble Q de tous les objets appartenant aux

bins, sauf celui le plus rempli, dont la capacité résiduelle est plus grande que w_h. Elle met à jour $I = (I \setminus P) \cup Q$, si $I \neq \emptyset$ réitérer tout dès le début. La complexité temporelle du calcul de la clique ainsi que FFD est de l'ordre de $\mathcal{O}(n^2 \times log(n))$ et le l'heuristique est itérée au plus n fois. La complexité temporelle de *H_5* est donc de l'ordre de $\mathcal{O}(n^3 \times log(n))$.

H_6 Cette heuristique est divisé en 3 étapes :
étape 1 : extraire de I un sous-ensemble H d'objets compatibles avec tous les autres objets de I, $(I = I \setminus H)$.
étape 2 : extraire de I les objets appartenant à une grande clique D dans le graphe de conflit G, $(I = I \setminus D)$. Si $I = \emptyset$ l'heuristique s'arrête et donne une solution de D bins.
étape 3 : pour chaque objet $i \in D$, calculer une grande clique D_i contenant i dans le graphe de compatibilité $\overline{G}_i$ induit par $I \cup \{i\}$. Résoudre ensuite chaque clique D_i comme étant une instance de $\mathcal{BPP}$-1$\mathcal{D}$, extraire de I tous les objets placés dans le même bin que i et mettre à jour $D = D \setminus \{i\}$. Si $I = \emptyset$ aller à l'étape 4. Si $D = \emptyset$ aller à l'étape 2, sinon aller à l'étape 3.
étape 4 : distribuer les objets de H sur les bins existants et sur des nouveaux bins si nécessaire par FFD.

La complexité temporelle de cette heuristique est de l'ordre de $\mathcal{O}(n^2 \times log(n))$.

H_6^{imp} Fernandes Muritiba *et al.* [2] ont proposé cette heuristique comme une amélioration de *H_6^{imp}*. Cette amélioration consiste à utiliser la méthode présentée dans la section 1.5.2.1 pour calculer la clique D (étape 2 dans *H_6*) ; et au lieu de résoudre pour chaque $i \in D$ la clique D_i comme étant une instance de $\mathcal{BPP}$-1$\mathcal{D}$, ils remplissent le bin contenant i par les objets de I triés par ordre décroissant des tailles. L'étape 4 reste la même. Cette heuristique a la même complexité temporelle que *H_6*.

U_AF(α) Fernandes Muritiba *et al.* [2] ont adapté les heuristiques classiques de bin packing proposées par Johnson en 1974 [69]. Plus précisément, ils proposent une borne supérieure *U_AF(α)* où le terme *AF* signifie "any fit" et qui peut être remplacé par toute fonction de type fit, *i.e.* "first fit", "best fit" et "worst fit". Ces heuristiques dépendent d'un paramètre $0 \leq \alpha \leq 1$ et qui est utilisé dans la fonction de pondération suivante

$$w_i^{'} = \alpha \times \frac{w_i}{\overline{w}} + (1 - \alpha) \times \frac{\delta_i}{\overline{\delta}} \tag{1.33}$$

qui est égale à la nouvelle taille d'un objet i, où $\overline{w}$ et $\overline{\delta}$ sont la taille moyenne et le degré moyen respectivement. Ils obtiennent ainsi 11 ordres de traitement sur les objets en les triant par ordre décroissant des $w_i^{'}$ et en faisant varier α entre 0 et 1 avec un pas incrémental de 0.1. Sur chacun de ces 11 ordres, il appliquent les 3 variantes de *U_AF* donnant ainsi un total de 33 solutions.

1.5.4 Métaheuristiques

Fernandes Muritiba *et al.* [2] ont proposé la seule métaheuristique pour le $\mathcal{BPPC}$-$1\mathcal{D}$. Ils initialisent l'algorithme avec un bassin de solutions contenant toutes les solutions qu'ils génèrent par les heuristiques *U_AF(α)* et H_6^{imp} décrites ci-dessus. Ils obtiennent des nouvelles solutions en appliquant un opérateur de croisement sur les solutions du bassin. Chacune des solutions obtenues ainsi est améliorée par une recherche tabou, qui se déplace dans un espace de recherche contenant des solutions faisables partielles, *i.e.* certains objets peuvent ne pas être affectés à des bins. Cette idée, basée sur le voisinage défini par Morgenstern [95] pour le VCP, consiste à améliorer une solution faisable partielle de valeur k afin d'obtenir une solution faisable complète de la même valeur. Le paramètre k représente une valeur cible égale à un nombre de bins qu'on désire utiliser pour placer tous les objets. Les auteurs ont définis une fonction objectif basée sur la somme pondérée des tailles des objets, et ils utilisent plusieurs types de mouvements afin de se déplacer dans l'espace de recherche. Afin de diversifier leur recherche, ils mettent en place une procédure de diversification basée sur un opérateur de croisement.

1.5.5 Méthodes exactes

Fernandes Muritiba *et al.* [2] ont proposé la première méthode exacte dédiée au $\mathcal{BPPC}$-$1\mathcal{D}$. Il s'agit d'une méthode arborescente de type branch & price. Les nœuds sont testés selon la stratégie "parcours en profondeur d'abord". La séparation se fait sur les variables ayant la partie fractionnaire la plus importante en fixant une variable à la fois à 1.

Chacun des nœuds explorés de l'arbre est évalué par la méthode de génération de colonnes décrite dans la section 1.5.2.5. Quand une variable z_b est fixée à 1, les contraintes (1.11) liées à un objet $i \in b$ peuvent être éliminées des sous-problèmes descendants car elles sont automatiquement satisfaites. Inversement, quand un variable z_b est fixée à 0, il faut éviter que la génération de colonnes pour les sous-problèmes descendants ne génèrent de nouveau la colonne associée au bin b. Cette interdiction est modélisée sous la forme de la contrainte suivante

$$\sum_{i \in b} \theta_i \leq |b| - 1 \tag{1.34}$$

qui impose que $|b| - 1$ objets soient au plus choisis parmi ceux appartenants à b.

Quelques temps après, Sadykov et Vanderbeck [102] ont amélioré cette méthode en proposant un algorithme de programmation dynamique pseudo-polynomial pour résoudre le sous-problème de pricing (1.28)-(1.31). La séparation est celle du framework BapCode [102].

1.6 Le problème de bin packing avec objets fragiles

Dans cette section, nous définissons le problème de bin packing avec objets fragiles. Nous décrivons aussi quelques méthodes de résolution dédiées à ce problème.

1.6.1 Définition formelle

Le problème de *bin packing avec objets fragiles* ($\mathcal{BPP}$-$\mathcal{FO}$ pour ***B**in **P**acking **P**roblem with **F**ragile **O**bjects*) consiste à minimiser le nombre de bins nécessaires pour placer un ensemble $I = \{1, \ldots, n\}$ de n objets. Chaque objet $j \in I$ est caractérisé par une taille w_i et une fragilité f_i. Une solution est considérée faisable si la somme des tailles des objets placés dans un bin ne dépasse pas la fragilité de chacun de ces objets. Plus formellement, soit $J(i)$ un ensemble d'objets placés dans un bin i, la condition suivante doit être vérifiée pour tout bin i pour que la solution soit faisable

$$\sum_{j \in J(i)} w_j \leq \min_{j \in J(i)} \{f_j\}. \tag{1.35}$$

Par conséquent, un "bin" n'a pas de taille fixe connue auparavant et que sa taille sera décidée en fonction de son contenu dans la solution.

Le $\mathcal{BPP}$-$\mathcal{FO}$ introduit par Bansal *et al.* [3] est un problème $\mathcal{NP}$-complet, puisqu'il généralise le problème de bin packing classique ($\mathcal{BPP}$). Tout instance de $\mathcal{BPP}$ est une instance de $\mathcal{BPP}$-$\mathcal{FO}$ où les objets ont tous une même fragilité égale à une constante C.

La figure 1.8 montre un exemple proposé dans [15] d'une instance de $\mathcal{BPP}$-$\mathcal{FO}$ et deux solutions possibles pour cette instance, une optimale et une non optimale. Chaque objet est représenté par un couple (w, f) où w est la taille et f est la fragilité. L'instance de la figure 1.8 est composé de 5 objets : $(1,4)$, $(2,6)$, $(2,6)$, $(2,6)$, $(3,6)$. La fragilité d'un objet est représenté par un rectangle à bordure pointillée et sa taille par un rectangle coloré. La première solution est une solution faisable nécessitant 3 bins et la deuxième est une solution optimale de 2 bins.

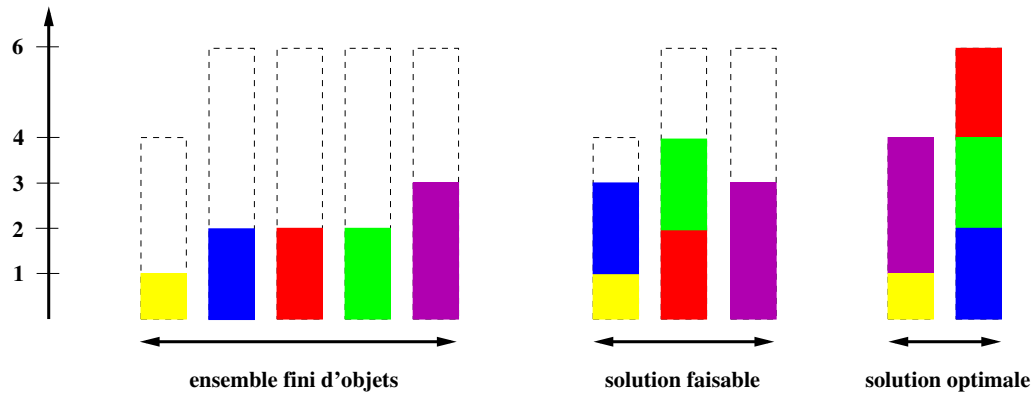


FIGURE 1.8 – *Une instance de $\mathcal{BPP}$-$\mathcal{FO}$ et deux solutions possibles associées à cette instance.*

La littérature sur le $\mathcal{BPP}$-$\mathcal{FO}$ en cours de croissance et rares sont les papiers qui traitent ce sujet. Bansal *et al.* [4] ont proposé des schémas d'approximation et des analyses probabilistes. Ils ont considéré des approximations basées sur le nombre de bins et sur la fragilité d'un bin. Ils ont présenté des résultats pour le $\mathcal{BPP}$-$\mathcal{FO}$ et pour un cas spécial du $\mathcal{BPP}$-$\mathcal{FO}$, dit *problème d'allocation de fréquence*, sans lequel les tailles et les fragilités sont strictement corrélées. Chan *et al.* [15] ont considéré une version on-line du BPP-FO, dans laquelle les objets arrivent les uns après les autres et chaque objet placé dans un bin ne peut plus être retiré et placé dans un autre bin. Ils ont étudié les cas où rapport entre la fragilité maximum et la fragilité minimum est borné ou bien non borné, et ils ont présenté, pour chacun des deux cas, des algorithmes avec des rapports asymptotiques compétitifs.

1.6.2 Relations avec d'autres problèmes

Le $\mathcal{BPP}$-$\mathcal{FO}$ est une généralisation du problème d'*allocation de fréquences (FAP)* dans lequel n utilisateurs communiquent avec une station de base. Si l'utilisateur i transmet une puissance p_i alors le signal reçu par la station de base est $s_i = p_i g_i$ où g_i est le gain du canal pour l'utilisateur i. Soit U l'ensemble d'utilisateurs transmettant vers la station de base dans le même canal de fréquence que l'utilisateur i, alors la transmission de l'utilisateur i est réussie si et seulement si

$$\frac{s_i}{N_0 + \sum_{j \in U, j \neq i} s_j} \geq \beta \tag{1.36}$$

où β est le rapport signal sur bruit minimal (SNR, *Signal to Noise Ratio*) nécessaire pour établir une communication, la constante N_0 est la puissance du bruit. Typiquement, β est une constante inférieure à 1. La constante N_0 est utilisée pour modéliser l'interférence due au bruit thermique et l'interférence due à la transmission des canaux voisins dans un réseau cellulaire multiple. Le FAP est un cas spécial du $\mathcal{BPP}$-$\mathcal{FO}$. La contrainte (1.36) peut être aussi écrite sous la forme suivante

$$(1+\beta)s_i \geq \beta(N_0 + \sum_{j \in U} s_j). \tag{1.37}$$

Si on considère les canaux de fréquence comme étant les bins et les utilisateurs comme étant les objets où chaque objet i a une largeur s_i et une fragilité $s_i(\beta+1)/\beta - N_0$, dans une instance de $\mathcal{BPP}$-$\mathcal{FO}$, la condition de faisabilité pour un bin U contenant un objet i est la suivante

$$\sum_{j \in U} s_j \leq s_i(\beta+1)/\beta - N_0 \tag{1.38}$$

qui est elle-même équivalente à

$$(1+\beta)s_i \geq \beta(N_0 + \sum_{j \in U} s_j). \tag{1.39}$$

Les fragilités modélisent donc la contrainte SNR. Affecter un objet à un bin dans un contexte de bin packing revient alors à affecter les utilisateurs à des canaux de fréquences dans le FAP.

Ceselli et Righini [13] ont étudié une variante du problème de bin packing dite *Ordered Open-End Bin Packing Problem* (OOEBPP) qui a été proposée par Leung *et al.* [79] en 2001. Dans le OOEBPP, la donnée consiste en une instance de $\mathcal{BPP}$ et d'un ordre total sur les objets. La seule différence avec le $\mathcal{BPP}$ réside dans le fait qu'on est autorisé à placer un objet dans un bin si la capacité résiduelle n'est suffisante de telle sorte que la somme des tailles des objets placés un bin dépasse la taille du bin d'une unité au maximum. Nous expliquons en détail dans la section 5.2.2 la relation entre ces deux problèmes.

1.6.3 Algorithmes d'approximation

Bansal *et al.* [4] ont proposé un algorithme 2-approximatif avec une complexité temporelle de l'ordre $\mathcal{O}(n \times log(n))$, et ils ont prouvé que :

1. cet algorithme produit une affectation fractionnaire nécessitant le moins de bins possible,
2. s'il existe une solution faisable de k bins, il existe alors une affectation fractionnaire qui utilise k bins au plus,
3. s'il existe une affectation fractionnaire de k bins, il existe alors une solution faisable qui utilise $2 \times k - 1$ bins au plus.

Ils ont prouvé aussi qu'il n'existe pas d'algorithme qui peut atteindre un rapport d'approximation plus petit que $3/2$ en terme de nombre de bins, à moins que $\mathcal{P} = \mathcal{NP}$. Ils utilisent ce résultat comme une borne inférieure sur le nombre de bins pour une instance de $\mathcal{BPP}$-$\mathcal{FO}$ quelconque. D'autres résultats théoriques sur l'approximabilité du $\mathcal{BPP}$-$\mathcal{FO}$ en se basant sur la fragilité des objets ont été proposés dans [4].

Chan *et al.* [15]) ont considéré la variante on-line du $\mathcal{BPP}$-$\mathcal{FO}$ et ils ont prouvé que le rapport asymptotique est au moins de 2. Ils ont considéré aussi le cas où le rapport de la fragilité maximum et la fragilité minimum est borné par une constante, et ils ont présenté pour ce cas une classe d'algorithmes on-line avec un rapport asymptotique inférieure ou égal à $1/4 + (3 \times r)/2$ pour tout $r > 1$.

1.6.4 Bornes inférieures fractionnaires

Une simple borne inférieure peut être obtenue en relâchant la contrainte de fragilité. Une instance de $\mathcal{BPP}$-$\mathcal{FO}$ est ainsi transformée en une instance de $\mathcal{BPP}$. Pour ce faire, on calcule la fragilité maximale $f_{\max} = max_{i=1,\ldots,n}\{f_i\}$, et on met à jour la fragilité de tous les objets comme suit : $f_i = f_{\max}, \forall i \in I$. Toute borne inférieure valide pour le $\mathcal{BPP}$ est ainsi une borne inférieure valide pour le $\mathcal{BPP}$-$\mathcal{FO}$. On définit alors :

$$L_0 = \left\lceil \sum_{i=1}^{n} \frac{w_i}{f_{\max}} \right\rceil . \tag{1.40}$$

Le résultat précédent est amélioré par Chan et al. [15] qui montrent que

$$L_1 = \left\lceil \sum_{i=1}^{n} \frac{w_i}{f_i} \right\rceil \tag{1.41}$$

est une borne inférieure valide.

Le résultat peut être encore amélioré en utilisant la *borne inférieure fractionnaire*, présentée par Bansal *et al.* [4] et décrite dans l'algorithme 4. L'idée est de placer des objets dans des bins selon un ordre croissant de fragilité. Quand un objet i ne peut pas être placé complètement dans un bin, la portion de i qui dépasse les limites du bin est placé dans un nouveau bin de fragilité f_i.

Algorithme 4 : Borne inférieure fractionnaire (L_2)

entrée : n objets triés par ordre croissant des fragilités (ordre décroissant des tailles en cas d'égalité)
sortie : borne inférieure L_2

```
1  k = 1 ;
2  f_res = f_1 − w_1 ;
3  for i = 2 to n do
4      if w_i ≤ f_res then
5          f_res = f_res − w_i ;
6      else
7          w_res = w_i − f_res ;
8          k = k + 1 ;
9          f_res = f_i − w_res ;
10 retourner k;
```

Dans la figure 1.9, nous donnons un exemple illustratif de la borne L_2 appliquée sur l'instance de la figure 1.8.

FIGURE 1.9 – *La borne L_2 appliquée sur l'instance de $\mathcal{BPP}$-$\mathcal{FO}$ de la figure 1.8.*

1.7 Conclusion

La littérature sur les problèmes de bin packing est très riche. Dans ce chapitre nous avons décrit une partie de ces travaux, surtout les plus récents parmi ceux qui

nous intéressent pour la suite de ce manuscrit. Nous avons remarqué que peu de travaux portent sur la variante avec conflits entre les objets ainsi que celle avec des objets fragiles, que ce soit au niveau des méthodes de résolution, des évaluations par défaut ou même de jeux d'essai dans le cas de la deuxième variante. Cependant ces variantes impliquent des spécificités fréquemment rencontrées dans les problèmes réels, qui mérite donc d'être étudiées.

Chapitre 2

Bornes inférieures pour le bin packing avec conflits

Le travail de ce chapitre a fait l'objet d'une publication dans "European Journal of Operational Research (EJOR)" [74] ainsi que deux présentations orales dans des conférences [71, 72].

2.1 Introduction

Ce chapitre est consacré au problème de bin packing uni- et bi-dimensionnel avec conflits. La principale contribution est une nouvelle borne inférieure pour ces problèmes.

Il existe deux types de bornes inférieures dans la littérature (voir section 1.4.3) : celles basées sur des algorithmes polynomiaux, et celles basées sur la résolution d'un programme linéaire à l'aide de techniques de génération de colonnes [2]. Les résultats de cette dernière méthode sont de loin les meilleurs, mais la résolution du programme linéaire peut demander un temps de calcul très important. Par ailleurs, cette borne est dédiée à la variante uni-dimensionnelle et ne pas être généralisée facilement pour le cas bi-dimensionnel. Il est donc nécessaire d'améliorer les bornes polynomiales et de proposer de nouvelles bornes multi-dimensionnelles efficaces et rapides.

Dans la suite, nous proposons des méthodes basées sur la généralisation du concept de "fonctions dual-réalisables (DFF)" et "fonctions dual-réalisables dépendantes de la donnée (DDFF)". Ces deux concepts ont été largement utilisés dans la littérature pour améliorer la résolution de plusieurs problèmes de cutting et packing. Nous présentons une nouvelle famille de DDFF dédiées au $\mathcal{BPP}$ qui peuvent améliorer les résultats de la littérature pour le ce problème. Nous généralisons ensuite le concept de DDFF pour le cas de $\mathcal{BPP}$-$\mathcal{C}$ en proposant le nouveau concept de "fonctions dual-réalisables généralisées dépendantes de la donnée (GDDFF) ". Les deux GDDFF que nous proposons prennent en considération le graphe de conflits en utilisant des techniques de "triangulation de graphes" et de "décomposition arborescente" (voir section 1.3.3).

Lorsque la taille des problèmes traités est grande, il devient difficile de calculer des bornes inférieures d'une façon rapide. Pour éviter d'avoir à traiter des instances de taille trop importante, il est intéressant de les réduire à l'aide de procédures de

réduction qui peuvent être utilisées en prétraitement avant d'appliquer une méthode de résolution. Pour cette raison, nous proposons des procédures de prétraitement qui peuvent être utilisées pour éliminer certains objets et modifier la taille d'autres sans modifier la valeur de la solution optimale du problème. Nous montrons la réduction en terme de taille d'instance que l'on peut obtenir en utilisant ces procédures.

Nous reproduisons les résultats expérimentaux obtenus par nos procédures de réduction, ainsi que par nos bornes inférieures. Nous les avons testées sur des jeux d'essai issus de la littérature [2], ce qui nous permet de nous comparer avec les meilleurs résultats connus [2]. Nos procédures de réduction sont très efficaces pour plusieurs classes d'instances : un grand nombre d'articles sont supprimés pour de multiples jeux d'essai. Les résultats expérimentaux confirment l'efficacité de nos bornes qui améliorent strictement les meilleurs résultats pour une grande partie du jeu d'essai.

Dans la section 2.2 nous proposons trois nouvelles procédures de réduction qui seront utilisées afin de réduire la taille d'une instance de $\mathcal{BPP}$-$\mathcal{C}$. Dans la section 2.3, nous rappelons le concept de *fonctions dual-réalisables dépendantes de la donnée* (DDFF) et nous proposons une nouvelle famille de fonctions de type DDFF. Dans la section 2.3.3 nous proposons le nouveau concept de *fonctions dual-réalisables généralisées dépendantes de la donnée*. La section 2.4 est consacrée à nos nouvelles bornes inférieures. Dans la section 2.5, nous comparons nos procédures de réduction et nos bornes inférieures avec les méthodes de la littérature. Les résultats sont strictement améliorés pour plusieurs instances.

2.2 Prétraitements

Cette section est consacrée à des procédures de prétraitement pour le $\mathcal{BPP}$-$\mathcal{C}$. Nous présentons quatre procédures. La première permet d'éliminer les objets en conflit avec tous autres. La deuxième est une adaptation d'un prétraitement de la littérature. La troisième et la quatrième consistent à éliminer les objets de l'instance que l'on sait placer à l'optimalité.

Definition 2.2.1 *Soit $D = \langle I, B, G\rangle$ une instance de $\mathcal{BPP}$-$\mathcal{C}$. Une fonction "pr" est une procédure de prétraitement associée à D si et seulement si l'instance $D' = \langle I', B, G\rangle$, obtenue en appliquant pr sur tous les objets de l'instance D, est telle que $\mathcal{OPT}(D) = \mathcal{OPT}(D')$.*

2.2.1 Supprimer un objet en conflit avec tout autre objet

La première procédure de prétraitement consiste à éliminer tout objet étant en conflit avec tous les autres objets de l'instance.

Proposition 2.2.1 *Soit $D = \langle I, B, G\rangle$ une instance de $\mathcal{BPP}$-$\mathcal{C}$ et $i \in I$ un objet. La fonction*

$$pr_1 : I \to I^{'}$$
$$i \to i^{'} \text{ où } \begin{cases} w_i^{'} = W \text{ et } h_i^{'} = H & si\ (i,j) \in E, \forall j \in I \setminus i \\ w_i^{'} = w_i \text{ et } h_i^{'} = h_i & sinon \end{cases}$$

est une procédure de prétraitement associée à l'instance D.

2.2.2 Adaptation du prétraitement de Boschetti et Mingozzi

La procédure de prétraitement proposée par Boschetti et Mingozzi [7] et décrite dans le chapitre 1 peut être généralisée pour le cas de $\mathcal{BPP}$-$\mathcal{C}$. Cette procédure de prétraitement repose sur le calcul des espaces perdus quand on place un objet dans un bin donné. Cet espace perdu, calculé initialement en résolvant des problèmes de sac-à-dos, est évalué d'une façon plus complexe dans le cas de $\mathcal{BPP}$-$\mathcal{C}$. Le problème de sac-à-dos est donc remplacé par un problème de sac-à-dos défini disjonctif comme suit :

$$\mathcal{KP}\text{-}1\mathcal{D}(C, J, c, \alpha) = \max\left\{\sum_{j\in J}\alpha_j x_j \ /\ \sum_{j\in J} c_j x_j \leq C,\ x_i \in \{0,1\}\right\} \tag{2.1}$$

où J représente un ensemble d'objets, c représente une fonction qui associe une taille c_i à chaque objet $j \in J$, α représente une fonction qui associe un coût α_i à chaque objet $j \in J$, et soit C la taille du sac. La solution optimale de $\mathcal{KP}$-$1\mathcal{D}$ est un ensemble de coût maximum d'objets $j \in J$ qui peuvent être placés ensemble sans que la somme de leur taille ne dépasse celle du sac.

En plaçant un objet $i \in I$ dans un bin, la quantité d'espace perdue à droite (resp. au dessus) de i est évalué par un problème de sac-à-dos $\mathcal{KP}$-$1\mathcal{D}(W, I_i, w, \alpha)$ (resp. $\mathcal{KP}$-$1\mathcal{D}(H, I_i, h, \alpha)$) où $I_i \subseteq I$ est l'ensemble d'objets $j \in I$ compatibles avec i tel que $w_i + w_j \leq W$ (resp. $h_i + h_j \leq H$). L'ensemble des paramètres α_i est fixé à $1, \forall i \in I$. En Pratique, $\mathcal{KP}$-$1\mathcal{D}(W, I_i, w, \alpha)$ (resp. $\mathcal{KP}$-$1\mathcal{D}(H, I_w, h, \alpha)$) représente la largeur (resp. hauteur) maximale d'objets qui peuvent être placés à coté (resp. au-dessus) de w_i (resp. h_i).

Proposition 2.2.2 *[7] Soit $D = \langle I, B, G\rangle$ une instance de $\mathcal{BPP}$-$\mathcal{C}$ et $i \in I$ un objet. La fonction*

$$pr_2 : I \to I^{'}$$
$$i \to i^{'} \text{ où } \begin{cases} w_i' = W - \mathcal{KP}\text{-}1\mathcal{D}(W - w_i, I_i, w, \alpha) \text{ et } h_i' = h_i & si\ i{=}j \\ h_i' = H - \mathcal{KP}\text{-}1\mathcal{D}(H - h_i, I_i, h, \alpha) \text{ et } w_i' = w_i & sinon \end{cases}$$

est une procédure de prétraitement associée à l'instance D.

Etant donné l'instance suivante : un ensemble d'objets $I = \{i_1 = (4,6), i_2 = (5,10), i_3 = (9,2), i_4 = (7,3)\}$ et un bin $B = (10,12)$. On considère le prétraitement pr_2 et l'objet i_1. La figure 2.1 donne un exemple illustratif de la valeur de W_1^* (zone hachurée dans *a*) et H_1^* (zone hachurée dans *b*). Ces valeurs représentent les quantités à ajouter aux tailles de l'objet i_1.

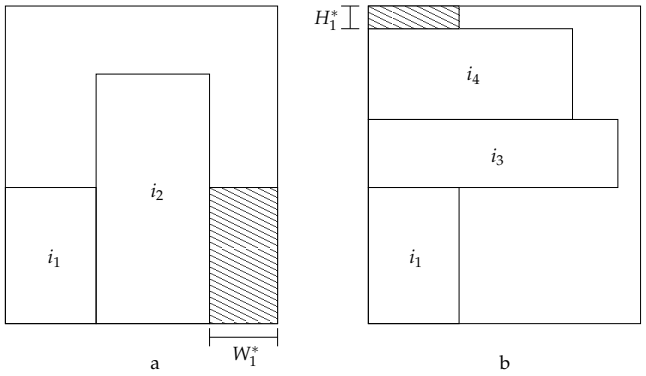


FIGURE 2.1 – *Illustration des espaces perdus* $W_1^* = W - \mathcal{KP}\text{-}1\mathcal{D}(W - w_1, I_i, w, \alpha)$ *et* $H_1^* = H - \mathcal{KP}\text{-}1\mathcal{D}(H - h_1, I_i, h, \alpha)$ *dans un bin pour un objet* i_1.

2.2.3 Résoudre l'instance à l'optimalité

L'idée de la troisième procédure pr_3 est la suivante. On calcule une grande clique Q dans le graphe de conflits G. Si on peut placer tous les objets de I dans un ensemble de $|Q|$ bins, l'instance $D = \langle I, B, G\rangle$ est donc résolue à l'optimalité.

Proposition 2.2.3 *Soit* $D = \langle I, B, G\rangle$ *une instance de* $\mathcal{BPP}$*-*$\mathcal{C}$ *et* $i \in I$ *un objet. La fonction*

$$pr_3 : I \rightarrow I'$$

$$i \rightarrow i' \text{ où } \begin{cases} w_i' = W \text{ et } h_i' = H & \text{si } i \in Q \\ w_i' = 0 \text{ et } h_i' = 0 & \text{sinon} \end{cases}$$

est une procédure de prétraitement valide associée à l'instance D *si et seulement si* $|Q|$ *bins sont suffisants pour placer tous les objets de* I.

Toute méthode de résolution dédiée au $\mathcal{BPP}$*-*$\mathcal{C}$ peut être utilisée pour tester si un placement de $|Q|$ bins est faisable.

2.2.4 Détecter les placements optimaux

La quatrième procédure pr_4 est appliquée si l'instance D n'a pas été résolue à l'optimalité par pr_3. On calcule une grande clique Q dans le graphe de conflits G. Pour chaque objet $i \in Q$, on calcule l'ensemble I_i d'objets compatibles avec i. Si $I_i = \emptyset$, on place i dans un bin seul et on l'enlève de l'instance. Sinon, on teste si on peut placer $\{i\} \cup I_i$ dans un seul bin ; si c'est le cas, on enlève $\{i\} \cup I_i$ de l'instance et on continue avec les autres objets de la clique Q.

Proposition 2.2.4 *Soit $D = \langle I, B, G\rangle$ une instance de $\mathcal{BPP}$-$\mathcal{C}$ et $i \in I$ un objet. La fonction*

$$pr_4 : I \to I^{'}$$
$$i \to i^{'} \text{ où } \begin{cases} w_i^{'} = W \text{ et } h_i^{'} = H & \text{si } i \in Q \\ w_i^{'} = 0 \text{ et } h_i^{'} = 0 & \text{si } i \in I_i \end{cases}$$

est une procédure de prétraitement valide associée à l'instance D si et seulement si un seul bin est suffisant pour placer $\{i\} \cup I_i$.

Les quatre procédures décrites dans cette section peuvent être utilisées ensemble et peuvent assurer une prétraitement de la taille de l'instance relativement grand. Elles seront exécutées avant de lancer les bornes inférieures qui seront décrites dans la suite de ce chapitre.

2.3 Fonctions dual-réalisables et extensions

Nous avons présenté dans le chapitre d'état de l'art le concept de fonction dual-réalisable (DFF) et de fonction dual-réalisable dépendante de la donnée (DDFF). Nous avons présenté aussi des méthodes pour obtenir des bornes inférieures qui utilisent des DFF et des DDFF.

Nous rappelons dans cette section la DDFF f_1 introduite par Carlier *et al.* [12] et nous montrons dans la suite comment générer de nouvelles DDFF qui généralise f_1. La particularité de ces nouvelles DDFF réside dans le fait qu'elles sont applicables dans le cas de $\mathcal{BPP}$ et $\mathcal{BPP}$-$\mathcal{C}$.

2.3.1 DFF dépendantes de la donnée (DDFF)

La fonction f_1^k de Carlier *et al.* [12] est définie pour un paramètre donné k, $1 \leq k \leq \frac{1}{2}C$ où C est la taille du bin, et une liste d'entiers $c_1, c_2, \ldots, c_n$ où c_i représente la taille de l'objet $i \in I$. Les auteurs utilisent un ensemble $J = \{i \in I : \frac{1}{2}C \geq c_i \geq k\}$ et le problème de sac-à-dos uni-dimensionnel $\mathcal{KP}$-$1\mathcal{D}(X, J, c, \alpha)$ définit dans (2.1) où J représente un ensemble d'objets, c représente une fonction qui associe une taille c_i à chaque objet $j \in J$, une fonction α qui associe un coût $\alpha_i = 1$ à chaque objet $j \in J$, et C la taille du sac. La solution optimale de ce problème de sac-à-dos est égale au nombre maximum d'objets qui peuvent être contenus ensemble dans un sac de taille X. Le problème peut être résolu en temps linéaire si les objets sont triés par ordre croissant de taille.

La fonction f_1 est définie formellement comme suit

$$f_1^k : [0, C] \to [0, \mathcal{KP}\text{-}1\mathcal{D}(C, J, c, \alpha)]$$
$$x \mapsto \begin{cases} \mathcal{KP}\text{-}1\mathcal{D}(C, J, c, \alpha) - \mathcal{KP}\text{-}1\mathcal{D}(C - x, J, c, \alpha) & \text{si } x > \frac{1}{2}C \\ 1 & \text{si } \frac{1}{2}C \geq x \geq k \\ 0 & \text{sinon.} \end{cases}$$

Théorème 2.1 *[12] La fonction f_1^k est une DDFF pour la Donnée $c_1, \ldots, c_n$ et la valeur C.*

Il est important de préciser que f_1 n'est pas une DFF car $\mathcal{KP}$-$1\mathcal{D}(C, J, c, \alpha)$ est dépendant des valeurs $c_1, \ldots, c_n$.

2.3.2 Une nouvelle famille de DDFF

Nous montrons dans cette section comment de nouvelles DDFF peuvent être dérivées à partir de la fonction f_1 de Carlier *et al.* [12]. L'idée de f_1 est de donner la valeur 1 à tout objet de taille c_i telle que $k \leq c_i \leq \frac{1}{2}C$, et les tailles des grands objets et du bin sont ensuite évaluées en fonction du nombre d'objets qui peuvent être placés ensemble dans un même bin. Une première amélioration de f_1 consiste à remplacer la valeur 1 donnée à tout objet $i \in I$ de taille c_i telle que $k \leq c_i \leq \frac{1}{2}C$ par un paramètre arbitraire $\alpha_i \in \mathbb{N}$.

Proposition 2.3.1 *Soit D une instance de bin-packing, la fonction $g_1 : [0, C] \rightarrow [0, \mathcal{KP}\text{-}1\mathcal{D}(C, J, c, \alpha)]$*

$$c_i \mapsto \begin{cases} \mathcal{KP}\text{-}1\mathcal{D}(C, J, c, \alpha) - \mathcal{KP}\text{-}1\mathcal{D}(C - x, J, c, \alpha) & \text{si } c_i > \frac{1}{2}C \\ \alpha_i & \text{sinon} \end{cases}$$

est une DDFF définie pour D.

En appliquant une DFF sur D, éliminer un objet peut réduire la valeur de la borne inférieure obtenue. Par contre, avec une DDFF, cette observation n'est plus vraie car la taille d'autres objets sera ajustée par le problème de sac-à-dos.

En appliquant g_1 sur D, les tailles des petits objets i $(c_i \leq C/2)$ sont modifiées de telles sorte que $c_i = \alpha_i$, et les tailles des objets restants sont évaluées en résolvant un problème de sac-à-dos. L'efficacité de g_1 dépend du vecteur de paramètres α. La fonction g_1 a été testée pour plusieurs valeurs de α_i, e.g. c_i, $\lfloor \frac{c_i}{k} \rfloor$ pour un k $(1 \leq k \leq C/2)$ donné, valeur aléatoire, etc. Quelque soit la valeur donnée à α_i, la fonction g_1 reste une DDFF valide, c-à-d elle respecte la définition (1.4.2). La validité est assurée par le problème de sac-à-dos en ajustant les tailles des grands objets i $(c_i > C/2)$.

Le problème de sac-à-dos utilisé est connu comme étant un problème $\mathcal{NP}$-difficile dans le cas général. Toutefois, il peut être résolu en temps pseudo-polynomial en utilisant la programmation dynamique. Quand la taille du bin est suffisamment grande, cette méthode peut entraîner un temps de calcul considérable. Dans ce cas, l'ensemble de paramètres α doit être soigneusement choisi pour permettre une résolution en temps polynomial. Cette idée a été étudiée par Carlier *et al.* [12] en choisissant $\alpha_i = 1, \forall i \in J$. La valeur optimale du problème de sac-à-dos est donc égale au nombre maximum d'objets qui peuvent être placés ensemble dans un sac de taille C. Ce problème peut être résolu en temps linéaire si les objets sont triés par ordre croissant des tailles.

Cependant, la forme générale g_1 est plus efficace que f_1 pour calculer des bornes

inférieures. Les meilleurs résultats en moyenne ont été obtenus en choisissant $\alpha_i = c_i, \ \forall i \in J$.

2.3.3 Le concept de DDFF généralisée (GDDFF)

Dans cette section, nous présentons une généralisation du concept de fonction dual-réalisable dépendante de la donnée. Cette généralisation permet à une DDFF de prendre en compte les conflits entre les objets. Nous appelons *fonctions dual-réalisables dépendantes de la donnée généralisées* (GDDFF) cette nouvelle famille de fonctions.

La motivation principale derrière cette nouvelle famille de fonctions réside dans le fait que les fonctions de type DDFF ne prennet pas en compte les relations de compatibilité entre les objets.

Nous commençons par donner une définition formelle de ce nouveau concept, et nous proposons ensuite deux fonctions de type GDDFF.

2.3.3.1 Définition formelle

Dans ce qui suit, nous considérons un ensemble d'entiers $I = \{0, 1, \ldots, n\}$, une fonction $c : i \rightarrow c_i \in \mathbb{N} \ (\forall i \in I)$ telle que $c_i \leq c_0 \ (\forall i \in I \setminus \{0\})$ et un ensemble δ de sous-ensemble de $I \setminus \{0\}$. Dans un contexte de bin packing, on désigne par $I \setminus \{0\}$ l'ensemble des objets, par le singleton $\{0\}$ le bin et par δ un ensemble de sous-ensemble d'objets de $I \setminus \{0\}$ compatibles entre eux. La taille d'un objet $i \in I \setminus \{0\}$ est donné par c_i et c_0 est la taille du bin au lieu de C.

Définition 2.1 *Soit δ un ensemble de sous-ensemble de $I \setminus \{0\}$. Une fonction dual-réalisable généralisée dépendante de la donnée (GDDFF) pour (I, c, δ) est un mapping h défini de I vers $\mathbb{N}$ tel que*

$$\forall S \in \delta, \forall S_1 \subseteq S, \sum_{i \in S_1} c_i \leq c_0 \Rightarrow \sum_{i \in S_1} h(i) \leq h(0).$$

Au contraire d'une DDFF, une GDDFF ne prend pas en considération tous les sous-ensembles de $I \setminus \{0\}$. La définition 2.1 n'est pas valide pour les sous-ensembles qui n'appartiennent pas à δ.

2.3.3.2 La première GDDFF h_1

La première GDDFF est une généralisation de la fonction g_1 définie dans la proposition 2.3.1. Elle se base sur le fait que quand un graphe de conflit G est considéré, seuls les stables de G peuvent être des solutions du problème de sac-à-dos. Quand la taille d'un objet i doit être évaluée, le problème de sac-à-dos est appliqué sur les stables maximaux contenant v_i.

2.3.3.2.1 Définition

Comme dans la cas de la fonction g_1, celle famille de fonctions utilise un ensemble arbitraire de paramètres entiers $\alpha_1 \ldots, \alpha_n$. Cette fonction prend en considération le fait que certaines combinaisons d'articles ne peuvent pas être considérées.

Proposition 2.3.2 *Soit D une instance (I, J, c, δ) où J un ensemble d'articles en conflits entre eux et δ un ensemble de sous-ensembles valides (ne contenant pas d'articles en conflits) de $I \setminus \{0\}$. La fonction h_1 :*

$$I \to \left[0, \max_{K\in\delta}\{\mathcal{KP}\text{-}1\mathcal{D}(c_0, K, c, \alpha)\}\right]$$

$$i \mapsto \begin{cases} \max\limits_{K\in\delta}\{\mathcal{KP}\text{-}1\mathcal{D}(c_0, K, c, \alpha)\} & \text{si } i = 0 \\ \max\limits_{K\in\delta}\{\mathcal{KP}\text{-}1\mathcal{D}(c_0, K, c, \alpha)\} - \max\limits_{K\in\delta/i\in K}\{\mathcal{KP}\text{-}1\mathcal{D}(c_0 - c_i, K \setminus \{i\}, c, \alpha)\} & \text{si } i \in J \\ \alpha_i & \text{sinon.} \end{cases}$$

est une GDDFF pour l'instance D.

Démonstration. Supposons que h_1 n'est pas une GDDFF. Dans ce cas, il existe un sous-ensemble K de δ et $S_1 \subseteq K$ tel que $\sum_{i\in S_1} c_i \leq c_0$ et $\sum_{i\in S_1} h_1(i) > \max\limits_{K\in\delta}\{\mathcal{KP}\text{-}1\mathcal{D}(c_0, K, c, \alpha)\}$ vu que $h_1(0) = \max\limits_{K\in\delta}\{\mathcal{KP}\text{-}1\mathcal{D}(c_0, K, c, \alpha)\}$. Nous considérons deux cas selon qu'il existe ou non $j \in S_1$ tel que $j \in J$. Les objets de taille égale à celle du bin ne sont pas considérés :

1. Pour tout $j \in S_1$, $j \notin J$. Dans ce cas, $\sum_{i\in S_1} h_1(i) = \sum_{i\in S_1} \alpha_i \leq \mathcal{KP}\text{-}1\mathcal{D}(c_0, K, c, \alpha) \leq \max\limits_{K\in\delta}\{\mathcal{KP}\text{-}1\mathcal{D}(c_0, K, c, \alpha)\}$. Donc la relation exprimée dans la définition 2.1 est vérifiée, et h_1 est bien une GDDFF.
2. Il existe $j \in S_1$ tel que $j \in J$. En se basant sur la relation exprimée dans la définition 2.1, nous obtenons la relation suivante :

$$h_1(j) + \sum_{i\in S_1\setminus\{j\}} h_1(i) > \max_{K\in\delta}\left\{\mathcal{KP}\text{-}1\mathcal{D}(c_0, K, c, \alpha)\right\} \tag{2.2}$$

Comme $j \in J$, nous avons

$$h_1(j) = \max_{K\in\delta}\left\{\mathcal{KP}\text{-}1\mathcal{D}(c_0, K, c, \alpha)\right\} - \max_{K\in\delta/j\in K}\left\{\mathcal{KP}\text{-}1\mathcal{D}(c_0 - c_j, K \setminus \{j\}, c, \alpha)\right\}.$$

En remplaçant $h_1(j)$ par sa valeur dans l'équation (2.2) on obtient

$$\sum_{i\in S_1\setminus\{j\}} \alpha_i > \max_{K\in\delta/j\in K}\left\{\mathcal{KP}\text{-}1\mathcal{D}(c_0 - c_j, K \setminus \{j\}, c, \alpha)\right\} \tag{2.3}$$

Puisque $\sum\limits_{i\in S_1} c_i \leq c_0$, $\sum\limits_{i\in S_1\setminus\{j\}} \alpha_i \leq \max\limits_{K\in\delta/j\in K}\left\{\mathcal{KP}\text{-}1\mathcal{D}(c_0 - c_j, K \setminus \{j\}, c, \alpha)\right\}$, ce qui est en contradiction avec l'équation (2.3).

Par conséquent, h_1 est une GDDFF pour la donnée en entrée. □

Pour cette fonction, le nombre de problèmes de sac à dos à résoudre peut être exponentiel. On s'intéresse donc aux cas où le nombre de stables maximaux est réduit.

2.3.3.2.2 Calcul de la fonction h_1

Énumérer tous les sous-ensembles δ consiste à énumérer les stables maximaux de G. Le nombre de stables maximaux d'un graphe est généralement exponentiel en fonction de la taille du graphe. Les cliques maximales dans un graphe (*i.e.* stable maximal dans le graphe complémentaire) peuvent être calculées en temps linéaire si le graphe est *triangulé*. Tarjan and Yannakakis ont prouvé dans [110] que tout graphe triangulé G a au maximum n cliques maximales où n est le nombre de nœuds de G (voir section 1.3.3.2). De plus, ils ont présenté un algorithme linéaire pour reconnaître un graphe triangulé et énumérer ses cliques maximales.

Dans notre cas, le graphe de compatibilité $\overline{G}$ n'est pas forcément un graphe triangulé. Puisque nous cherchons une borne inférieure, nous relâchons le problème en ajoutant des arêtes au graphe $\overline{G}$ de telle sorte que le graphe résultant soit triangulé. En pratique, on utilise l'heuristique minimum fill-in (voir section 1.3.3)

2.3.3.2.3 Exemple illustratif

On considère une instance de $\mathcal{BPP}$-$\mathcal{C}$ composée des 16 objets suivants

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
w	2	2	3	1	3	1	2	1	1	1	1	1	1	1	1	1

La figure 2.2 montre un graphe de compatibilité associé à cette instance. Une décomposition arborescente de ce graphe triangulé comporte 6 clusters. Pour modifier l'instance par la fonction h_1, on commence par calculer la nouvelle taille du bin. Pour ce faire, six problèmes de sac-à-dos, chacun correspond à un cluster, sont à évaluer afin de calculer la nouvelle taille du bin ($\max_{i\in\{1,2,3,4,5,6\}}\{KP_i\}$). Dans la figure 2.2, $KP_1 = KP_2 = 6, KP_3 = KP_4 = 3, KP_5 = 2, KP_6 = 4$. La nouvelle taille du bin est alors $\max\{KP_1, KP_2, KP_3, KP_4, KP_5, KP_6\} = 6$.

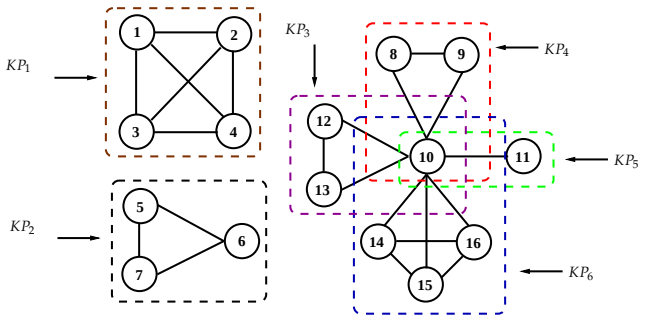


FIGURE 2.2 – *Une décomposition arborescente en 6 clusters d'un graphe triangulé. Un problème de sac-à-dos doit être résolu pour chaque cluster.*

Il reste à évaluer les nouvelles tailles des objets qui appartiennent à J et qui représentent un stable dans le graphe de compatibilité de la figure 2.2. Prenons l'exemple de l'objet 10 de la figure 2.3. Cet objet est évalué de la façon suivante : tous les clusters contenant l'objet 10 sont évalués par un problème de sac-à-dos de capacité $c_0 - c_{10}$ et la valeur maximale est gardée. La nouvelle taille de l'objet 10 est donc égale à la nouvelle taille du bin moins cette valeur.

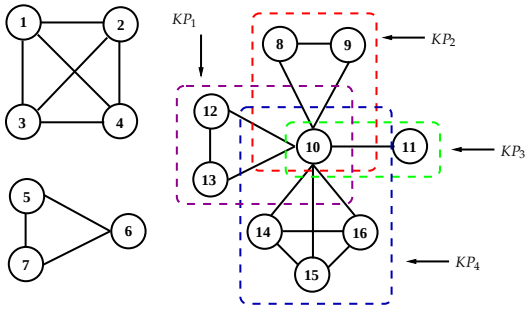


FIGURE 2.3 – *On ne résout un problème da sac-à-dos que sur les clusters pouvant accueillir l'objets* 10.

Dans la figure 2.3, $KP_1 = KP_2 = 2, KP_3 = 1, KP_4 = 3$. La nouvelle taille de l'objet $i = 10$ est $w_{10} = 6 - 3 = 3$.

2.3.3.3 La deuxième GDDFF h_2

La deuxième GDDFF h_2 est basée sur le concept de décomposition arborescente (voir section 1.3.3).Nous utilisons la notation $\delta(G)$ pour représenter l'ensemble des stables maximaux de G. Pour cette fonction, nous travaillons essentiellement sur le graphe de compatibilité $\overline{G}$ (le graphe complémentaire du graphe de conflits G).

Soit $T = (S, A)$ une décomposition arborescente de $\overline{G}$ où S est l'ensemble de sous-ensembles $I \setminus \{0\}$ et A un ensemble d'arêtes connectant les nœuds de S dans un arbre.

L'idée de base de h_2 consiste à affecter une certaine DFF f_s à chaque nœud s d'une décomposition arborescente T. Soit F une liste de DFF valides $f_1, \ldots, f_{|S|}$, affectée chacune à un nœud de la décomposition arborescente. Pour chaque nœud i du graphe, on définit S^i comme étant l'ensemble de nœuds de la décomposition arborescente contenant i. La valeur affectée à i est $\min_{s \in S^i} f_s(c_i)$. Pour discrétiser la fonction, nous introduisons un facteur de mise en échelle $\psi = \prod_{s \in S} f_s(c_0)$. Clairement, il existe toujours un ensemble de fonctions $f_1, \ldots, f_{|S|}$ qui fera mieux qu'appliquer une seule DFF (e.g. $f_1 = f_2 = \ldots = f_{|S|}$).

2.3.3.3.1 Définition

Proposition 2.3.3 *La fonction suivante h_2 est une famille de GDDFF associée à une instance $(I, c, \delta(G))$. Formellement $h_2 : I \to [0, \psi]$ est définie comme suit*

$$i \mapsto \begin{cases} \min\limits_{s \in S^i} \left\{ \frac{f(c_i)}{f(c_0)} \psi \right\} & \text{if } i \neq 0 \\ \psi & \text{if } i = 0 \end{cases}$$

Démonstration. Par définition, une clique dans le graphe de compatibilité $\overline{G}$ est incluse dans un nœud de la décomposition arborescente T de G. Considérons un nœud s_i de T tel que $\sum_{j \in s_i} c_j \leq c_0$. En appliquant h_2 sur les articles $j \in s_i$ on aboutit à l'inégalité $h_2(j) \leq f(c_j)\psi / f(c_0)$ pour tout $f \in F^j$. En sommant cette inégalité pour tous les articles $j \in s_i$, on obtient $\sum_{j \in s_i} h_2(j) \leq \sum_{j \in s_i} f(c_j)\psi / f(c_0)$. Puisque f est une DFF valide, $\sum_{j \in s_i} f(c_j)/f(c_0) \leq 1$, et donc $\sum_{j \in s_i} f(c_j)\psi / f(c_0) \leq \psi$, ce qui prouve le résultat attendu. □

2.3.3.3.2 Calcul de h_2

Pour calculer la décomposition arborescente on utilise l'heuristique MCS (voir algorithme 2).

Une autre difficulté est de choisir un bon ensemble de fonctions à appliquer sur les nœuds de la décomposition arborescente. Nous avons utilisé l'heuristique suivante : pour chaque nœud s, nous calculons la valeur de la borne associée à chacune des fonctions de l'ensemble initial F et nous gardons la fonction qui permet d'obtenir le meilleur résultat. Cette stratégie peut ne pas être optimale mais elle garantit un résultat rapide.

2.3.3.3.3 Exemple illustratif

Reprenons l'instance présentée dans l'exemple de la section 2.3.3.2. La décomposition arborescente du graphe de compatibilité de cette instance est composée de six clusters. L'ensemble F est composé des DFF f_0 et f_1 décrites dans le chapitre d'état de l'art. L'idée de h_2 est la suivante : pour chaque objet $i \in I$, on a deux cas possibles à considérer :

- si i appartient à un seul cluster, on affecte à i la DFF qui donne la valeur maximale quand on l'applique sur ce cluster ;
- si i appartient à plusieurs clusters, on calcule pour chacun de ces clusters la DFF qui donne la valeur maximale on l'applique sur ce cluster, et parmi ces DFF, on affecte à i celle qui donne la valeur minimale quand on l'applique sur i.

2.4 Nouvelles bornes inférieures pour le $\mathcal{BPP}$-$\mathcal{C}$

Dans cette section, nous présentons des nouvelles bornes inférieures pour le $\mathcal{BPP}$-$\mathcal{C}$. Ces bornes reposent sur les DFF et leurs extensions ainsi que sur une amélioration d'un modèle de transport utilisé dans [53, 2].

2.4.1 Bornes inférieures à base de DFF

La borne LB_{DFF} a été proposée par Fernandes-Muritiba *et al.* [2]. Les expérimentations qu'ils ont menées montrent que LB_{DFF} n'est pas efficace dans le cas de $\mathcal{BPPC}$-$1\mathcal{D}$ si cette borne est appliquée directement sur l'instance.

Pour améliorer LB_{DFF}, nous proposons LB_{DFF}^{imp} qui utilise des DFF et des techniques de calcul de cliques sur le graphe de conflits.

Dans ce qui suit, $\mathcal{F} = \{f_0^k, f_1^k : 1 \leq k \leq C/2\}$ représente un ensemble de DFF décrites dans le chapitre 1. Toute autre DFF peut être ajoutée à cet ensemble $\mathcal{F}$ et peut améliorer la valeur de la borne inférieure obtenue.

Nous utilisons une DFF $f \in \mathcal{F}$ pour diviser l'ensemble d'articles en deux sous-ensembles. Chaque article i est affecté à un sous-ensemble selon la valeur de $f(c_i)$. Soit $I_l = \{i \in I \ : \ f(c_i) = f(C)\}$ le sous-ensemble de grands objets, et $I_s = \{i \in I \ : \ 0 < f(c_i) < f(C)\}$ le sous-ensemble de petits objets. Soit Q_s une grande clique sur I_s. Une borne inférieure LB_{DFF}^{imp} dédiée au $\mathcal{BPPC}$-$2\mathcal{D}$ est

$$LB_{DFF}^{imp} = |I_l| + \max\left\{\left\lceil \frac{\sum_{i \in I_s} f(w_i) f(h_i)}{f(W) f(H)} \right\rceil, |Q_s|\right\}$$

Pour le cas bi-dimensionnel, la complexité temporelle de cette borne est de l'ordre de $\mathcal{O}(n^3)$. La borne LB_{DFF}^{imp} est appliquée sur l'instance prétraitée avec les procédures décrites dans la section 2.2.

2.4.2 Bornes inférieures à base de GDDFF

Dans cette section, nous décrivons comment utiliser le concept de GDDFF afin d'améliorer les bornes inférieures existantes. Nous commençons par l'amélioration de la borne continue LB_0 et nous détaillons ensuite l'amélioration que nous avons apporté à la meilleure borne inférieure polynomiale de la littérature [2, 53].

2.4.2.1 Une borne continue améliorée

Pour améliorer LB_0, nous proposons LB_0^{GDDFF} qui utilisent des GDDFF afin d'avoir une meilleure vision du problème, en prenant en compte le graphe de conflits ou le graphe de compatibilité. La méthode LB_0^{GDDFF} modifie l'instance en utilisant une GDDFF donnée et évalue ensuite la borne continue LB_0 sur l'instance modifiée.

Pour $u = 1, 2$ nous définissons $LB_0^{h_u}$:

$$LB_0^{h_u} = \left\lceil \frac{\sum_{i \in I} h_u(w_i) \times h_u(h_i)}{h_u(W) \times h_u(H)} \right\rceil \tag{2.4}$$

$$LB_0^{GDDFF} = \max_{u \in \{1,2\}} \left\{ LB_0^{h_u} \right\} \tag{2.5}$$

La complexité temporelle de cette nouvelle borne est de l'ordre $\mathcal{O}(n^3)$.

2.4.2.2 Une amélioration de LB_{CP} et LB_{CP}^{imp}

Nous présentons maintenant la borne inférieure LB_{CP}^{IMP}, une amélioration de la meilleure borne inférieure polynomiale de la littérature LB_{CP}^{IMP} proposée par Fernandes-Muritiba *et al.* [2] comme une amélioration de LB_{CP} initialement proposée

par Gendreau *et al.* [53]. Nous décrivons de manière détaillée la borne de Gendreau décrite dans la section 1.5.2. L'idée de LB_{CP} consiste à calculer une clique maximale K dans le graphe de conflits G par l'heuristique de Johnson. Ensuite, chaque article de la clique K est placé dans un bin tout seul et les articles de $I \setminus K$ seront distribué, complètement ou partiellement, dans ces bins en résolvant un problème de transport. Tous les articles et portions d'articles restants seront placés dans un bin fictif de capacité illimitée. Soit q la valeur de la borne continue appliquée sur le contenu du bin fictif. La borne LB_{CP} est évalué par $LB_{\text{CP}} = |K| + q$. L'amélioration LB_{CP}^{IMP} consiste à calculer une clique maximale K par l'heuristique de Johnson sur le graphe G, à ajouter à E les arcs (i, j) tel que $(w_i + w_j > W)$ pour obtenir le graphe $G^{'}$, ensuite à améliorer K en sélectionnant les sommets de $I \setminus K$ par ordre décroissant des degrés et la clique $K^{'}$ obtenue est une clique maximale dans $G^{'}$ et contient obligatoirement K.

La première phase de la méthode LB_{CP}^{IMP} consiste à appliquer les différentes procédures de prétraitements décrites dans la section 2.2. Ensuite, elle modifie l'instance en appliquant la GDDFF h_1. Une fois que l'instance est modifiée, LB_{CP}^{IMP} procède de la même façon que LB_{CP}^{imp}. La seule différence concerne le problème de transport.

Dans notre cas, le problème de transport est résolu une seule fois tandis que dans [53, 2] ce problème est résolu plusieurs fois en faisant varier le seuil l (voir section 1.5.2.2).

Le modèle de transport de [53, 2] ne prend pas en considération le fait que deux objets i et j de $I \setminus Q$ peuvent être en conflit. Soit k un élément de Q, alors deux arcs (i, k) de capacité c_i et (j, k) de capacité c_j sont créés. Ainsi, plusieurs morceaux des deux articles i et j peuvent être placés dans le même bin, même s'ils sont en conflit. Formellement, la contrainte de compatibilité est représentée comme suit

$$x_{ik} + x_{jk} \leq 1, \forall (i, j) \in E. \tag{2.6}$$

Puisque cette contrainte n'est pas gérée directement par le problème de transport, nous pouvons la relâcher en ajoutant la contrainte suivante

$$c_i x_{ik} + c_j x_{jk} \leq \max\{c_i, c_j\}. \tag{2.7}$$

Nous créons donc un nœud fictif ijk qui remplace les arcs (i, k) et (j, k) par trois nouveaux arcs (i, ijk) de capacité c_i, (j, ijk) de capacité c_j, et (ijk, k) de capacité $\max\{c_i, c_j\}$.

La figure 2.4 montre un exemple avec une instance $I = \{i_1, i_2, i_3, i_4, i_5\}$ et un graphe de conflits $G = (I, E)$ où $E = \{(i_1, i_2), (i_2, i_3), (i_1, i_5), (i_3, i_4)\}$. Soit $Q = \{i_4, i_5\}$ une grande clique dans G. Le cas "a" présente le modèle de transport de [53, 2] et le cas "b" le notre.

Dans la figure 2.4(b) on présente notre modèle de transport qui vise à limiter le nombre d'objets en conflit soient placés dans le même bin. Pour ce faire, nous ajoutons au réseau des nœuds supplémentaires qui serviront de nœuds fictifs (*i.e.* nœuds en couleur). Prenons par exemple le cas des deux objets i_1 et i_2, afin d'éviter qu'ils soient placés dans le bin contenant i_4, un nœud fictif (nœud de couleur rouge)

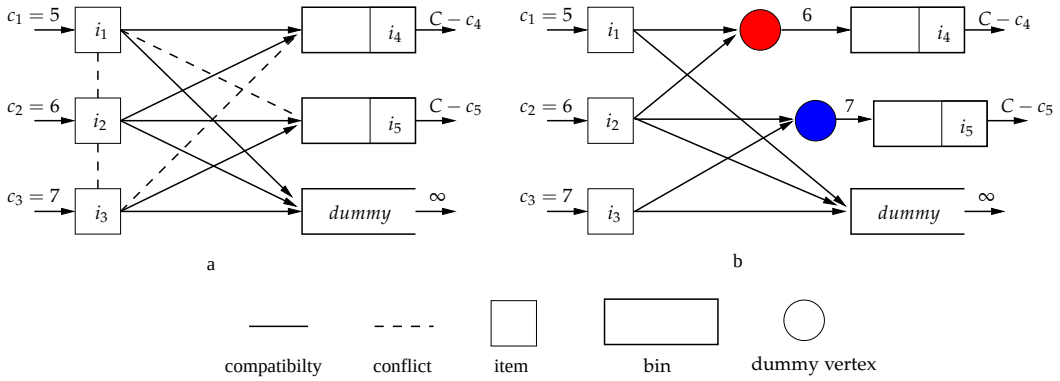


FIGURE 2.4 – *Le cas "a" montre le modèle de transport de [53, 2] et le cas "b" notre modèle.*

est créé limitant le flot à une quantité égale à la taille d'un des deux objets i_1 et i_2. Afin d'obtenir une borne inférieure valide, la capacité de cet arc doit être égale à la taille de l'objet le plus grand $max\{c_{i_1}, c_{i_2}\} = 6$.

La même opération est appliquée sur chaque triplet (i, j, k). L'ensemble des nœuds fictifs à créer peut être très grand. Pour réduire sa taille, nous proposons de créer un seul nœud pour chaque clique d'objets.

Quand le réseau est dense, une partie de ces nœuds fictifs est créée et les autres seront générées d'un façon dynamique. Le nouveau résultat obtenu permet d'obtenir des résultats qui dominent ceux de [53, 2].

L'algorithme de flot max à coût min a une complexité temporelle de l'ordre $\mathcal{O}(\widehat{n}^3)$, où $\widehat{n}$ est le nombre de nœuds dans le nouveau réseau et qui est borné par $\mathcal{O}(n^2)$, ce qui peut entraîner une complexité temporelle théorique de l'ordre $\mathcal{O}(n^6)$. Practiquement parlant, le nombre de nœuds est beaucoup plus petit que n^2. Néanmoins, pour les grands instances, il peut être pratique de limiter le nombre de nœuds fictifs de telle manière à avoir $\widehat{n} \in \mathcal{O}(n)$.

2.5 Partie expérimentale

Dans cette section, nous présentons les résultats numériques de nos bornes inférieures et de nos procédures de prétraitement. Tous nos algorithmes, ainsi que ceux de [2], ont été codés en C++ et exécutés sur un processeur Pentium IV 3 GHz avec 1Go de RAM.

2.5.1 Jeux d'essai

Dans cette section nous décrivons les jeux d'essai utilisés dans nos expérimentations afin de juger la qualité des différents algorithmes proposés dans ce chapitre.

2.5.1.1 Générateur de graphes de conflits

Le graphe de conflits est généré aléatoirement en utilisant le $\hat{p}$-generateur introduit par Soriano and Gendreau [107]. Ce générateur a été utilisé dans [2, 53] pour générer

des graphes de conflits pour le $\mathcal{BPP}$-$\mathcal{C}$. Une valeur p_i est donnée à chaque nœud $i \in I$ selon une distribution continue uniforme sur $[0, 1]$. Un arête $(i, j) \in E$ est créée si $(p_i + p_j)/2 \leq d$, où d est la densité prévue pour G et ayant pour valeur possible dans $\{0\%, 10\%, \ldots, 90\%\}$.

2.5.1.2 Instances pour le $\mathcal{BPPC}$-1$\mathcal{D}$

Le jeu d'essai pour le $\mathcal{BPPC}$-1$\mathcal{D}$ a été généré par Fernandes-Muritiba *et al.* [2] et disponible à l'URL `http://www.or.deis.unibo.it/research_pages/ORinstances/ORinstances.htm`

2.5.1.3 Instances pour le $\mathcal{BPPC}$-2$\mathcal{D}$

Le jeu d'essai pour le $\mathcal{BPPC}$-2$\mathcal{D}$ a été construit à partir de celui pour le $\mathcal{BPP}$-2$\mathcal{D}$ généré par Berkey et Wang [6] et Martello et Vigo [86]. Il est composé de dix classes d'instances où chaque classe contient cinq groupes ($n = \{20, 40, 60, 80, 100\}$) de dix instances chacun. Pour chaque classe et chaque valeur de n, dix instances ont été considérées et un graphe de conflits a été généré pour chacune des instances avec dix densités $d = \{0\%, 10\%, \ldots, 90\%\}$, donnant un total de 5000 instances.

Dans ce qui suit, vue la grande taille du jeux d'essai, nous nous concentrons sur les valeurs moyennes des résultats. Les tableaux 2.1 et 2.2 comprennent quatre colonnes principales. La première colonne indique, pour chaque ligne dans le premier sous-tableau, la classe (cl) à laquelle appartiennent les instances et la densité ($\%d$) du graphe de conflit dans le deuxième sous-tableau. La deuxième colonne UB donne la valeur de la borne supérieure. La troisième colonne, libellée KCT, représente nos bornes inférieures et la quatrième, libellée FIMT, celles de [2]. Pour mieux évaluer la performance de chaque des bornes inférieures LB, nous calculons le gap comme suit $100(UB - LB)/UB$. Dans la colonne KCT, les bornes inférieures ont été exécutées deux fois, une avec prétraitement et l'autre sans prétraitement. Pour évaluer l'efficacité des procédures de prétraitement, la colonne $\%n \rightarrow n'$ représente le pourcentage de réduction de l'instance prétraitée par rapport à celle d'origine évalué comme suit $100(n - n')/n$.

2.5.2 Résultats numériques pour le $\mathcal{BPPC}$-1$\mathcal{D}$

Dans le tableau 2.1 nous présentons les résultats numériques obtenus par nos bornes inférieures LB^{imp}_{DFF} et LB^{IMP}_{CP} et celles de la littérature LB_{DFF}, LB^{imp}_{CP} et LB_{SC}. La borne supérieure UB a été fournie par [2]. Chaque ligne dans le premier sous-tableau montre les valeurs moyennes pour 100 instances tandis que chaque ligne dans le deuxième sous-tableau montre les valeurs moyennes pour 80 instances.

La borne inférieure LB^{imp}_{DFF} est égale à LB_{DFF} quand elle est appliquée sans prétraitement. Ses résultats ne sont pas rapportés dans le tableau 2.1. Le temps de calcul de LB_{DFF} et LB^{imp}_{DFF} sont suffisamment petits pour ne pas être rapportés ainsi que le nombre de solutions optimales atteintes par LB_{DFF}. Comme on peut le voir dans

le tableau 2.1, LB_{DFF}^{imp} peut aboutir à des résultats meilleurs que LB_{DFF} si elle est appliquée après une phase de prétraitemet. Le pourcentage de gap de LB_{DFF} est toujours supérieur à 20% tandis que celui de LB_{DFF}^{imp} ne dépasse jamais 2%.

Appliquer LB_{CP}^{IMP} avec ou sans prétraitement donne les mêmes résultats en matière de gap et d'optimalité. La seule différence est la réduction du temps de calcul. Néanmoins, dans les deux cas, LB_{CP}^{IMP} a été capable de fournir une borne inférieure meilleure que celle trouvée par LB_{CP}^{imp} pour 39 instances. La moyenne du pourcentage de LB_{CP}^{IMP} est égale à 0.17 et à 0.26 dans le cas de LB_{CP}^{imp}, ce qui veut dire qu'on a pu assurer un gain en matière de gap égal à 0.09. Ce gain peut atteindre 4.76 pour certaines instances. En utilisant LB_{CP}^{IMP}, on a prouvé l'optimalité de 30 solutions supplémentaires à celles obtenues par LB_{CP}^{imp}. Toutefois, cela est obtenu avec un temps de calcul plus grand (2.06 secondes en moyenne et 91.04 secondes pour l'instance la plus lente [*BPPC_4_7_8.txt* : $LB_{CP}^{IMP} = 712$ en 91.04 secondes, $LB_{CP}^{IMP} = 704$ en 4.95 secondes et LB_{SC}=712 en 154 secondes]) en comparaison avec le temps de calcul requis par LB_{CP}^{imp}.

Mis à part le temps de calcul pouvant être relativement grand, la borne inférieure LB_{SC} dépasse toutes les autres. Ceci n'est pas surprenant étant donné que LB_{SC} est basée sur des techniques plus avancées que LB_{CP}^{imp} or LB_{CP}^{IMP} et nécessitant un temps de calcul plus important. Le temps de calcul requis par LB_{CP}^{IMP} est beaucoup plus petit que celui requis par LB_{SC} même pour les instances dont le graphe de conflits est dense. A savoir que pour 32 instances parmi les 800, LB_{CP}^{IMP} a pu avoir le même résultat que LB_{SC} avec un temps de calcul égal à 16.29 secondes en moyenne contre 142.16 secondes.

Les procédures de réduction apparaissent efficaces pour les densités supérieures à 60%. Pour la densité 90%, la plupart des objets ont été éliminés. Ceci n'étant pas surprenant : dans une solution optimale, chacun de ces objets sera placé tout seul dans un bin.

Pour le $\mathcal{BPPC}$-1$\mathcal{D}$, nos prétraitements et bornes inférieures seront le bon choix quand on est limité en terme de temps de calcul, sinon la borne inférieure LB_{SC} doit être utilisée.

2.5.3 Résultats numériques pour le $\mathcal{BPPC}$-2$\mathcal{D}$

Les résultats pour le $\mathcal{BPPC}$-2$\mathcal{D}$ ont été obtenus en transformant une instance de $\mathcal{BPPC}$-2$\mathcal{D}$ en une instance de $\mathcal{BPPC}$-1$\mathcal{D}$ après l'avoir prétraité et modifié par une GDDFF.

Comme il n'y a pas de résultats pour le $\mathcal{BPPC}$-2$\mathcal{D}$, nous comparons nos bornes inférieures LB_{CP}^{IMP} avec LB_{CP}^{imp} proposée dans [2] pour le $\mathcal{BPPC}$-1$\mathcal{D}$. D'une façon similaire à nos méthodes, l'instance bi-dimensionnelle initiale est transformée en une instance uni-dimensionnelle, et LB_{CP}^{imp} est ensuite appliquée. La comparaison avec LB_{SC} n'est pas directe car cette dernière se base sur une recherche tabou à base de population de solutions dédiée au cas bi-dimensionnel. Par conséquence, LB_{SC} n'a pas été utilisée dans la comparaison numérique. L'objectif de cette analyse numérique est

de prouver l'efficacité de nos méthodes et de montrer l'amélioration qu'elles peuvent apporter par rapport à LB_{CP}^{imp}.

Dans le tableau 2.2 nous présentons les résultats obtenus par les bornes inférieures suivantes : LB_{DFF}^{imp}, LB_{CP}^{imp} et LB_{CP}^{IMP}. Chaque ligne dans les deux sous-tableaux montre des résultats en moyenne pour 500 instances. La borne supérieure UB est évaluée par l'heuristique *Bottom-Left-Conflict (BLC)*, une simple adaptation de l'heuristique classique *Bottom-Left* de Coffman *et al.* [23].

Dans le premier sous-tableau, on note que LB_{DFF}^{imp}, LB_{CP}^{imp} et LB_{CP}^{IMP} sont égaux quelle que soit la densité du graphe dans le cas des classes 2, 4 et 6. Dans le cas de la classe 1, LB_{CP}^{IMP} améliore LB_{CP}^{imp} pour 43.6% des 500 instances, 28.6% pour la classe 3, 54.6% pour la classe 5, 44% pour la classe 7, 41.4% pour la classe 8, 82.2% pour la classe 9 et 13.2% pour la classe 10. Dans le cas de classe 9, les résultats de LB_{CP}^{IMP} sont clairement meilleurs que ceux de LB_{CP}^{imp} et sont proches de l'optimum à 0.66 près.

Le fait d'être moins bon que LB_{CP}^{imp} pour quelques instances (1.4% pour la classe 1, 2.6% pour la classe 2, 1% pour la classe 7, 1.6% pour la classe 8, 6% pour la classe 10) peut être justifiée par le fait que la triangulation de $\overline{G}$ peut être de mauvaise qualité en terme de nombre d'arêtes ajoutées, ou bien à cause du problème de sac-à-dos qui ne prend pas en considération les conflits entre les articles. Il est donc conseillé d'utiliser les deux bornes.

2.6 Conclusion

Dans ce chapitre, nous avons étudié plusieurs procédures de réduction et techniques qui peuvent être utilisées afin d'améliorer les bornes inférieures polynomiales existantes pour le $\mathcal{BPPC}$-1$\mathcal{D}$. Nous avons aussi proposé des bornes inférieures et des procédures de prétraitement pour le $\mathcal{BPPC}$-2$\mathcal{D}$.

En se basant sur le concept de fonctions dual-réalisables (DFF) et fonctions dual-réalisables dépendantes de la donnée (DDFF), nous avons proposé une nouvelle famille de DDFF ainsi que le nouveau concept de fonctions dual-réalisables généralisées dépendantes de la donnée (GDDFF).

Les résultats numériques prouvent l'amélioration apportée par nos techniques appliquées sur la borne inférieure LB_{CP}^{imp} de Fernandes-Muritiba *et al.* [2]. Pour le cas uni-dimensionnel, l'amélioration est modeste mais obtenue en un temps de calcul raisonnable. Toutefois, de meilleurs résultats peuvent être obtenus en appliquant la relaxation linéaire du modèle à base de couverture d'ensemble [2]. Pour le cas bi-dimensionnel, nos bornes inférieures améliorent les généralisations des bornes inférieures uni-dimensionnelles existantes pour 30.76% du jeu d'essai. L'amélioration obtenue peut être très importante pour certaines instances (jusqu'à 77.78%).

Une suite intéressante de ce travail consiste à proposer de nouvelles DFF pour le $\mathcal{BPP}$-$\mathcal{C}$ afin d'améliorer les résultats et d'obtenir des résultats proches de la borne de génération de colonnes pour le $\mathcal{BPPC}$-1$\mathcal{D}$.

TABLE 2.1 – *Résultats des bornes inférieures et des prétraitements pour le cas uni-dimensionnel pour les instances proposées par Fernandes-Muritiba et al. [2]. La colonne cl représente la classe, %d indique la densité du graphe de conflits, et UB donne la meilleure borne supérieure connue et proposée dans [2]. La colonne KCT représente nos bornes inférieures et FIMT ceux de [2]. Pour chaque borne inférieure, %gap représente le gap, sec. le temps de calcul en secondes et #opt le nombre de solutions résolues à l'optimalité ($LB = UB$). La colonne $\%\Delta n$ représente le pourcentage de réduction de l'instance initiale. Pour chaque sous-tableau, la ligne* Avg *montre les résultats en moyenne et* Ttl *les résultats en total.*

		KCT								FIMT						
		Procédures de réduction		LB^{imp}_{DFF}		LB^{IMP}_{CP}				LB_{DFF}	LB^{imp}_{CP}			LB_{SC}		
								Avec prétraitement	*Sans prétraitement*							
	UB	$\% n \rightarrow n'$	*sec.*	*%gap*	*#opt*	*%gap*	*#opt*	*sec.*	*sec.*	*%gap*	*%gap*	*sec.*	*#opt*	*%gap*	*sec.*	*#opt*
cl																
1	68.17	24.56	0.12	0.61	73	0.12	90	0.42	0.37	22.05	0.24	0	87	0.01	22.8	99
2	139.49	21.34	0.07	0.35	74	0.1	85	0.57	0.90	20.33	0.18	0.1	78	0	57.4	100
3	277.82	21.27	0.45	0.29	65	0.11	82	7.39	7.01	20.3	0.17	0.4	74	0.07	151.2	95
4	555.03	20.64	0.80	0.19	65	0.07	81	7.62	8.56	20.54	0.09	2.3	78	0.04	275.1	98
5	32.06	28.58	0.00	1.57	65	0.56	87	0.74	0.19	28.37	0.98	0	78	0.05	38.3	99
6	63.54	23.58	0.01	0.65	76	0.27	87	0.72	0.30	27.49	0.27	0	87	0.12	44.8	95
7	130.59	20.96	0.06	0.34	64	0.11	87	0.78	1.30	26.79	0.11	0.1	87	0.05	68.5	96
8	264.87	20.82	0.39	0.16	70	0.05	89	6.33	6.02	27.35	0.05	0.5	89	0.03	232.6	96
Avg	191.45	22.71	0.24	0.52		0.17		3.07	3.56	24.15	0.26	0.43		0.05	111.34	
Ttl					552		688						658			778
%d																
0	133.01	0.00	0.53	0.00	80	0.00	80	1.94	2.10	0.00	0.00	0.3	80	0.00	40.7	80
10	133.05	0.00	0.55	0.13	77	0.13	77	1.7	1.91	0.13	0.13	0.2	77	0.01	127.6	79
20	133.16	0.00	0.54	0.61	68	0.37	72	1.67	2.19	0.61	0.61	0.4	68	0.14	131.4	76
30	133.54	0.00	0.51	0.81	58	0.45	64	1.82	2.32	1.41	0.69	0.6	60	0.16	289.3	71
40	144.43	1.25	0.56	1.39	34	0.27	61	5.77	7.10	10.66	0.34	0.7	58	0.13	262.7	73
50	177.36	4.23	0.56	0.78	40	0.11	69	6.97	7.24	27.38	0.18	0.7	66	0.00	41.1	80
60	213.16	25.56	0.35	0.52	41	0.12	64	3.06	4.02	39.63	0.22	0.6	58	0.00	53.0	80
70	247.38	45.01	0.18	0.53	46	0.14	65	3.99	4.37	47.97	0.20	0.5	59	0.00	60.3	80
80	282.25	66.11	0.09	0.26	52	0.09	66	2.60	2.71	54.38	0.17	0.3	63	0.01	73.9	79
90	317.13	85.02	0.03	0.16	56	0.06	70	1.23	1.64	59.36	0.09	0.1	69	0.00	33.4	80
Avg	191.45	22.71	0.24	0.52		0.17		3.07	3.56	24.15	0.26	0.43		0.05	111.34	
Ttl					552		688						658			778

TABLE 2.2 – *Résultats des bornes inférieures pour le cas bi-dimensionnel pour les instances proposées par Berkey et Wang [6] et Martello et Vigo [86]. La colonne cl représente la classe, %d indique la densité du graphe de conflit, et UB donne une borne supérieure évaluée par l'heuristique BLC. La colonne KCT représente nos bornes inférieures et FIMT ceux de [2]. Pour chaque borne inférieure, %gap représente le gap, sec. le temps de calcul en secondes et #opt le nombre de solutions résolues à l'optimalité (LB = UB). La colonne %Δn représente le pourcentage de réduction de l'instance initiale. Pour chaque sous-tableau, la ligne* Avg *montre les résultats en moyenne et* Ttl *les résultats en total.*

		KCT													FIMT		
		Avec prétraitement							Sans prétraitement								
		Procédures de réduction		LB^{imp}_{DFF}		LB^{IMP}_{CP}			LB^{imp}_{DFF}		LB^{IMP}_{CP}				LB^{imp}_{CP}		
	UB	% $n \to n'$	sec.	%gap	#opt	%gap	sec.	#opt	%gap	#opt	%gap	sec.	#opt		%gap	sec.	#opt
cl																	
1	36.65	51.3	0.00	5.47	163	5.19	0.12	170	5.56	154	5.21	0.35	169		7.49	0.11	110
2	26.65	84.8	0.00	1,33	455	1,33	0.10	455	1,33	455	1,33	0.40	455		1.33	0.11	455
3	32.9	53.63	0,01	5,69	177	5,61	0.21	180	5,84	172	5,61	0.41	180		7.46	0.21	139
4	25.2	83.4	0,03	1,68	435	1,68	0.36	435	1,68	435	1,68	0.64	435		1.68	0.21	435
5	33.83	50.77	0,04	6,05	174	5,9	1.03	178	6,27	167	5,91	1.84	178		10.02	0.21	118
6	28.89	86.6	0.03	1.46	458	1.46	1.97	458	1.46	458	1.46	2.70	458		1.46	0.22	458
7	32.32	43.9	0.00	6.03	183	5.68	1.10	188	6.16	181	5.69	1.54	188		9.91	0.21	155
8	33.1	42.62	0.03	6.19	159	6.04	1.28	161	6.39	153	6.05	1.96	161		10.12	0.22	102
9	48.16	94.65	0.00	0.76	344	0.72	0.43	352	0.85	333	0.72	0.88	352		14.85	0.18	78
10	27.8	52.65	0.02	7	191	6.9	1.13	193	7.08	186	6.91	1.79	193		6.95	0.22	165
Avg	32.55	64.43	0.01	4.17		4.05	0.77		4.26		4.05	1.25			7.13	0.19	
Ttl					2739			2770		2694			2769				2215
%d																	
0	15.98	21.1	0.04	6.48	218	6.42	0.93	221	6.63	213	6.42	1.48	221		12.76	0.18	160
10	16.95	33.29	0.04	7.17	180	7.13	0.81	183	7.28	176	7.13	1.21	183		13.38	0.19	125
20	19.63	40.29	0.03	8.65	197	8.58	0.89	198	8.73	193	8.58	1.15	198		13.77	0.2	148
30	23.57	50.79	0.01	7.92	231	7.63	1.03	234	7.97	229	7.63	1.95	234		11.06	0.2	196
40	28.16	58.85	0.02	4.45	247	4.24	1.18	252	4.67	246	4.29	1.67	251		7.07	0.2	213
50	33.22	70.21	0.02	3.5	231	3.16	0.8	237	3.64	227	3.18	1.2	237		5.77	0.2	195
60	39.32	84.18	0.00	1.65	292	1.59	0.76	294	1.73	287	1.61	1.21	294		3.23	0.19	241
70	44.71	90.08	0.00	1.04	329	1	0.42	331	1.12	324	1	0.81	331		2.25	0.18	267
80	49.55	97.12	0.00	0.53	383	0.51	0.56	386	0.58	378	0.51	1.10	386		1.26	0.18	316
90	54.42	98.41	0.00	0.26	431	0.24	0.35	434	0.31	421	0.24	0.73	434		0.73	0.17	354
Avg	32.55	64.43	0.01	4.17		4.05	0.77		4.26		4.05	1.25			7.13	0.19	
Ttl					2739			2770		2694			2769				2215

Chapitre 3

Résolution du bin packing avec conflits basée sur la décomposition arborescente

Le travail de ce chapitre a fait l'objet d'un article accepté dans "Computers & Operations Research (COR)", d'une publication dans une conférence internationale [73] et d'une présentation orale dans une conférence nationale française [75].

3.1 Introduction

La principale contribution de ce chapitre est une nouvelle approche de résolution pour le bin packing avec conflits basée sur le concept de décomposition arborescente (voir section 1.3.3).

Le problème de $\mathcal{BPP}$-$\mathcal{C}$ est à mi chemin entre coloration de graphes et bin-packing. Par conséquent, nous avons cherché à mettre au point une méthode de résolution capable de prendre en compte les spécificités des deux problèmes. Cela a été réalisé à l'aide du concept de décomposition arborescente (voir section 1.3.3), qui permet de décomposer le graphe de compatibilité de notre problème en sous-ensembles qui vont être résolus à l'aide d'algorithmes dédiées au bin-packing. Le concept de décomposition arborescente a déjà fait ses preuves pour résoudre le problème de coloration de graphes (*e.g.* [82]).

Néanmoins, appliquer une décomposition arborescente sur une instance de $\mathcal{BPP}$-$\mathcal{C}$ n'est pas direct. Dans un premier temps, une décomposition arborescente est calculée pour le graphe de compatibilité d'une instance donnée. Cette décomposition réalise une couverture du graphe en un ensemble de sous-ensembles de sommets, dits "clusters", qui peuvent ou non avoir des sommets en commun qu'on appelle "séparateurs". Afin de pouvoir résoudre les problèmes associés aux clusters de manière indépendante, on affecte chaque article à un cluster unique (on parle de séparation de clusters). Une méthode de résolution est ensuite appliqué à chaque cluster, et les résultats obtenus sont fusionnés pour obtenir une solution globale.

La séparation des clusters est l'étape la plus importante de notre méthode. Nous prouvons que le problème qui consiste à trouver la meilleure séparation des clusters est $\mathcal{NP}$-complet. Pour cette raison, nous proposons un ensemble d'algorithmes rapi-

des et efficaces pour trouver une séparation de bonne qualité. Une recherche tabou à base de mouvements oscillatoires est aussi proposée afin d'améliorer les résultats obtenus par les algorithmes constructifs.

Il est à noter que notre méthode de décomposition peut donner lieu à de multiples hybdridations de méthodes. En effet, des méthodes heuristiques, métaheuristiques ou même exactes peuvent être appliquées à chaque étape de l'algorithme. En pratique, nous l'avons validée à l'aide d'heuristiques et de métaheuristiques.

Outre le fait qu'elle permet d'améliorer les résultats des heuristiques constructives, notre méthode permet aussi de traiter des instances de grande taille si leur largeur d'arbre est petite. Dans ce cas, notre méthode divise les problèmes de grande taille en un ensemble de sous-problèmes de petites tailles.

Nous reproduisons les résultats expérimentaux obtenus par les différentes procédures proposées dans ce chapitre. Nous les avons testées sur des jeux d'essai décrits dans la section . Les résultats expérimentaux montrent l'efficacité de notre approche qui améliorent strictement les résultats obtenus pour le $\mathcal{BPPC}$-$2\mathcal{D}$ en adaptant les méthodes dédiées au $\mathcal{BPPC}$-$1\mathcal{D}$ pour la majorité du jeu d'essai.

Le reste de ce chapitre est organisé comme suit. Dans la section 3.2, nous présentons en détail la nouvelle approche de résolution basée sur le concept de décomposition arborescente. La section 3.3 est consacrée à des heuristiques rapides pour exploiter la décomposition arborescente dans un contexte de bin packing. Afin d'assurer une bonne exploitation de la structure d'une décomposition arborescente, nous proposons une recherche tabou dans la section 3.4. Avant d'aborder les résultats numériques des différentes méthodes dans la section 3.6, une analyse de complexité de l'approche est développée dans la section 3.5.

3.2 Nouvelle approche de résolution

Dans cette section, nous présentons une nouvelle approche pour résoudre le $\mathcal{BPP}$-$\mathcal{C}$. Cette approche est basée essentiellement sur une décomposition arborescente du graphe de compatibilité de l'instance traitée. Pour tester son efficacité, cette approche sera appliquée à la variante bi-dimensionnelle du $\mathcal{BPP}$-$\mathcal{C}$. L'idée est la suivante : une fois que le graphe de compatibilité est décomposé en sous-ensembles, chaque sommet est affecté à un seul cluster afin d'éviter qu'un objet soit placé dans plusieurs bins, ce qui peut dégrader la qualité de la solution finale. Nous nommons cette affectation une *séparation de clusters* et nous montrons que le problème consistant à trouver la meilleure séparation de clusters est un problème $\mathcal{NP}$-complet. Dans les sections 3.3 et 3.4, des algorithmes sont présentés afin de résoudre ce problème.

3.2.1 Idée principale

Étant donné un graphe de conflits $G = (I, E)$, on note $\overline{G} = (I, I \times I \setminus E)$ le graphe de compatibilité correspondant.

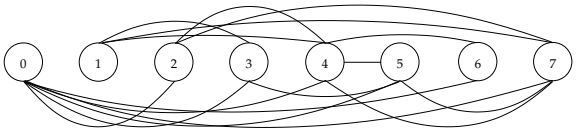


FIGURE 3.1 – *Un exemple de graphe de conflits G.*

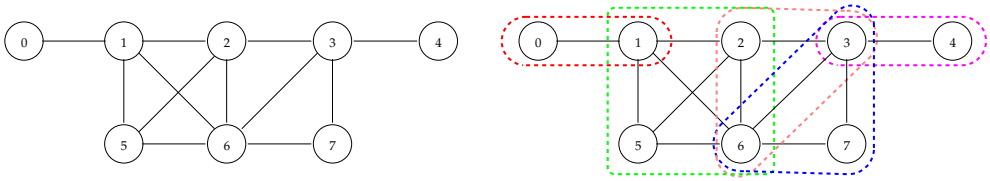


FIGURE 3.2 – *La sous-figure (a) montre le graphe de compatibilité $\overline{G}$ qui correspond au graphe de conflits G de la figure 3.1. La sous-figure (b) montre une décomposition arborescente de $\overline{G}$ établie par l'algorithme MCS (voir algorithme 2).*

La figure 3.1 présente un graphe de conflits G et un graphe de compatibilité $\overline{G}$ qui correspond à G est représenté dans la sous-figure 3.2(a). La décomposition arborescente (C, T) de $\overline{G}$, établie dans la sous-figure 3.2(b), a été établie par la méthode MCS où $C = \{c_1 = \{0,1\}, c_2 = \{1,2,5,6\}, c_3 = \{2,3,6\}, c_4 = \{3,6,7\}, c_5 = \{3,4\}\}$.

Notre approche consiste à appliquer une décomposition arborescente sur le graphe de compatibilité $\overline{G}$. Cette décomposition comprend un ensemble de clusters qui sont traités comme étant des sous-problèmes à résoudre itérativement. Ensuite, les solutions partielles sont fusionnées pour aboutir à une solution pour l'instance initiale. L'algorithme 5 montre les différentes étapes de l'approche.

Algorithme 5 : Approche de résolution pour le $\mathcal{BPP}$-$\mathcal{C}$ basée sur la décomposition arborescente.

donnée : Un ensemble I de n objets et $G = (I, E)$ un graphe de conflits.
résultat : Un placement des n objets dans un ensemble B de bins.

1 $(C, T) \longleftarrow$ établir_décomposition-arborescente($\overline{G}$);
2 $\mu(C, T) \longleftarrow$ séparer_clusters(C, T);
3 $B \longleftarrow \emptyset$;
4 **pour chaque** $c_i \in \mu(C, T)$ **faire**
5 $\quad B \longleftarrow B \cup$ résoudre_sous-problème(c_i);
6 $B \longleftarrow$ améliorer_solution_finale(B);

A la ligne 1, on décompose en clusters le graphe de compatibilité. Afin de résoudre le problème lié aux sommets appartenant à plusieurs clusters, on applique à la ligne 2 une méthode détaillée dans la section suivante pour séparer les clusters qui ont des sommets en commun. Dans la figure 3.3 nous montrons un exemple d'une séparation des clusters de la décomposition présentée dans la figure 3.2.

Les sous-problèmes induits par les clusters sont ensuite traités itérativement par une méthode de résolution à la ligne 5 (voir section 3.2.3), et les bins résultants sont ensuite ajoutés à l'ensemble de bins B. Une fois les solutions partielles fusionnées

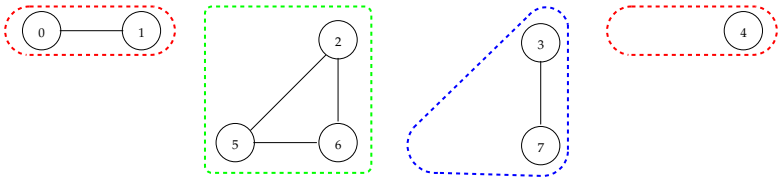


FIGURE 3.3 – *Un exemple d'une séparation des clusters de la décomposition présentée dans la figure 3.2.*

pour obtenir une solution globale, cette solution est améliorée par une heuristique décrite dans la section 3.2.4 qui vise à éliminer certains bins si leurs contenus peuvent être distribués sur les autres bins.

Les méthodes utilisées dans ce chapitre sont des méthodes heuristiques. Néanmoins, la nouvelle approche peut aussi être exploitée par des méthodes exactes ou hybrides sur plusieurs niveaux (calculer une décomposition arborescente, séparer les clusters, placer les objets, améliorer la solution finale). Cela pourrait améliorer les résultats mais aussi augmenter le temps de calcul (placer des objets bi-dimensionnels optimalement est plus difficile que de placer des objets uni-dimensionnels, même lorsque tous les objets peuvent être placés dans le même bin, voir [22, 100]).

3.2.2 Le problème de séparation de clusterss

Comme on peut le voir sur la figure 3.2(b), un objet peut appartenir à plusieurs clusters (par exemple le sommet 6 appartient à c_2, c_3 et c_4). Si l'objet correspondant est traité autant de fois qu'il appartient à un cluster, la solution obtenue peut être de très mauvaise qualité.

Le point délicat de cette nouvelle approche est donc de partitionner l'ensemble des sommets dans les clusters, un problème qu'on appelle *séparation de clusterss* dans le suite du chapitre. Le choix de la séparation de clusters fait partie des points importants à évoquer dans la suite. Nous prouvons qu'il existe une séparation de clusterss qui permet de trouver une solution optimale de l'instance.

Proposition 3.2.1 *Pour chaque instance de $\mathcal{BPP}$-$\mathcal{C}$ avec un graphe de compatibilité $\overline{G}$ et une décomposition arborescente (C, T) de $\overline{G}$, il existe une séparation de clusters μ de (C, T) tel que, pour chaque cluster $c_i \in \mu$, résoudre le sous-problème induit par c_i par une méthode exacte aboutit à la solution optimale de l'instance.*

Démonstration. Soit s^* une solution optimale pour une instance de $\mathcal{BPP}$-$\mathcal{C}$. Cette solution comprend un certain nombre de bins, chacun contenant un ensemble d'objets compatibles entre eux placés ensemble et formant un stable dans G, l'équivalent d'une clique dans $\overline{G}$. La décomposition arborescente d'un graphe triangulé étant l'ensemble de ses cliques maximales (voir section 1.3.3), une clique dans un graphe appartient forcément à un cluster dans la décomposition arborescente de ce même graphe. Alors, les clusters résolus pour avoir s^* sont obtenus par une séparation de clusterss de la décomposition arborescente. □

Nous prouvons maintenant que trouver "la meilleure séparation de clusters" est $\mathcal{NP}$-complet pour un graphe quelconque. Pour ce faire, nous allons montrer comment transformer toute instance d'un *problème de partition* [51] en une instance d'un séparation de clusters.

Définition 3.1 *Etant donnée une instance de $\mathcal{BPPC}$-2$\mathcal{D}$ avec un graphe de compatibilité $\overline{G}$ et (C, T) sa décomposition arbitraire et k une valeur entière. La problème de la meilleure séparation de clusters consiste à trouver une séparation des clusters de (C, T) dont la résolution des clusters aboutit à une solution optimale de valeur k, si elle existe.*

Définition 3.2 *Etant donné un ensemble S d'entiers, le problème de partition consiste à répondre à la question suivante : Existe-t-il une façon de séparer S en deux sous-ensembles S_1 et S_2 de telle façon que la somme des entiers dans S_1 soit égale à celle de S_2 ?*

Proposition 3.2.2 *([51]) Le problème de partition est $\mathcal{NP}$-complet.*

Proposition 3.2.3 *Le problème de la meilleure séparation de clusters est $\mathcal{NP}$-complet.*

Démonstration. Le preuve se fait en construisant une instance du problème de la meilleure séparation de clusterss à partir de toute instance de problème de partition. On considère le problème de partition avec un ensemble S d'entiers de taille $s_1, \ldots, s_l$. A partir de cette instance, on construit une instance du problème de meilleure séparation de clusters :

- pour chaque taille s_i, on crée un objet i de largeur s_i et hauteur 1
- on crée deux objets supplémentaires $l+1$ et $l+2$ de largeur 1 et hauteur 1
- la largeur du bin, dite W, est égale à $\sum_{i=1}^{l} s_i/2 + 1$ et la hauteur H est 1
- le graphe de compatibilité est $(I, I \times I \setminus \{[l+1, l+2]\})$, *i.e.* tous les objets sont deux-à-deux compatibles sauf $l+1$ et $l+2$
- la décomposition arborescente (C, T) est composée de deux clusters $c_1 = I \setminus \{l+1\}$ et $c_2 = I \setminus \{l+2\}$.

Si on est capable de répondre à la question "Existe t-il une séparation de clusters entraînant une solution à 2 bins ?", on est alors capable de résoudre le problème de partition lié à l'ensemble S_1. Par conséquent, le problème de meilleure séparation de clusters est $\mathcal{NP}$-complet. □

3.2.3 Résoudre les sous-problèmes induits par les clusters

Sur les lignes 4-6 de l'algorithme 5, le sous-problème induit par le cluster c_i est à résoudre et les bins résultants seront ajoutés à l'ensemble des bins B. Ce sous-problème peut être vu comme une instance de $\mathcal{BPP}$-$\mathcal{C}$ et donc résolu par tout algorithme de résolution, exact, hybride ou approché dédié à ce problème.

Pour tester l'efficacité de cette approche, nous l'avons appliquée sur la variante bi-dimensionnelle du $\mathcal{BPP}$-$\mathcal{C}$. Nous avons donc utilisé deux algorithmes de la littérature dédiées au $\mathcal{BPPC}$-2$\mathcal{D}$. Le premier algorithme "BLC" est une simple généralisation de l'heuristique *bottom-left* (BL) proposée par Coffman *et al.* [23] pour le $\mathcal{BPP}$-2$\mathcal{D}$ et

décrite dans la section 1.4.4.2. Cette adaptation consiste à tester, avant de mettre un objet dans un bin donné, si cet objet est compatible avec tous les objets déjà placés dans ce bin. Le deuxième algorithme "H6-2D" est une simple adaptation de l'heuristique *H6* proposée par Gendreau *et al.* [53] pour résoudre le $\mathcal{BPPC}$-1$\mathcal{D}$ et décrite dans la section 1.5.3.

3.2.4 Heuristique d'amélioration

Une fois les sous-problèmes résolus, les solutions partielles sont fusionnées afin d'avoir une solution de la globalité de l'instance. Ensuite, une heuristique d'amélioration est appliquée sur cette solution. Cette heuristique est basée sur un processus de réduction progressive du nombre de bins utilisés par la solution. L'idée consiste à vider un certain nombre de bins et de redistribuer leurs contenus sur le reste des bins.

Soit B l'ensemble des bins obtenus en résolvant les sous-problèmes induits par les différents clusters. Un ensemble $\mathcal{B}$ de bins *candidats* est extrait de B en considérant uniquement les bins qui ne contiennent que des objets appartenant à plusieurs clusters.

Les bins $b \in \mathcal{B}$ sont successivement éliminés et les objets de ces bins sont redistribués sur les bins restants. Si la solution est faisable, une meilleure solution est obtenue et le processus est réitéré.

3.3 Heuristiques basées sur la décomposition arborescente

Comme la séparation de clusterss d'un problème difficile, nous avons utilisé des méthodes heuristiques pour le résoudre. Cette phase est la phase la plus cruciale de notre approche : établir une mauvaise séparation de clusterss peut aboutir à une solution de mauvaise qualité même si l'on dispose d'une très bonne heuristique pour résoudre les problèmes induits par les clusters. Dans cette section, on propose deux façons d'utiliser l'algorithme 5, des stratégies gloutonnes et une méthode "multi-start".

3.3.1 Stratégies gloutonnes

Ces méthodes consistent à établir un certain ordre de traitement sur les clusters. En d'autre terme, elles appliquent l'algorithme 5 tout en calculant une séparation de clusters (ligne 2) par des heuristiques basées sur un ordre de traitement prédéfini sur les clusters. Considérons une décomposition arborescente (C, T) et soit $N = |C|$. Une séparation de clusterss peut être calculée comme suit :

- Numéroter les clusters selon un critère donné $\mathcal{N} : c \to \{1, \ldots, N\}$.
- Pour chaque valeur i en ordre croissant :

- Placer les objets restants du cluster courant c_i dans un ensemble S_i
- Retirer tous les objets de S_i de tous les clusters c_j tel que $j > i$.

Plusieurs critères ont été proposés pour explorer l'arbre des clusters associé à une décomposition arborescente (voir *e.g.* [66]). Deux types de critères ont été introduits dans [66] : *local* et *global*. Un critère local (resp. global) évalue l'importance d'un cluster candidat sans prendre (resp. en prenant) en considération ses interactions avec d'autres clusters. Ils ont aussi proposé deux critères, la *taille d'un cluster* ("local", le nombre d'éléments dans le cluster) et la *taille du voisinage d'un cluster* ("global", le nombre de clusters auxquels il est connecté dans l'arbre).

Dans ce chapitre nous proposons un nouveau critère global, la *demande* $\mathcal{D}(i)$ d'un objet i comme étant le nombre de clusters contenant i. Ce critère peut être généralisé et appliqué aux clusters comme suit : la demande d'un cluster c_k est égale à la somme des demandes des objets qu'il contient $\mathcal{D}(c_k) = \sum_{i \in c_k} \mathcal{D}(i)$. Un cluster avec une grande demande partage beaucoup d'objets avec d'autres clusters, et donc ce critère aide à identifier ces clusters afin qu'ils soient considérés comme des clusters *centraux*. Nous avons aussi introduit le critère local *rand* qui consiste à établir un ordre aléatoire sur les clusters.

Le choix de ces heuristiques simples est justifié par le fait qu'elles sont rapides, car une heuristique plus complexe peut augmenter le temps de calcul total de l'algorithme 5.

3.3.2 Méthode multi-start

Le résultat de l'algorithme 5 dépend essentiellement de la qualité de la séparation de clusters établie à la ligne 2. Les méthodes de recherche basées sur une optimisation locale nécessitent une certaine diversification afin d'échapper aux optima locaux. Une manière d'implémenter cette diversification est de relancer l'algorithme itérativement en modifiant ses paramètres de manière aléatoire. On parle de la technique "multi-start". Les stratégies de relance peuvent être utilisées pour aider la recherche à trouver de nouvelles solutions en visitant de nouvelles zones prometteuses de l'espace de recherche.

Le mécanisme de relance peut être inséré dans l'algorithme 5 comme suit. A chaque itération, une nouvelle séparation de clusters est aléatoirement construite (ligne 2), les sous-problèmes induits par les clusters séparés sont ensuite résolus par des méthodes dédiées au $\mathcal{BPP}$-$\mathcal{C}$ et la solution obtenue est ensuite améliorée par l'heuristique décrite plus haut. Le processus est itéré tant qu'aucun critère d'arrêt n'a été vérifié.

3.4 Recherche tabou basée sur la décomposition arborescente

Les méthodes de recherche locale sont largement connues comme étant un outil puissant garantissant des solutions de bonne qualité pour un grand nombre de problèmes combinatoires. Comme nous l'avons déjà mentionné, le problème de séparation de clusterss représente le cœur de notre approche. Afin d'obtenir des meilleurs résultats par rapport à ceux obtenus par les heuristiques, nous proposons dans cette section une méthode de recherche tabou qui sera basée essentiellement sur le concept de décomposition arborescente et sur une stratégie de diversification alternant des phases de construction et de destruction de la solution courante. Nous nommons notre méthode TS-TD.

Nous commençons par introduire les points essentiels d'une recherche tabou, ensuite nous expliquons en détail les différentes composantes de notre méthode. Nous avons utilisé les composants logiciels de la plateforme ParadisEO [10].

3.4.1 Recherche tabou

La méthode de recherche tabou a été introduite par Glover en 1986 [57]. Sa principale particularité tient dans la mise en oeuvre de mécanismes inspirés de la mémoire humaine. Son fonctionnement est dérivé de celui de la méthode de descente. A chaque itération, la recherche tabou consiste à retenir le meilleur élément du voisinage considéré même si celui-ci dégrade la solution courante. Ceci permet d'échapper aux optima locaux mais peut être à l'origine de cycles. Afin de remédier à ce problème, les configurations visitées sont mémorisées dans une mémoire à court terme dite *liste tabou*. Pour des raisons de temps de calcul et de place mémoire, ne sont stockées que certaines attributs caractérisant les configurations visitées, en conséquence de quoi toutes les configurations possédant ces attributs deviennent tabou, y compris celles qui n'ont jamais été visitées. Pour pallier ce défaut, un mécanisme particulier, l'*aspiration*, est mis en place. Celui-ci permet de révoquer le statut tabou d'une configuration lorsque les circonstances s'y prêtent, par exemple si celle-ci est meilleure que la meilleure configuration rencontrée depuis le début de la recherche. La méthode tabou constitue une métaheuristique flexible qui permet d'incorporer facilement des connaissances propres au problème traité.

3.4.2 Codage d'une solution

Une solution est représentée par un vecteur v de taille n (n étant le nombre d'objets dans l'instance traitée). A chaque objet i est associé un élément v_i du vecteur v, qui représente la valeur affectée à l'objet i, où $v_i \in D_i$ et D_i est l'ensemble des clusters pouvant accueillir i. Par exemple, pour le cas de la figure 3.2, le domaine de l'objet 3 est $D_3 = \{3, 4, 5\}$, ce qui veut que l'objet 3 peut être placé dans les clusters 3, 4 ou 5.

La phase d'initialisation est simple : les objets i dont le domaine D_i ne contient qu'une seule valeur (*i.e.* $|D_i| = 1$) sont fixés à cette unique valeur et les autres objets sont initialement fixés à -1. La figure 3.4 illustre l'exemple d'un vecteur et de son initialisation.

FIGURE 3.4 – *Un vecteur représentant une solution et son initialisation pour le cas de la figure 3.2.*

Dans la figure 3.4, on peut remarquer que les objets i_0, i_4, i_5 and i_7 sont les seuls à être initialisés (chacun d'eux ne peut appartenir qu'à un seul cluster, voir figure 3.2). Les objets restants fixés à -1 sont ensuite affectés à des clusters au cours de la recherche.

3.4.3 Espace de solutions

Dans une recherche locale, un espace de solution $\mathcal{S}$ est défini comme étant l'ensemble des solutions possibles. L'espace de solutions de notre TS-TD comprend des solutions complètes et incomplètes. Une solution est dite complète (resp. incomplète) si elle ne possède pas (resp. possède) des variables non affectées. Dans notre approche, toute solution complète est faisable. La figure 4.2 montre l'espace de solution correspondant à la figure 3.2.

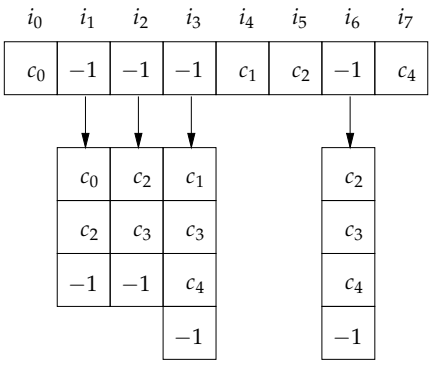


FIGURE 3.5 – *L'espace de recherche qui correspond au graphe de la figure 3.2. Pour chaque objet i, sauf i_0, i_4, i_5, et i_7 dont la valeur est fixée à -1, on crée une liste D_i contenant l'ensemble des clusters pouvant accueillir i.*

La taille de $\mathcal{S}$ dépend de la taille des domaines des objets (voir figure 4.2), et peut être calculée comme suit :

$$|\mathcal{S}| = \prod_{i \in I / v_i = -1} (|D_i| + 1)$$

3.4.4 Mouvements

Un mouvement est une affectation d'une variable v_i à une valeur de l'ensemble $D_i \cup \{-1\}$. Cela revient à affecter un objet i à un cluster ($v_i \in D_i$) "*mouvement en*

avant", ou à enlever un objet i d'un cluster ($v_i \leftarrow -1$) "*mouvement en arrière*". L'existence de deux types de mouvements est justifiée par le fait que notre recherche tabou comprend deux phases, une phase *constructive* et une phase *destructive*. La phase constructive guide la recherche afin d'obtenir des solutions complètes tandis que la phase destructive a pour rôle de détruire la solution courante en désinstanciant certains variables. Faire alterner ces deux phases assure le rôle d'un processus de diversification permettant à la recherche tabou de visiter de nouvelles zones éventuellement plus prometteuses de l'espace de recherche.

3.4.5 Fonction d'évaluation

Un facteur crucial de toute recherche locale est la fonction d'évaluation f. Dans le contexte de bin packing, choisir la fonction objectif classique, qui consiste à minimiser le nombre de bins, n'est pas pertinent étant donné que plusieurs différentes solutions peuvent avoir le même nombre de bins sans avoir la même qualité. Néanmoins, cette information est intéressante à condition qu'elle soit couplée avec d'autres informations. Pour cela, nous avons choisi une fonction objectif f qui contient des informations sur le nombre d'objets instanciés ainsi que la valeur d'écart (GAP_c) définie comme suit pour un cluster c :

$$\mathrm{GAP}_c = \mathrm{UB}_c * (WH)^2 - \left(\sum_{i \in c} w_i h_i \right)^2$$

où UB_c est le nombre de bins nécessaires pour placer les objets associés au cluster c et évalué par n'importe quelle heuristique dédiée au $\mathcal{BPPC}$-$2\mathcal{D}$ (BLC dans notre cas).

Ainsi la fonction d'évaluation peut être écrite comme suit

$$f(s) = \alpha_1 |I^{-1}| + \alpha_2 \sum_{c \in C} \mathrm{UB}_c + \alpha_3 \sum_{c \in C} \mathrm{GAP}_c$$

où $I^{-1} = \{i \in I : v_i = -1\}$, et α_1, α_2 et α_3 sont trois coefficients réels.

Le premier terme dans f a pour rôle de choisir la solution ayant le plus de variables affectées. Le troisième terme a pour rôle de choisir la solution ayant le plus petit GAP entre celles ayant un même nombre de bins.

3.4.6 Exploration du voisinage

En partant d'une solution initiale, une méthode de recherche locale procède en effectuant une séquence de mouvements, chacun améliorant la valeur de la fonction objectif, jusqu'à aboutir à un optimum local. Au contraire des méthodes de recherche locale classiques, une recherche tabou accepte d'effectuer des mouvements non améliorants si, à partir de la solution courante, il n'existe plus de mouvements améliorants. Le voisinage est exploré d'une façon déterministe et le meilleur des voisins qui est

choisi remplace la solution courante. Quand un optimum local est atteint, le processus de recherche choisit le voisin qui dégrade le moins possible la solution courante.

L'évaluation de la solution courante après avoir effectué un mouvement est effectuée d'une façon incrémentale. La valeur apportée par un mouvement dépend de la nature du mouvement (mouvement en avant ou en arrière). Un mouvement en avant déplaçant un objet i d'un cluster c_1 vers un cluster c_2 est évalué de la façon suivante :

$$\Delta f^i_{c_1,c_2} = \begin{cases} -1 + \mathrm{UB}_{c_2\cup\{i\}} + \mathrm{GAP}_{c_2\cup\{i\}} & \text{si } v_i = -1 \\ \mathrm{UB}_{c_2\cup\{i\}} + \mathrm{GAP}_{c_2\cup\{i\}} - \mathrm{UB}_{c_1\setminus\{i\}} - \mathrm{GAP}_{c_1\setminus\{i\}} & \text{sinon.} \end{cases}$$

La valeur d'un mouvement en arrière enlevant un objet i d'un cluster c est évalué comme suit :

$$\Delta f^i_c = 1 - \mathrm{UB}_{c\setminus\{i\}} - \mathrm{GAP}_{c\setminus\{i\}}.$$

Après avoir identifié, à partir de la solution courante, l'ensemble des mouvements améliorants la valeur de la fonction objectif, notre méthode de recherche tabou choisit celui aboutissant à la meilleure amélioration.

3.4.7 Stratégies de diversification et d'intensification

Afin de s'échapper des minima locaux et de permettre à la recherche tabou de visiter des zones de l'espace de recherche potentiellement plus prometteuses, nous avons mis en place un processus de diversification. Certaines stratégies de diversification de la littérature modifient totalement ou quasi totalement la solution courante. Ce genre de stratégie n'est pas adapté à notre cas, ce qui peut être justifié par le fait qu'arrêter la recherche et la relancer dans une nouvelle région peut entraîner une dégradation radicale et éventuellement laisser échapper des solutions qui étaient dans un voisinage proche de la solution courante.

De manière similaire à notre exploration du voisinage, notre stratégie de diversification comporte deux phases, "constructive" et "destructive". Tout au long de la recherche, nous allons veiller à alterner ces deux phases pour assurer une bonne diversification de la solution courante. La figure 3.6 montre le déroulement de la recherche tabou depuis son départ jusqu'à son arrêt. Elle commence par une solution incomplète s et déclenche une phase de construction afin d'avoir une solution complète, une phase de destruction est ensuite déclenchée aboutissant à une solution incomplète de nouveau et le processu est itéré tant qu'un critère d'arrêt n'a pas été atteint.

Ce trajet peut être contrôlé par un ensemble de paramètres, l' *amplitude* a et la *fréquence* f. L'amplitude représente le nombre maximum de mouvements en arrière à effectuer pendant une phase destructive. La fréquence est le nombre minimum de solutions complètes à atteindre tout au long de la recherche.

L'intensification dans les zones prometteuses consiste d'habitude à relancer la recherche en partant de la meilleure solution trouvée depuis le début. Dans notre cas,

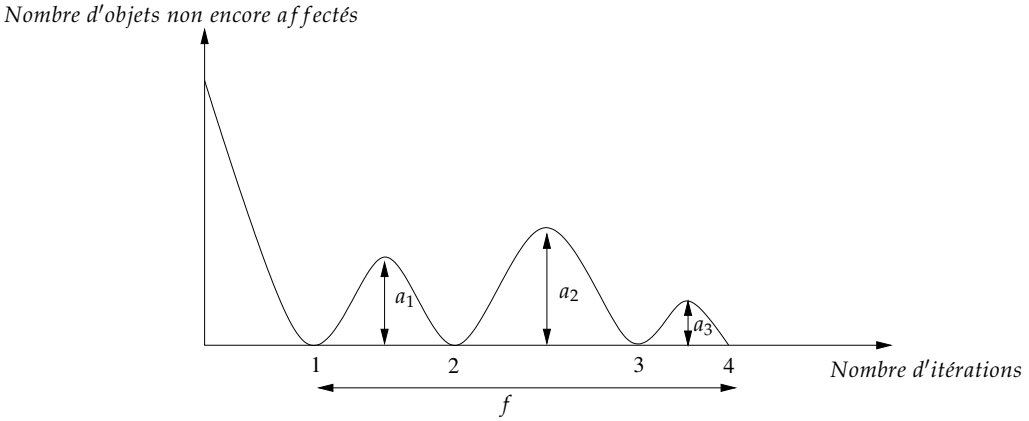


FIGURE 3.6 – *Le déroulement de la recherche tout au long du processus d'optimisation. La fréquence f est le nombre de fois qu'une solution complète est obtenue. L'amplitude a représente le nombre d'objets à désinstancier dans une phase destructive.*

l'intensification consiste à appliquer l'heuristique d'amélioration décrite plus haut sur toute solution complète atteinte à la fin d'une phase constructive. En utilisant cette heuristique, la recherche tabou va essayer de concentrer sa recherche autour de la meilleure solution.

3.4.8 Liste tabou

Dans une recherche tabou, une liste tabou est nécessaire afin d'éviter les cycles, une chose qui arrive quand on essaie d'instancier les derniers objets restants. Dans notre implémentation, nous utilisons une liste tabou contenant l'ensemble des mouvements qui sont interdits pour une certaine durée. Cette durée est définie dynamiquement en fonction de la taille de la liste tabou initialement égale au nombre de mouvements possibles plus un certain ϵ où ϵ représente le nombre d'objets non affectés dans la solution initiale. Une fois la liste tabou remplie, on vide sa moitié la plus ancienne. Par exemple, la taille de la liste tabou de l'exemple de la figure 3.2 est égale à 14.

3.5 A propos de la complexité

Pour les graphes à largeur d'arbre bornée, la complexité temporelle de l'algorithme 5 est réduite par rapport à une heuristique constructive équivalente.

La complexité temporelle pour calculer une séparation de clusterss basée sur un critère d'ordre dépend du nombre d'objets n, du nombre de clusters N dans la décomposition arborescente ainsi que de sa largeur w. Dans le cas d'un critère local, la complexité temporelle de cette phase est de l'ordre de $\mathcal{O}(N \times \log(w))$, l'équivalent de $\mathcal{O}(n \times \log(n))$ pour un graphe arbitraire. Dans le cas d'un critère global, la complexité temporelle est de l'ordre de $\mathcal{O}(n^2)$, car une complexité de $\mathcal{O}(n^2)$ est nécessaire pour calculer la taille du voisinage d'un cluster ou sa demande.

Dans le cas d'un graphe pour lequel il existe un algorithme capable de trouver

une décomposition arborescente dont la largeur est bornée par une constante, la complexité temporelle serait de l'ordre $\mathcal{O}(1)$ (étant donné que le nombre d'objets dans les sous-problèmes est bornée par une constante).

La complexité temporelle de l'heuristique d'amélioration est de l'ordre de $\mathcal{O}(|I^{\mathcal{B}}| \times n^2)$ où $I^{\mathcal{B}}$ est l'ensemble d'objets placés dans l'ensemble $\mathcal{B}$ de bins. La complexité temporelle est induite par le nombre maximum de placements possibles à chaque étape de l'algorithme de packing. Cette complexité peut être réduite en considérant un nombre constant d'objets dans $I^{\mathcal{B}}$ et un nombre de bins ouverts tels que le nombre d'objets placés soit aussi borné par une petite constante. Dans ce cas, la complexité temporelle de cette phase est de l'ordre $\mathcal{O}(1)$. Les expérimentations ont montré que cette technique dégrade l'efficacité de la phase d'amélioration.

Pour conclure, nous proposons d'étudier le cas de l'algorithme 5 en utilisant MCS pour calculer une décomposition arborescente, un algorithme glouton basé sur un critère local pour calculer une séparation de clusterss, BLC pour résoudre les sous-problèmes induits par les clusters résultants et l'heuristique d'amélioration de la section 3.2.4, la complexité temporelle totale est de l'ordre $\mathcal{O}(m + n \times \log(n) + n \times w^3)$.

Si la largeur de la décomposition obtenue par MCS est bornée par une constante, la complexité temporelle devient $\mathcal{O}(m + n \times \log(n))$, tandis que les algorithmes constructives sont au moins de l'ordre de $\mathcal{O}(n^3)$. Pour les grandes instances, l'algorithme de tri peut être évité, ce qui aboutit à une complexité temporelle linéaire tout en sachant que la qualité de la solution obtenue risque d'être de mauvaise qualité.

3.6 Partie expérimentale

Dans cette section, nous présentons les résultats numériques obtenus par les différents algorithmes décrits dans ce chapitre. Toutes les méthodes ont été codées en C++ et exécutées sur un processeur Intel(R) Core(TM)2 Duo CPU T7500 @ 2.20GHz. Pour la recherche tabou, nous avons utilisé les composants logiciels de la plateforme ParadisEO [10]. Dans ces expérimentations, le temps de calcul de la recherche tabou ainsi que de la méthode de relance aléatoire a été fixé à 60 secondes. Les instances utilisées sont celles générées pour le $\mathcal{BPPC}$-$2\mathcal{D}$ et décrites dans le chapitre 2.

On dénote par BLC-TD (resp. H6-2D-TD) l'algorithme complet qui consiste à appliquer l'algorithme 5 en utilisant BLC (resp. H6-2D) comme une méthode de résolution (ligne 5) et ensuite améliorer la solution finale comme on l'a expliqué dans la section 3.2.4. Les deux heuristiques BLC-TD et H6-2D-TD ont été utilisées dans une procédure de relance aléatoire appelée RMS-TD. L'idée de RMS-TD est simple, à chaque itération, chacun des clusters est résolu par BLC-TD et H6-2D-TD et la meilleur résultat est gardé.

3.6.1 L'impact de la décomposition arborescente

Dans le tableau 3.1 nous comparons les résultats obtenus par deux heuristiques constructives (BLC et H6-2D), trois heuristiques constructives basées sur une décomposition arborescente (BLC-TD, H6-2D-TD et RMS-TD) et une recherce tabou basée aussi sur une décomposition arborescente (TS-TD).

TABLE 3.1 – *Résultats de nos méthodes pour les instances de [6, 86] (classes 1, 3, 5, 7, 8, 9 et 10). Chaque ligne montre les résultats moyens sur 50 instances. La ligne* Avg *montre les résultats moyens et les temps de calcul moyen sur* 1800 *instances. La ligne* Ttl *montre le nombre total de solutions optimales atteintes (i.e. pour lesquelles $UB = LB$) par les différents algorithmes.*

		BLC			H6-2D			BLC-TD			H6-2D-TD			RMS-TD			TS-TD		
Classe	d(%)	*%gap*	*sec.*	*#opt*	*%gap*	*sec.*	*#opt*	*%gap*	*sec.*	*#opt*	*%gap*	*sec.*	*#opt*	*%gap*	*sec.*	*#opt*	*%gap*	*sec.*	*#opt*
I	30	12.36	0.02	10	9.93	0.06	12	8.09	0.07	15	7.9	0.11	15	7.4	41.02	17	7.27	39.75	18
	50	3.2	0.03	13	2.71	0.08	15	1.26	0.08	32	1.1	0.15	33	1.1	26.11	33	0.55	20.41	38
	70	1.22	0.03	29	0.83	0.07	30	0.53	0.05	38	0.51	0.1	38	0.53	15.1	38	0.33	11.55	42
	90	0.27	0.03	44	0.27	0.08	44	0.24	0.02	45	0.24	0.06	45	0.24	7.4	45	0.14	3.37	48
III	30	0.57	0.04	35	0.54	0.04	36	0.46	0.12	40	0.46	0.09	40	0.46	10.21	40	0.46	19.15	41
	50	0.62	0.04	21	0.41	0.07	25	0.27	0.09	37	0.26	0.13	38	0.25	7.18	39	0.23	14.29	43
	70	0.65	0.05	39	0.47	0.04	41	0.37	0.06	44	0.22	0.08	45	0.22	2.0	45	0.13	6.03	47
	90	0.41	0.04	39	0.28	0.04	40	0.23	0.03	42	0.23	0.04	42	0.23	3.7	42	0.16	6.82	44
V	30	13.24	0.04	6	11.66	0.05	8	10.42	0.14	12	9.33	0.18	13	9.33	48	13	8.25	43.01	15
	50	4.75	0.05	9	4.13	0.05	11	2.29	0.12	24	2.01	0.21	26	1.9	31.4	27	1.71	28.49	29
	70	1.32	0.05	31	0.97	0.06	35	0.41	0.1	42	0.41	0.09	42	0.41	5.2	42	0.41	11.26	42
	90	0.11	0.04	37	0.10	0.05	37	0.08	0.02	38	0.08	0.03	38	0.08	9.55	38	0.03	1.1	40
VII	30	20.08	0.04	1	17.4	0.07	1	16.58	0.15	2	15.21	0.54	4	15.21	54.72	4	13.16	50.92	7
	50	3.0	0.05	16	2.84	0.05	17	2.21	0.12	21	2.02	0.23	22	1.86	30.11	25	1.51	33.16	27
	70	0.44	0.05	40	0.42	0.04	41	0.21	0.1	45	0.21	0.06	45	0.21	1.86	45	0.05	4.13	48
	90	0.09	0.04	46	0.07	0.05	47	0.23	0.03	48	0.23	0.03	48	0.23	0.42	48	0.02	1.13	49
VIII	30	19.39	0.05	3	18.0	0.05	3	16.84	0.16	3	15.47	0.2	4	15.47	52.69	4	13.96	46.51	8
	50	3.74	0.05	16	3.12	0.06	19	1.77	0.12	29	1.80	0.17	27	1.7	18.3	31	1.18	20.09	37
	70	0.39	0.05	41	0.29	0.04	42	0.28	0.06	43	0.1	0.08	45	0.08	4.07	46	0.03	3.85	49
	90	0.31	0.04	41	0.31	0.06	41	0.25	0.03	43	0.13	0.04	47	0.13	3.61	47	0.13	4.55	47
IX	30	0.19	0.05	46	0.19	0.04	46	0.0	0.09	50	0.0	0.15	50	0.0	0.72	50	0.0	5.22	50
	50	0.84	0.05	33	0.60	0.04	35	0.12	0.08	46	0.09	0.09	47	0.09	0.38	47	0.09	6.81	47
	70	0.25	0.04	43	0.24	0.05	43	0.16	0.05	48	0.04	0.12	49	0.04	0.09	49	0.04	1.37	49
	90	0.04	0.04	48	0.04	0.04	48	0.2	0.02	45	0.2	0.02	45	0.2	0.17	45	0.0	0.0	50
X	30	16.95	0.04	29	16.81	0.05	29	16.02	0.11	34	15.6	0.17	35	15.6	16.98	35	14.27	21.54	36
	50	0.88	0.04	39	0.88	0.05	39	0.25	0.09	46	0.25	0.13	46	0.25	8.05	46	0.25	7.59	46
	70	1.41	0.05	26	1.3	0.07	26	0.56	0.08	39	0.56	0.10	39	0.56	11.29	39	0.22	13.1	45
	90	0.11	0.04	47	0.11	0.05	47	0.06	0.02	48	0.03	0.03	49	0.03	1.37	49	0.03	3.32	49
Avg		3.82	0.04		3.39	0.05		2.87	0.08		2.67	0.12		2.64	14.71		2.31	15.3	
Ttl				828			858			999			1017			1029			1091

Vu la taille du jeu d'essai, nous présentons des résultats moyennés pour chaque classe d'instances, *i.e.* chaque ligne représente une valeur moyenne pour 50 instances (10 instances pour chaque valeur de n). La colonne *%gap* donne la moyenne du gap, égale à $100(U - L)/U$, où U représente la valeur obtenue par l'heuristique correspondante et L la valeur de notre borne inférieure proposée dans le chapitre 2. La colonne *#opt* indique le nombre total d'instances résolues à l'optimalité, et *sec* est le temps de calcul moyen pour chacun des algorithmes.

Aucune des stratégies de tri ne domine l'autre en moyenne dans le cas BLC-TD. Par conséquent, les résultats rapportés sont obtenus en utilisant toutes les stratégies tour à tour et en gardant le meilleur résultat (on rapporte le temps de calcul total).

Dans le tableau 3.1 nous pouvons remarquer que l'heuristique H6-2D est meilleure que BLC en moyenne, ce qui n'est pas surprenant étant donné que H6-2D utilise des techniques plus avancées que BLC qui est un simple algorithme glouton. La différence est de l'ordre de 0.43% en terme de gap et de 30 en terme de nombre de solutions

optimales trouvées en faveur de H6-2D. De plus, les temps de calcul moyens des deux heuristiques est quasiment le même.

L'efficacité de la nouvelle approche apparaît clairement dans le tableau 3.1. Les résultats obtenus en intégrant une heuristique constructive (*e.g.* BLC ou H6-2D) dans l'algorithme 5 sont meilleurs que ceux obtenus en appliquant seulement l'heuristique constructive. Par exemple, BLC-TD a pu obtenir 171 solutions optimales de plus que BLC, et le gap a été réduit de 0.95%. Néanmoins, le temps de calcul moyen a augmenté, tout en restant raisonnable. Pour les graphes de conflit denses, le temps de calcul est réduit du fait que la largeur de la décomposition arborescente est très petite pour les graphes de compatibilité de faible densité. Les clusters résultants sont de petites tailles et les sous-problèmes correspondant sont résolus rapidement. Il en est de même quand on compare H6-2D-TD avec H6-2D où l'amélioration est de l'ordre de 0.72% en terme de gap et 159 en terme de nombre de solutions optimales.

On peut noter aussi que BLC-TD est meilleure que H6-2D, ce qui veut dire qu'en exploitant notre approche avec une heuristique constructive simple on est compétitif par rapport à une heuristique plus complexe. Cette amélioration est due au fait que la nouvelle approche permet à l'heuristique d'avoir une certaine connaissance sur la structure du graphe. Il en est de même quand on compare H6-2D-TD avec BLC et H6-2D.

Le comportement de RMS-TD est satisfaisant car elle améliore en moyenne le gap et le nombre de solutions optimales atteintes par rapport à BLC, BLC-TD, H6-2D et H6-2D-TD. Toutefois, cette amélioration est accompagnée avec une dégradation du temps étant donné que le temps de calcul de RMS-TD est fixé à 60 secondes. Un temps de calcul plus petit peut dégrader la qualité de la solution.

En considérant TS-TD, l'amélioration est beaucoup plus importante. Elle dépasse tous les autres algorithmes, assurant une gap de 2.31% en un temps moyen de 15 secondes. Le nombre de solutions optimales est égal à 1091 pour un total de 1800 instances.

3.6.2 Résultats sur des instances de grande taille

Dans le cas d'instances de grande taille, le temps de calcul d'une méthode de résolution peut être grand si cette dernière est appliquée sur la totalité de l'instance. Dans cette section, nous montrons le comportement de notre méthode sur ce genre d'instances.

Dans le tableau 3.2 nous considérons deux classes d'instances (1 and 4), deux valeurs de n (1000 et 2000) et 4 valeurs pour la densité d (40%, 70%, 80% et 90%). L'heuristique constructive utilisée est BLC sans appliquer l'heuristique d'amélioration à la solution finale. Il s'agit donc d'appliquer l'algorithme 5 en calculant une décomposition arborescente du graphe de compatibilité, calculer une séparation de clusters et résoudre les sous-problèmes induits par les clusters résultants par BLC. Les solutions partielles sont ensuite fusionnées pour avoir une solution de la globalité de l'instance.

Pour BLC, nous rapportons la borne supérieure (U) obtenue et le temps de calcul (*sec.*). Pour BLC-TD, la colonne *%dégradation* montre le pourcentage de perte en terme de nombre de bins comparé avec la valeur obtenue par BLC, *sec.* est le temps de calcul total, *TD* est le temps nécessaire pour calculer une décomposition arborescente et *résolution_sous-problèmes* est le temps requis pour résoudre les sous-problèmes induits par les clusters. On rappelle que BLC-TD est appliqué sans amélioration de la solution finale.

TABLE 3.2 – *Résultats de nos méthodes pour 8 instances de la classe 1 et 8 instances de la classe 4 [6, 86]. Pour BLC-TD, la colonne* %degradation *montre le pourcentage de perte en terme de nombre de bins par rapport à la valeur obtenue par BLC,* sec. *est le temps de calcul total,* TD *est le temps requis pour calculer une décomposition arborescente et* solving_clusters *est le temps requis pour résoudre les sous-problèmes induits par les différents clusters.*

			BLC		BLC-TD sans améliorer la solution finale			
Classe	n	d(%)	*U*	*sec.*	*%dégradation*	*sec.*	*TD*	*résolution_sous-problèmes*
I	1000	40	441	30,7	41,04	95,22	82,86 (87,02%)	12,36(12,98%)
	1000	70	724	72,91	9,8	18,3	17,07 (93,28%)	1,23(6,72%)
	1000	80	803	74,05	5,6	7,82	7,22 (92,33%)	0,6(7,67%)
	1000	90	915	71	1,85	0,92	0,78 (84,78%)	0,14(15,22%)
	2000	40	884	217.4	42,30	1629	1099 (67,46%)	530 (32,54%)
	2000	70	1388	540,28	12,53	325,8	303.69(93,21%)	22.11(6,79%)
	2000	80	1645	597,72	6,6	90,51	85,93 (94,94%)	4,58 (5,06%)
	2000	90	1812	558,75	3,31	16,56	15,69 (94,75%)	0,87 (5,25%)
IV	1000	40	403	67,58	1,48	278,43	109,13(39,19%)	169,3(60,81%)
	1000	70	700	100,56	0,0	37,54	27,64 (73,63%)	9,9 (26,37%)
	1000	80	811	100,84	0,0	9,54	7,97 (83,54%)	1,57 (16,46%)
	1000	90	900	87,68	0,0	1,46	1,24 (84,93%)	0,22 (15,07%)
	2000	40	813	492,75	1,72	7663,12	1524.53(19,89%)	6138.59(80,11%)
	2000	70	1394	771,05	0,0	684,16	400,54 (58,54%)	283,62(41,46%)
	2000	80	1624	773,5	0,0	150,53	124,89 (82,97%)	25,64(17,03%)
	2000	90	1791	703,51	0,0	26,56	23,34 (87,88%)	3,22 (12,12%)

Dans le tableau 3.2, le *%dégradation* varie selon la densité du graphe de conflit. Pour les instances de grande taille avec un graphe de conflit de petite densité, notre approche n'est pas intéressante, car le graphe de compatibilité étant dense, la largeur de sa décomposition arborescente générée par MCS est presque égale à $n - 1$.

Toutefois, pour les graphes de conflit denses, BLC-TD fonctionne bien et surtout pour les densités 80% et 90%. Pour l'instance de la classe 1 où $n = 1000$ et $d = 90\%$, une perte de 1.85% en terme de qualité de solution entraîne une réduction de 98.71% du temps de calcul. Les instances de la classe 4 montrent que notre approche peut améliorer énormément le temps de calcul sans dégrader la qualité de la solution. Pour l'instance de la classe 4 où $n = 2000$ et $d = 90\%$, la même solution a été obtenue par BLC et BLC-TD mais dans un temps de calcul réduit de 96.23%.

Si les graphes de compatibilité considérés étaient déjà triangulés, seul la colonne "solving_clusters" serait prise en considération. Il en résulterait une grande réduction du temps de calcul même lorsque les graphes de conflit ont une petite densité.

3.7 Conclusion

Dans ce chapitre, nous avons proposé une nouvelle approche basée sur le concept de décomposition arborescente afin d'améliorer l'efficacité des méthodes de résolution dédiées au $\mathcal{BPPC}$-$2\mathcal{D}$. L'approche a été exploitée par des heuristiques, bien que des méthodes exactes et même hybrides peuvent être utilisées.

En résolvant des instances de $\mathcal{BPPC}$-$2\mathcal{D}$ de grande taille ayant un graphe de compatibilité de petite largeur, cette approche s'est révélée très efficace vu la réduction importante qu'elle a pu assurer en terme de temps de calcul. Néanmoins, cette amélioration est accompagnée d'une dégradation en terme de qualité de solution (nombre de bin) si l'heuristique d'amélioration n'est pas appliquée sur la solution finale.

Comme une extension naturelle de ce travail, il serait intéressant de tester l'efficacité de cette approche sur le $\mathcal{BPPC}$-$1\mathcal{D}$. Il serait aussi intéressant de concevoir des méthodes plus complexes pour séparer les clusters, pour résoudre les problèmes induits par les clusters ainsi que des heuristiques d'amélioration de la solution finale. Une piste aussi importante consiste à hybrider cette approche en impliquant des méthodes exactes dans le processus de résolution.

Chapitre 4

Un problème bi-objectif de bin packing avec conflits

Le travail de ce chapitre a été réalisé en collaboration avec Saïd Hanafi de l'équipe de recherche opérationnelle de l'université de Valenciennes.

4.1 Introduction

Ce chapitre est consacré au $\mathcal{BPP}$-$\mathcal{C}$ bi-objectif. Nous proposons des heuristiques et des bornes inférieures pour ce problème.

Dans les approches de la littérature, la majorité des problèmes de bin packing sans/avec conflits mettent l'accent sur la minimisation du nombre de bins. Tandis que, dans les applications de la vie réelle et surtout les applications industrielles, plusieurs critères entrent en jeu. Prenons l'exemple de l'application qui consiste à charger des conteneurs dans des avions cargo [94] : étant donné un nombre de vols limité par jour, chaque vol représente un avion équipé d'un conteneur, il faut trouver une façon de distribuer l'ensemble des produits à transporter sur l'ensemble des conteneurs en respectant les conflits. Le nombre de conteneurs pouvant ne pas être suffisant pour trouver un placement respectant toutes les contraintes de compatibilité entres les produits, il faut donc un placement violant le moins de conflits possible.

Dans ce genre de situations, nous faisons face à un problème de bin packing multi-objectif. Le nombre de bins utilisés par une solution est l'un des critères les plus importants à prendre en considération. Un deuxième critère aussi important que le premier est le nombre de couples d'objets incompatibles placés dans le même bin.

Nous considérons donc le *problème de bin packing bi-objectif avec conflits* où on cherche à :

- **minimiser le nombre de bins utilisés**
  Les objets à placer dans les bins peuvent être vus comme des agents partageant un ensemble de ressources. Ces ressources représentent une certaine charge, probablement variable et croissante, pour l'acteur décisionnel. Elles doivent donc être toutes exploitées.
- **minimiser le nombre de conflits par bin**
  Dans le cas d'une limitation en terme de ressources, celles-ci doivent être gérées

de telle façon à pouvoir recevoir tous les objets. Le nombre de couples en conflits placés dans un même bin devra être alors minimisé.

Nos approches de résolution sont de deux types : celles qui prennent en compte la structure du front Pareto, et celles qui reposent sur la résolution itérative de *problèmes de bin packing avec conflits faibles* ($\mathcal{BPP}$-$\mathcal{SC}$ pour ***B**in **P**acking **P**roblem with **S**oft **C**onflicts*). Le $\mathcal{BPP}$-$\mathcal{SC}$ consiste à minimiser le nombre de conflits violés en rangeant un ensemble $I = \{1, 2, \ldots, n\}$ d'objets dans m bins. Chaque objet i est caractérisé par sa longueur $w_i \leq W$ où W est la longueur du bin. On dispose aussi d'un graphe de conflits $G = (I, E)$ où chaque conflit entre deux objets (i, j) est indiqué par une arête $(i, j) \in E$.

Nous montrons dans un premier temps comment calculer les nombres de bins pour les deux points extrêmes de la frontière Pareto. Nous procédons ensuite à une approche itérative de génération des points de la frontière. Les problèmes pour générer chacun des points sont très difficiles à résoudre, nous allons donc générer des bornes inférieures et supérieures de ces points. Nous obtenons ainsi deux évaluations par excès et par défaut de la frontière Pareto, une frontière inférieure et une frontière supérieure. En particulier, une borne inférieure a été implémentée. Elle est basée sur des formulations mathématiques et la génération de colonnes. Le sous-problème de pricing est résolu d'une manière hybride : des heuristiques gloutonnes rapides et une recherche locale sont utilisées ainsi que CPLEX si nécessaire. L'efficacité de cette hybridation est démontré par les résultats numériques dans la partie expérimentale.

Nous proposons aussi des méthodes basées directement sur le front Pareto : une borne inférieure et une heuristique. Pour cette dernière, nous avons développé une recherche tabou avec une stratégie de diversification qui utilise différents points de la frontière afin de diversifier la recherche.

Le chapitre est organisé comme suit. Dans la section 4.2, nous présentons en détail la nouvelle approche de résolution basée sur le concept de décomposition arborescente. La section 3.3 est consacrée à des heuristiques rapides pour exploiter la décomposition arborescente dans un contexte de bin packing. Afin d'assurer une bonne exploitation de la structure d'une décomposition arborescente, nous proposons une recherche tabou dans la section 3.4. Avant d'aborder les résultats numériques des différentes méthodes dans la section 3.6, une analyse de complexité de l'approche est développée dans la section 3.5.

4.2 Caractérisation de la frontière Pareto et approche de résolution

Dans cette section, nous décrivons en détail le problème bi-objectif. Nous montrons que la frontière Pareto a une structure prticulière qui nous permet de mettre au point une approche de résolution itérative.

4.2.1 Modélisation mathématique

Soit $M = \{1, 2, \ldots, m\}$ un ensemble de m bins indexés de 1 à m. Soit x_{ik} une variable booléenne égale à 1 si l'objet $i \in I$ est placé dans le bin $k \in M$ et 0 sinon. Soit y_k une variable booléenne égale à 1 si le bin $k \in M$ est utilisé et 0 sinon. Les deux fonctions objectif considérées f^1 et f^2 sont décrites formellement comme suit :

$$f^1(x) = \sum_{(i,j)\in E} \sum_{k\in M} x_{ik} \times x_{jk} \tag{4.1}$$

$$f^2(y) = \sum_{k\in M} y_k \tag{4.2}$$

où la fonction quadratique $f^1(x)$ représente le nombre de conflits violés et la fonction linéaire $f^2(x)$ correspond au nombre de bins utilisés. Chacun de ces deux critères étant important, il n'est pas possible d'en privilégier un par rapport l'autre. Le modèle synthétisé du problème de bin packing bi-objectif peut alors être décrit comme suit afin de prendre en compte les deux critères :

$$\begin{aligned}
\min \quad & \langle f^1(x), f^2(y)\rangle && && (4.3)\\
\text{t.q.} \quad & \textstyle\sum_{k\in M} x_{ik} = 1 && \forall i \in I && (4.4)\\
& \textstyle\sum_{i\in I} w_i \times x_{ik} \leq W \times y_k && \forall k \in M && (4.5)\\
& x_{ik} \in \{0,1\} && \forall i \in I, \forall k \in M && (4.6)\\
& y_k \in \{0,1\} && \forall k \in M && (4.7)
\end{aligned}$$

Les contraintes (4.4) sont utilisées pour assurer qu'un objet est placé dans un et un seul bin. Les contraintes (4.5), dites contraintes de sac-à-dos, assurent que la somme des tailles des objets placés dans un même bin ne dépasse pas la taille du bin.

Pour linéariser la fonction f^1, nous proposons la nouvelle fonction suivante :

$$f'^1(x) = \sum_{i\in I} \sum_{j\in I} z_{ij} \tag{4.8}$$

où z_{ij} est une variable booléenne égale à 1 si deux objets en conflit i et j sont placés dans un même bin et 0 sinon.

Un modèle linéaire pour le $\mathcal{BPP}$-$\mathcal{MO}$ suit :

$$\begin{aligned}
\min \quad & \langle f'^1(x), f^2(y)\rangle && && (4.9)\\
\text{t.q.} \quad & \textstyle\sum_{k\in M} x_{ik} = 1 && \forall i \in I && (4.10)\\
& \textstyle\sum_{i\in I} w_i \times x_{ik} \leq W \times y_k && \forall k \in M && (4.11)\\
& z_{ij} \geq x_{ik} + x_{jk} - 1 && \forall (i,j) \in E, \forall k \in M && (4.12)\\
& z_{ij} \in \{0,1\} && \forall i, j \in I && (4.13)\\
& x_{ik} \in \{0,1\} && \forall i \in I, \forall k \in M && (4.14)\\
& y_k \in \{0,1\} && \forall k \in M && (4.15)
\end{aligned}$$

Les valeurs des variables z_{ij} sont contrôlées par les contraintes (4.12).

4.2.2 Frontière Pareto

D'un point de vue multi-objectif, on peut noter que $\mathcal{BPP}$-$\mathcal{MO}$ implique le problème mono-objectif $\mathcal{BPP}$-$\mathcal{SC}$ en fixant la fonction objectif liée au nombre de bins et minimisant la fonction objectif liée au nombre de conflits violés. Ce nombre de bins est borné par deux valeurs connues. La première (resp. deuxième) valeur notée $\mathcal{OPT}(\mathcal{BPP}\text{-}\mathcal{C})$ (resp. $\mathcal{OPT}(\mathcal{BPP})$) est égale au nombre de bins nécessaire pour effectuer un placement respectant (resp. négligeant) toutes les contraintes de compatibilité entre les objets.

On note $\mathcal{X}$ l'ensemble des solutions potentielles du problème, et $\mathcal{X}^*$ l'ensemble des solutions Pareto optimales. A chaque point $x \in \mathcal{X}$ correspond un point réalisable dans l'espace objectif $\mathcal{Z}$. L'ensemble $\mathcal{Z}^*$ des points optimaux dans l'espace objectif $\mathcal{Z}$ représente la frontière Pareto du problème.

Théorème 4.1 *Pour $\mathcal{BPP}$-$\mathcal{MO}$, la cardinalité de $\mathcal{Z}^*$ est calculée comme suit*

$$|\mathcal{Z}^*| = \mathcal{OPT}(\mathcal{BPP}\text{-}\mathcal{C}) - \mathcal{OPT}(\mathcal{BPP}) + 1. \tag{4.16}$$

Démonstration. Comme le nombre de bins est un entier, qu'il n'est pas possible de ranger tous les articles dans moins de $\mathcal{OPT}(\mathcal{BPP})$ bins, et qu'on a déjà 0 conflits si on utilise $\mathcal{OPT}(\mathcal{BPPC})$ bins, on a directement $|\mathcal{Z}^*| \leq \mathcal{OPT}(\mathcal{BPP}\text{-}\mathcal{C}) - \mathcal{OPT}(\mathcal{BPP}) + 1$.

On montre maintenant que chaque valeur de m entre les deux extrêmes conduit à un nombre de conflits différent. On prouve ce résultat par récurrence. Si on considère uniquement $m = \mathcal{OPT}(\mathcal{BPP})$, le résultat est clairement vrai puisqu'il n'y a qu'une valeur de m. Supposons maintenant que le résultat est vrai pour l'ensemble des valeurs de m inférieures à $k < \mathcal{OPT}(\mathcal{BPP})$. A partir de la solution avec k bins, on obtient directement une solution avec $k+1$ bins utilisant un conflit de moins en enlevant un article en conflit dans un bin et en le plaçant seul dans le bin ajouté. Par conséquent, le nombre de conflits nécessaires dans $k+1$ bins est strictement inférieur à celui nécessaire pour k bins, ce qui est le résultat recherché.

□

La figure 4.1 montre la structure spéciale de la frontière pareto du $\mathcal{BPP}$-$\mathcal{MO}$. Nous pouvons remarquer les deux points extrêmes (points en rouge) de la frontière, $\mathcal{OPT}(\mathcal{BPP})$ et $\mathcal{OPT}(\mathcal{BPP}\text{-}\mathcal{C})$. Les autres points verts représentent des valeurs intermédiaires qui correspondent à des solutions du $\mathcal{BPP}$-$\mathcal{SC}$ avec un nombre de bins $m \in [\mathcal{OPT}(\mathcal{BPP}), \ldots, \mathcal{OPT}(\mathcal{BPP}\text{-}\mathcal{C})[$.

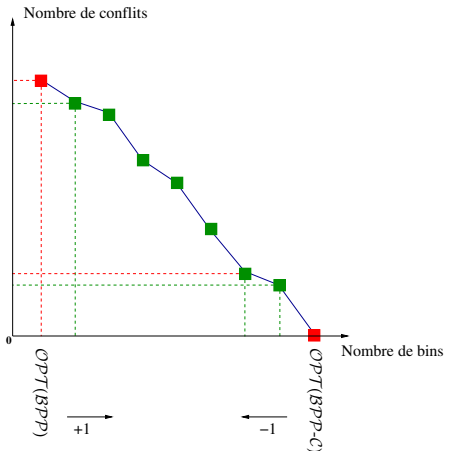


FIGURE 4.1 – *Un exemple de frontière pareto pour le $\mathcal{BPP}$-$\mathcal{MO}$.*

4.2.3 Approche de résolution itérative

Le nombre de points optimaux dans l'espace objectif (égal à $|\mathcal{Z}^*|$) étant connu et faible, une approche de résolution itérative est naturelle pour générer la frontière Pareto.

On commence par calculer les deux points extrêmes $\mathcal{OPT}(\mathcal{BPP})$ et $\mathcal{OPT}(\mathcal{BPP}\text{-}\mathcal{C})$, et ensuite, pour chaque valeur $m \in [\mathcal{OPT}(\mathcal{BPP}), \ldots, \mathcal{OPT}(\mathcal{BPP}\text{-}\mathcal{C})[$, on résout un problème mono-objectif qui consiste à minimiser le nombre de conflits violés en plaçant les n objets dans m bins.

Algorithme 6 : Génération itérative de la frontière Pareto pour le $\mathcal{BPP}$-$\mathcal{MO}$

entrée : une instance de $\mathcal{BPP}$-$\mathcal{MO}$
sortie : la frontière Pareto $\mathcal{Z}^*$

1 $\underline{m} \leftarrow \mathcal{OPT}(\mathcal{BPP}\text{-}\mathcal{C})$;
2 $\overline{m} \leftarrow \mathcal{OPT}(\mathcal{BPP})$;
3 $\mathcal{Z}^* \leftarrow \emptyset$;
4 **pour** *m allant de $\underline{m}$ à $\overline{m}$* **faire**
5 $\quad \mathcal{Z}^* \leftarrow \mathcal{Z}^* \cup \{f^1(x^*), f^2(x^*)\}$ où x^* est une solution optimale d'une instance de $\mathcal{BPP}$-$\mathcal{SC}$ avec $M = \{1, \ldots, m\}$;
6 **retourner** $\mathcal{Z}^*$;

Il y a eu beaucoup de travaux sur le $\mathcal{BPP}$-$\mathcal{C}$ et le $\mathcal{BPP}$, mais à notre connaissance, aucun travail n'a déjà été mené sur le $\mathcal{BPP}$-$\mathcal{SC}$.

4.3 Heuristiques pour le problème de bin packing avec conflits faibles

Beaucoup d'heuristiques ont été développées pour le bin packing avec conflits forts $\mathcal{BPP}$-$\mathcal{C}$, où la fonction objectif consiste à minimiser le nombre de bins nécessaires

pour assurer un placement de tous les objets en respectant toutes les contraintes de compatibilité entre les objets (*e.g.* [53, 2], voir chapitre 1 pour plus de détails).

Ces heuristiques ne prennent pas donc compte le fait que certains conflits peuvent être violés lorsque le nombre de bins est limité. Afin d'utiliser ces heuristiques pour le cas de $\mathcal{BPP}$-$\mathcal{SC}$, nous fixons une borne supérieure sur le nombre de bins.

4.3.1 Heuristiques de type Any-fit

Les objets sont triés selon un ordre donné. Suivant cet ordre, chaque objet est placé dans le bin selon une stratégie de type *any-fit*. Les différentes stratégies dont on cite *first-fit*, *next-fit*, *best-fit* et *worst-fit* sont discutées en détail dans la section 1.4.4.1. Dans la stratégie *best-fit* par exemple, le bin choisi est celui ayant la capacité résiduelle la plus petite et pouvant accueillir l'objet en cours sans qu'il soit en conflit avec les objets déjà placés dedans ; si aucun bin n'est convenable, l'objet sera placé dans un nouveau bin jusqu'à ce que le nombre maximum de bins est atteint ; dans ce cas, l'objet est placé dans le bin causant le moins de conflits.

4.3.2 Heuristiques à base de coloration de graphes

Nous proposons maintenant d'une manière détaillée les deux principales méthodes de réparation proposées pour la résolution des problèmes SGC.

4.3.2.1 Min-conflit

La première méthode de réparation est la méthode *Min-Conflit* (MC) qui date de 1990 [91]. Elle s'inspire d'un réseau de neurones utilisé pour l'ordonnancement des prises de vue du téléscope spatial Hubble [1].

Une itération de MC consiste à choisir aléatoirement l'une des variables en conflit, puis à remplacer sa valeur courante par la nouvelle valeur qui minimise le nombre de contraintes violées. Dans le cas particulier où toutes les variables conduisent à une dégradation du coût, on conserve la valeur courante.

Nous observons que MC est une simple méthode de descente qui améliore la solution à chaque itération.

Algorithme 7 : Méthode de Min-Conflit

entrée : un ensemble $I = \{1, 2, \ldots, n\}$ de n objets, un ensemble $M = \{1, 2, \ldots, m\}$ de m bins, une solution initiale s ayant k conflits où chaque élément s_i de s, lié à un objet $i \in I$, est égal à l'indice du bin où il est placé.

sortie : solution améliorée s

```
1  tant que critère_arrêt and k > 0 faire
2      choisir aléatoirement i ∈ I t.q. ∃j ∈ I, (i,j) ∈ E, s_i = s_j;
3      max_gain ← −1;
4      meilleure_valeur ← −1;
5      meilleur_k ← −1;
6      pour chaque valeur possible l de s_i faire
7          s' ← s;
8          s'_i ← l;
9          nb_conflits ← evaluer_solution(s');
10         si max_gain < k - nb_conflits alors
11             max_gain ← k − nb_conflits;
12             meilleure_valeur ← l;
13             meilleur_k ← nb_conflits;
14     si meilleur_k < k alors
15         s_i ← meilleure_valeur;
16         k ← meilleur_k;
17 retourner s;
```

4.3.2.2 GSAT

La méthode *greedy satisfiability* (GSAT) a été proposée en 1992 pour le *problème de satisfaction de contraintes* (SAT) [105]. Une itération de GSAT consiste à inverser la valeur de la variable booléenne en conflit de manière à obtenir la plus grande diminution possible du nombre de clauses non satisfaites. Lorsqu'aucun de ces mouvements ne permet de diminuer le nombre de clauses non satisfaites, on accepte d'effectuer un mouvement qui ne dégrade pas. La méthode GSAT consiste à effectuer un nombre

fixé d'itérations pour améliorer une solution construite initialement par une méthode constructive gloutonne.

Algorithme 8 : Méthode de GSAT

entrée : un ensemble $I = \{1, 2, \ldots, n\}$ de n objets, un ensemble $M = \{1, 2, \ldots, m\}$ de m bins, une solution initiale s ayant k conflits où chaque élément s_i de s, lié à un objet $i \in I$, est égale à l'indice du bin où il est placé.

sortie : solution améliorée s

```
1  tant que critère_arrêt & k > 0 faire
2      max_gain ← −1;
3      meilleure_variable ← −1;
4      meilleure_valeur ← −1;
5      meilleur_k ← −1;
6      pour chaque variable i ∈ I faire
7          choisir aléatoirement l ∈ M;
8          s' ← s;
9          s'_i ← l;
10         nb_conflits ← evaluer_solution(s');
11         si max_gain < k - nb_conflits alors
12             max_gain ← k − nb_conflits;
13             meilleure_s ← s';
14             meilleur_k ← nb_conflits;
15     si meilleur_k < k alors
16         s ← meilleure_s;
17         k ← meilleur_k;
18 retourner s;
```

4.3.3 Technique de pondération

Une autre technique qui peut être utilisée pour résoudre le $\mathcal{BPP}$-$\mathcal{SC}$ est la technique de pondération. La *méthode de pondération* a déjà été utilisée par Fernandes Muritiba *et al.* [2] pour résoudre le $\mathcal{BPP}$-$\mathcal{C}$.

On utilise ici une fonction de coût dans laquelle l'indice de l'objet et celui du bin qui vont représenter le meilleur couple *(objet, bin)* sont pondérés. L'exécution de l'algorithme consiste en une suite d'itérations. Après chaque itération, on incrémente la pondération de chacun de ces indices. Du fait de cette modification, la configuration courante cesse généralement d'être un optimum local, ce qui permet à la recherche de se poursuivre.

L'idée de la méthode est simple et détaillée étape par étape dans l'algorithme 9.

Algorithme 9 : Technique de pondération

entrée : un ensemble $I = \{1, 2, \ldots, n\}$ de n objets, un ensemble $M = \{1, 2, \ldots, m\}$ de m bins, une solution initiale s ayant k conflits où chaque élément s_i de s, lié à un objet $i \in I$, est égal à l'indice du bin où il est placé.

sortie : solution améliorée s'

```
1  tant que α = 0.0 ≤ 1 faire
2      trier_I_par_ordre_décroissant_selon(α × d_i + (1 − α) × w_i);
3      tant que β = 0.0 ≤ 1 faire
4          créer m bins vides;
5          pour chaque objet i ∈ I faire
6              gain ← ∞;
7              meilleur_bin ← −1;
8              pour chaque bin b ∈ B faire
9                  K_b ← nombre_conflit_dans_bin(b ∪ {i});
10                 W* ← W − Σ_{j∈b} w_j − w_i;
11                 si W* < 0 alors
12                     passer au bin suivant;
13                 si gain > ⌊β⌋ · K_b + ⌊1 − β⌋ · W* + β · (1 − β) · (K_b − w_i/W*) alors
14                     gain = ⌊β⌋ · K_b + ⌊1 − β⌋ · W* + β · (1 − β) · (K_b − w_i/W*);
15                     meilleur_bin ← b;
16             placer l'objet i dans le meilleur bin trouvé;
17         β ← β + 0.1;
18     α ← α + 0.1;
```

Le choix de l'objet à placer est établi selon le critère suivant :

$$\alpha \times d_i + (1 - \alpha) \times w_i \tag{4.17}$$

où $0 \leq \alpha \leq 1$ est une valeur réelle à pas progressif de l'ordre de 0.1, et d_i est le degré du sommet i dans le grahe G. Le bin accueillant l'objet choisi selon (4.17) est déterminé selon le critère suivant :

$$\lfloor \beta \rfloor \times K_b + \lfloor 1 - \beta \rfloor \times W^* + \beta \times (1 - \beta) \times (K_b - \frac{w_i}{W^*}) \tag{4.18}$$

où $0 \leq \beta \leq 1$ est une valeur réelle à pas progressif de l'ordre de 0.1. La variable K_b est égale au nombre de conflits violés au sein du bin b y compris ceux causés par i. La valeur de W^* est égale à la capacité résiduelle dans le bin b après avoir placé i dedans.

4.4 Génération de colonnes pour le problème de bin packing avec conflits faibles

Dans cette section, nous proposons un algorithme de génération de colonnes pour le $\mathcal{BPP}$-$\mathcal{SC}$ qui permet d'obtenir des bornes inférieures pour le problème. Nous nous concentrons sur la résolution du sous-problème de pricing pour le quel nous proposons deux modèles mathématiques, une heuristique et une méta-heuristique.

4.4.1 Modèle à base d'un problème de recouvrement d'ensemble

La relaxation linéaire du modèle (4.10)-(4.15) étant de mauvaise qualité, nous proposons un modèle alternatif basé sur la décomposition de Dantzig-Wolfe [29]. Soit P un ensemble de *patterns* possibles, *i.e.* l'ensemble des placements possibles de placer d'objets dans un seul bin. Chaque pattern est décrit par une colonne $p = (a_{1p}, \ldots, a_{ip}, \ldots, a_{|I|p})^T$, où a_{ip} est égal à 1 si l'objet i est présent dans le pattern p et 0 sinon. Le nombre de conflits dans un pattern p est noté K_p. Le modèle décomposé est le suivant

$$\begin{aligned}
\min \quad & \textstyle\sum_{p\in P} \lambda_p K_p && && (4.19)\\
\text{s.t.} \quad & \textstyle\sum_{p\in P} a_{ip}\lambda_p \geq 1 && \forall i \in I && (4.20)\\
& \textstyle\sum_{p\in P} \lambda_p = M && && (4.21)\\
& \lambda_p \in \{0,1\} && \forall p \in P. && (4.22)
\end{aligned}$$

En pratique, on remplace 4.21 par $\sum_{p\in P} \lambda_p \leq M$.

Le nombre de patterns possibles peut être très grand, ainsi même résoudre la relaxation linéaire du modèle (4.19)-(4.22) peut s'avérer très difficile. Nous présentons dans cette section une méhode de génération de colonnes pour résoudre ce modèle. En remplaçant la contrainte d'intégralité (4.22) par

$$\lambda_p \geq 0, \quad \forall p \in P, \tag{4.23}$$

nous obtenons la relaxation LP du modèle linéaire (4.19)-(4.22). Soit z^* la valeur optimale de cette relaxation LP. Nous obtenons une borne inférieure valide, dite LB_{bpsc}, pour le $\mathcal{BPP}$-$\mathcal{SC}$ en majorant z^* :

$$LB_{\text{bpsc}} = \lceil z^* \rceil. \tag{4.24}$$

Le modèle (4.19), (4.20), (4.21) et (4.23) a un nombre de variables exponentiel égal au nombre de sous-ensembles de I qui représente un pattern valide. La méthode de génération de colonnes est donc une méthode efficace pour générer de nouvelles variables qui sont utiles parmi le nombre exponentiel de variables existant.

Pour résoudre ce modèle, dit problème maître dans la littérature, on l'initialise par un ensemble de colonnes issues de la résolution de l'instance traitée par une heuristique. On résout ensuite le modèle par CPLEX pour obtenir les valeurs optimales $\pi_i, i \in I$, des variables duales associées aux contraintes (4.20). L'information duale est passée à un sous-problème de pricing qui sera chargé de déterminer s'il existe encore des colonnes qui peuvent être ajoutées au problème maître pour améliorer la solution courante. Le processus est arrêté une fois qu'aucune colonne n'est trouvée par le pricing.

4.4.2 Résoudre le sous-problème de pricing

Dans cette section, nous proposons deux modèles mathématiques pour résoudre le sous-problème de pricing. Si la solution optimale d'un modèle de pricing a une valeur inférieure à 0, la colonne trouvée a un coût réduit négatif et elle sera ajoutée au problème maître ; sinon, l'optimalité du problème maître est atteinte.

Pour résoudre le sous-problème de pricing, on a recours dans un premier temps à un ensemble de méthodes heuristiques rapides afin d'éviter l'utilisation de CPLEX qui peut être coûteuse en terme de temps de calcul. Ces algorithmes sont expliqués en détail dans la suite de cette section. Si ces méthodes ne réussissent pas à générer une colonne réalisable, le pricing est donc résolu en utilisant CPLEX. Pour réduire le temps de calcul, nous stoppons le solveur après la première colonne à coût réduit trouvée. Si le pricing est résolu à l'optimalité et aucune colonne à coût réduit n'a été trouvée, la solution optimale de la relaxation LP du programme maître est atteinte et le processus de génération de colonnes est arrêté.

4.4.2.1 Un modèle simple

Pour générer la meilleure colonne afin de l'ajouter à la base courante, le modèle non linéaire suivant est à résoudre.

$$\max \quad \sum_{i\in I}(\pi_i a_i - \sum_{j\in N(i)} a_i a_j) - \theta \tag{4.25}$$

$$\text{s.t.} \quad \sum_{i\ \in I} a_i w_i \leq W \tag{4.26}$$

$$a_i \in \{0,1\} \qquad \forall i \in I \tag{4.27}$$

Il s'agit d'une généralisation du problème de sac-à-dos, où chaque conflit entre deux objets réduit la valeur de la solution d'une unité. Pour chaque variable $i \in I$, $N(i)$ représente l'ensemble des voisins de i d'indice supérieur à i dans le graphe de conflits G.

Le modèle (4.25)-(4.27) est facilement linéarisé en introduisant une variable booléenne b_{ij} qui prend la valeur 1 si deux objets en conflits i et j sont placés dans le même bin et 0 sinon. Le modèle linéaire est le suivant :

$$\max \quad \sum_{i\in I}(\pi_i a_i - \sum_{j\in N(i)} b_{ij}) - \theta \tag{4.28}$$
$$\text{s.t.} \quad \sum_{i\ \in I} a_i w_i \leq W \tag{4.29}$$
$$b_{ij} \geq a_i + a_j - 1 \qquad \forall i \in I, j \in N(i) \tag{4.30}$$
$$b_{ij} \in \{0,1\} \qquad \forall i \in I, j\ \in N(i) \tag{4.31}$$
$$a_i \in \{0,1\} \qquad \forall i \in I \tag{4.32}$$

Le modèle (4.28)-(4.32) peut être vu comme un problème de sac-à-dos binaire avec conflits pénalisés. Les profits des objets sont les valeurs de π_i et les conflits entre les objets sont imposés par les contraintes (4.30). Les pénalités sont imposées par les valeurs des variables b_{ij} où chaque couple d'objets en conflit placés dans le sac réduit la valeur de la solution d'une unité.

4.4.2.2 Une formulation améliorée

Nous proposons maintenant une deuxième formulation pour le sous-problème de pricing. Dans ce modèle, la variable b_i représente le nombre d'objets d'indice supérieur en conflit avec l'objet i. Cette variable est égale à 0 si i n'est pas dans le bin.

$$\max \qquad \sum_{i\in I}(\pi_i a_i - b_i) - \theta \tag{4.33}$$
$$s.t. \qquad \sum_{i\ \in I} a_i w_i \leq W \tag{4.34}$$
$$\sum_{j\in N(i)} a_j - |N(i)| \times (1 - a_i) \leq b_i \quad \forall i \in I \tag{4.35}$$
$$a_i \in \{0,1\} \qquad \forall i \in I \tag{4.36}$$
$$b_i \in \mathbb{N} \tag{4.37}$$

On fixe $b_i = 0$ si $\pi_i < 1$.

4.4.2.3 Heuristiques pour résoudre le sous-problème de pricing

Dans cette section nous proposons des algorithmes pour résoudre le sous-problème de pricing. Il s'agit d'un algorithme glouton et d'un algorithme à base de recherche locale.

Ces algorithmes disposent en entrée d'un vecteur d'objets i chacun de taille w_i, d'un vecteur de valeurs duales π où chaque valeur π_i est liée à un objet i, et de la valeur de θ. Ils doivent fournir en sortie un vecteur représentant un pattern valide de coût réduit négatif à ajouter au problème maître restreint.

4.4.2.3.1 Algorithme glouton

L'idée de cet algorithme est de définir un ordre de traitement sur les objets à placer dans le sac. Différentes stratégies de tri ont été testées :

- ordre décroissant des π_i / w_i
- ordre décroissant des $(\pi_i + deg(i))/w_i$
- ordre décroissant des $(\pi_i + \sum_{j \in N(i)} \pi_j)/w_i$

Les stratégies impliquant les degrés des sommets ont été testées d'une façon statique et dynamique. Dans un mode statique, le degré d'un sommet est le même tout au long de l'exécution de l'algorithme. Au contraire, dans un mode dynamique, le degré d'un sommet change au cours de l'exécution de l'algorithme.

Algorithme 10 : Résolution gloutonne du sous-problème de pricing

entrée : un ensemble d'objets I, une taille de bin W, un vecteur de valeurs duales π, un graphe de conflits $G = (I, E)$ et une valeur θ
sortie : le pattern trouvé et la valeur de la solution

```
1  trier_I_selon(critère de tri);
2  valeur ← 0;
3  pattern ← ∅;
4  pour i = 1 → n faire
5      si W > w_i alors
6          valeur ← π_i;
7          pattern ← pattern ∪ {i};
8          W ← W − w_i;
9  pour chaque {i, j} ∈ pattern faire
10     si (i, j) ∈ E alors
11         valeur ← valeur − 1;
12 valeur ← valeur − θ;
13 retourner pattern et valeur;
```

4.4.2.3.2 Recherche locale

Dans le cas où la solution trouvée par l'algorithme glouton n'est pas de coût réduit négatif, une méthode de recherche locale est appliquée sur cette solution afin d'essayer de trouver une solution à coût réduit négatif en appliquant une séquence de *mouvements*. Pour implémenter cette méthode, nous avons utilisé les composants logiciels de la plateforme ParadisEO [10].

Les deux types de mouvements utilisés sont le mouvement de type *bit flip* et le mouvement de type *pairwise exchange*. Un mouvement bit flip consiste à inverser la valeur a_i d'un objet i ($a_i \leftarrow 1 - a_i$) et un mouvement pairwise exchange consiste à remplacer un objet i par un autre objet j.

L'idée de cette méthode détaillée dans l'algorithme 11 est la suivante. A chaque itération, le premier mouvement améliorant le coût de la colonne courante est ap-

pliqué. Ce processus est itéré tant qu'il existe un mouvement réduisant le coût de la colonne.

Algorithme 11 : Recherche locale pour résoudre le sous-problème de pricing

entrée : un pattern p de valeur val, une taille de bin W, un vecteur de valeurs duales π, un graphe de conflits $G = (I, E)$ et une valeur θ
sortie : le pattern trouvé et la valeur de la solution

1 **répéter**
2 **pour** $i = 1 \rightarrow n$ **faire**
3 **pour** $j = 1 \rightarrow n$ **faire**
4 **si** $i = j$ **alors**
 `// swap(1-0) "bit flip"`
5 **si** $p_i = 0$ **alors**
6 $val^{'} \leftarrow val + \pi_i - \sum_{k \in I \setminus i} a_k \times E(i,k)$;
7 **sinon**
8 $val^{'} \leftarrow val - \pi_i + \sum_{k \in I \setminus i} a_k \times E(i,k)$;
9 **sinon**
 `// swap(1-1) "pairwise exchange"`
10 **si** $p_i \neq p_j$ **alors**
11 **si** $p_i = 0$ **alors**
12 $val^{'} \leftarrow val + \pi_i - \pi_j - \sum_{k \in I \cup \{j\} \setminus \{i\}} a_k \times E(i,k) + \sum_{k \in I \setminus j} a_k \times E(i,k)$;
13 **sinon**
14 $val^{'} \leftarrow val - \pi_i + \pi_j + \sum_{k \in I \cup \{j\} \setminus \{i\}} a_k \times E(i,k) - \sum_{k \in I \setminus j} a_k \times E(i,k)$;
15 **sinon**
16 **continuer**;
17 **si** $val^{'} > val$ **alors**
18 $val \leftarrow val^{'}$;
19 $p_i = 1 - p_i$;
20 **si** $i \neq j$ **alors**
21 $p_j = 1 - p_j$;
22 **passer à la ligne 1**;
23 **jusqu'à** $val^{'} = val$;
24 **retourner** p ***et*** val;

4.5 Méthodes basées sur la frontière Pareto

Lorsqu'on cherche à générer tous les points de la frontière pareto, on peut exploiter la structure particulière de cette frontière pour concevoir des méthodes de résolution. En particulier, nous proposons une borne inférieure très simple ainsi qu'une méta-heuristique qui tire de l'information de l'ensemble des points de la frontière pour l'améliorer localement.

4.5.1 Une borne inférieure basée sur la frontière Pareto

L'ensemble $\mathcal{Z}^*$ des points optimaux dans l'espace objectif $\mathcal{Z}$ représente la frontière Pareto du problème. Notons $\mathcal{Z}_m^*$ la valeur optimale du nombre de conflits lorsqu'on utilise m bins. Par définition du problème $\mathcal{BPP}$-$\mathcal{C}$, si $m = \mathcal{OPT}(\mathcal{BPP}\text{-}\mathcal{C})$, on a $\mathcal{Z}_m^* = 0$. On a remarqué précédemment qu'ajouter un bin à une solution optimale permettait de supprimer au moins un conflit. Cette considération nous permet d'obtenir une borne inférieure très simple pour le nombre de conflits nécessaires lorsqu'on a m bins, basée sur la solution d'autres points de la frontière Pareto.

Proposition 4.5.1 *Pour* $\mathcal{OPT}(\mathcal{BPP}) \geq m > k \geq \mathcal{OPT}(\mathcal{BPP}\text{-}\mathcal{C})$, $\mathcal{Z}_m^* \geq \mathcal{Z}_k^* + m - k$.

Démonstration. On a montré dans le théorème 4.1 qu'ajouter un bin permet de supprimer au moins un conflit. Par récurrence, on obtient directement le résultat recherché. □

En se basant sur ce résultat, connaître la valeur optimale de $\mathcal{BPP}$-$\mathcal{C}$permet de calculer directement des bornes inférieures pour toutes les valeurs de valides de m. Au cours de la recherche, toute amélioration d'une borne inférieure peut être propagée vers les solutions utilisant moins de bins.

4.5.2 Une recherche tabou basée sur la frontière Pareto

Notre recherche locale commence par une solution initiale construite par une des heuristiques constructives décrites dans les sections précédentes, et visite itérativement les solutions voisines en essayant de réduire le nombre de conflits. La recherche locale essaye de réparer la solution courante en s'appuyant sur une fonction d'évaluation qui compte le nombre total de conflits violés. Nous avons utilisé les composants logiciels de la plateforme ParadisEO [10].

4.5.2.1 Configuration et espace de solutions

Soit b_i $(i \in I)$ l'indice du bin où il faut placer l'objet i et M un ensemble de bins disponibles numérotés de 1 à m. Une configuration $s = \langle b_1, b_2, \ldots, b_n \rangle$ est un placement de tous les objets qui satisfait la condition suivante :

$$\forall j \in M, \sum_{i \in \{1,\ldots,n\}; b_i = j} w_i \leq W \tag{4.38}$$

Dans une recherche locale, un espace de solution $\mathcal{S}$ est défini comme l'ensemble des solutions possibles. L'espace de solutions de notre recherche tabou ne comprend que des solutions complètes et faisables. La figure 4.2 montre l'espace de solution de notre recherche tabou.

L'espace de recherche S est composé de toutes les configurations possibles vérifiant la contrainte (4.38). La taille de S est bornée par $\prod_{i \in I}(|M|)$.

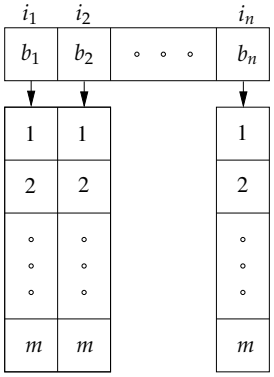


FIGURE 4.2 – *Un exemple d'espace de recherche qui correspond à une instance de n objets et m bins. Pour chaque objet i, on crée une liste contenant l'ensemble des bins pouvant accueillir i.*

4.5.2.2 Fonction d'évaluation

Nous définissons une fonction objectif $f(s)$ pour chaque configuration $s \in \mathcal{S}$, mesurant le degré de conflit comme étant le poids total des conflits violés divisé par le poids total des arêtes :

$$f(s) = \frac{|(i,j) \in E; b_i = b_j|}{|E|} \tag{4.39}$$

4.5.2.3 Exploration du voisinage et évaluation incrémentale

Un voisin de s est obtenu en échangeant le placement d'un objet dans s de telle façon que le nouveau placement satisfait toujours la contrainte (4.38).

Le voisinage est exploré d'une façon déterministe et le meilleur des voisins qui sera choisi remplace la solution courante.

L'évaluation de la solution courante après avoir effectué un mouvement est faite d'une façon incrémentale. Un mouvement déplaçant un objet i d'un bin b_1 vers un bin b_2 est évalué de la façon suivante :

$$\Delta f^i_{b_1,b_2} = -\frac{|(i,q) \in E \mid q \in b_1 \setminus \{i\}|}{|E|} + \frac{|(i,q) \in E \mid q \in b_2|}{|E|}. \tag{4.40}$$

4.5.2.4 Liste tabou

Dans notre implémentation, nous utilisons une liste tabou contenant l'ensemble des mouvements qui sont interdits pour une certaine durée. Pratiquement parlant, cette liste appelée *tabu* est une matrice de $|I|$ lignes et de $|M|$ colonnes. La case $tabu[i][j]$ correspond au mouvement qui consiste à placer l'objet $i \in I$ dans le bin $j \in M$. A chaque fois que le mouvement (i, j) est effectué, la case correspondante est mise à jour de la façon suivante

$$tabu[i][j] = |I| \times |M|. \tag{4.41}$$

De cette façon, le mouvement (i, j) pendant les $|I| \times |M|$ itérations suivantes ou jusqu'à ce que son statut tabou soit levé.

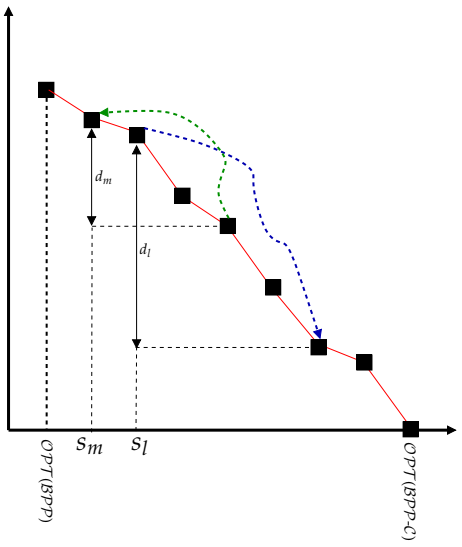
4.5.2.5 Diversification et intensification

Deux stratégies de diversification ont été conçues et intégrées au sein de la recherche tabou. Ces stratégies exploitent le même type d'information pour accomplir leur objectif. Diversifier une solution s_m sur un point $m \in [\mathcal{OPT}(\mathcal{BPP}),\mathcal{OPT}(\mathcal{BPP}\text{-}\mathcal{C})]$ peut prendre deux formes, une diversification de type *top-down* ou une diversification de type *bottom-up*. Dans une diversification top-down (resp. bottom-up), une solution s_m est remplacée par une solution s_{m+d} (resp. s_{m-d}) où $d \in]0,\mathcal{OPT}(\mathcal{BPP}\text{-}\mathcal{C})-m]$ (resp. $d \in]0, m-\mathcal{OPT}(\mathcal{BPP})]$).

Ces stratégies de diversification peuvent être contrôlées par un paramètre dit *distance d*. La distance d'une diversification représente le nombre de bins à ajouter (resp. enlever) à la solution courante dans le cas d'une stratégie top-down (resp. bottom-up). Dans le cas de top-down par exemple, d_m bins sont ajoutés à la solution s_m et donc une solution s_{m+d_m} est obtenue en distribuant le contenu de m bins dans $m + d_m$ bins en utilisant une heuristique classique de bin packing.

La figure 4.3 illustre ces deux stratégies de diversification. La flèche bleue (resp. verte) montre une diversification top-down (resp. bottom-up) d'une solution s_l (resp. s_m) avec une distance d_l (resp. d_m).

FIGURE 4.3 – *Exemple illustrant les deux stratégies de diversification "top-down" et "bottom-up".*



4.6 Partie expérimentale

Nous présentons dans cette section les résultats numériques concernant les différents algorithmes proposés dans ce chapitre. Toutes les méthodes sont codées en C++ et CPLEX 10.0 est utilisé pour résoudre les modèles linéaires des sous-problèmes de pricing. Pour les méthodes de recherche locale, nous avons utilisé les composants logiciels de la plateforme ParadisEO [10]. Les expérimentations sont menées sur une machine équipée d'un processeur Intel (R) Core (TM) 2 6400 @ 2.13 GHz équipée d'une carte mémoire de 2 GO.

Le jeu d'essai utilisé est celui proposé par Fernandes-Muritiba *et al.* [2] pour le $\mathcal{BPPC}$-1$\mathcal{D}$. Ce benchmark et disponible à l'URL `http://www.or.deis.unibo.it/research_pages/ORinstances/ORinstances.htm`

Vu la taille du jeu d'essai et de la taille de la fontière Pareto de chaque instance, on se limite à un sous-ensemble d'instances pour faire des analyses numériques. On considère deux classes d'instances, la classe 5 où $n = 60$ et la classe 1 où $n = 120$. Plusieurs densités de graphe sont considérées $d \in \{40\%,50\%,60\%,70\%\}$. Le nom d'une instance est représenté par "x_y_z" où x est la classe à laquelle appartient l'instance, y la densité du graphe de conflits et z le numéro de l'instance allant de 1 à 10. Une instance x_y_z est équivalente à l'instance $x\_y\_z.BPPC$ du jeu d'essai de Fernandes-Muritiba *et al.* [2]. On note $|\mathcal{Z}^*|$ le nombre de points optimaux dans l'espace objectif et *sec* le temps de calcul moyen requis par chacun des algorithmes.

Dans le tableau 4.1, nous comparons les résultats obtenus par deux heuristiques constructives ("any-fit" et "technique de pondération") et trois recherches locales ("coloration", "méthode de descente" et "recherche tabou"). Parmi ces algorithmes, la recherche locale est le seul à utiliser la frontière Pareto afin d'extraire de l'information pour améliorer les résultats. La colonne "g" donne la moyenne du gap sur la totalité des points de la frontière Pareto évaluée comme suit

$$g = \frac{1}{|\mathcal{Z}^*|} \times \sum_{i=1}^{|\mathcal{Z}^*|} 100(U_i - L_i)/U_i \tag{4.42}$$

où U_i (resp. L_i) représente la valeur obtenue par l'heuristique (resp. la borne inférieure) correspondante sur le point $i \in [\mathcal{OPT}(\mathcal{BPP}),\mathcal{OPT}(\mathcal{BPP}\text{-}\mathcal{C})]$. La colonne s représente la variance de la distance entre les différentes solutions d'une même frontière Pareto. La valeur de s pour une frontière Pareto donnée est évaluée comme suit

$$s = \frac{1}{|\mathcal{Z}^*| - 1} \times \sum_{i=1}^{|\mathcal{Z}^*|-1} U_i - U_{i+1}. \tag{4.43}$$

La colonne N^* indique le nombre de points optimaux atteints par l'heuristique correspondante, et *sec* est le temps de calcul moyen utilisé sur un point de la frontière Pareto. Aucune des deux algorithmes "min-conflit" et "GSAT" ne domine l'autre en

moyenne. Par conséquent, les résultats reportés sont obtenus en utilisant les deux algorithmes et en gardant le meilleur résultat pour un temps de calcul égal à la somme du temps de calcul des deux algorithmes.

Dans le tableau 4.1 nous pouvons remarquer que la recherche tabou est meilleure que tous les autres algorithmes en moyenne, ce qui n'est pas surprenant étant donné qu'elle utilise des techniques plus avancées que celles des autres. Un gap moyen de 8.59% est obtenu en 12.16 secondes ainsi qu'un nombre de solutions optimales égal à 418 sur un total de 920 solutions. La valeur moyenne $s = 1.27$ prouve la qualité de la frontière Pareto obtenue par la recherche tabou. Cette valeur indique que, en moyenne, la différence entre les valeurs de deux solutions consécutives i et $i+1$ est égale à 1.27. De son côté, la méthode de descente a fait preuve d'une technique très efficace pour avoir des solutions rapides de bonne qualité. Cette heuristique consiste à lancer la recherche tabou et à l'arrêter après la première descente. Son gap moyen, plus grand que celui de la recherche tabou, est plus petit que celui des autres et améliore de 11% le meilleur d'entre eux. La valeur moyenne de s pour la méthode de descente confirme la diversité des solutions fournies. L'heuristique any-fit fournit les résultats les moins bons. Cette heuristique a pu assurer un gap moyen de 52.16% et a atteint 57 solutions optimales pour un temps de calcul moyen très faible. La valeur moyenne $s = 3.048$ peut être considérée élevée, ce qui veut dire que les solutions d'une frontière Pareto peuvent être très distantes les unes des autres. La technique de pondération et l'algorithme de coloration viennent entre ces deux algorithmes. Le gap moyen de l'algorithme de coloration est meilleur que celui de la technique de pondération, tandis que le nombre de solutions optimales atteintes par ce dernier algorithme dépasse celui du premier. Ceci est justifié par le fait que les algorithmes de coloration, basés sur des techniques d'amélioration itérative d'une fonction de coût définie en fonction du nombre de conflits violés, sont plus puissants à surmonter les optima locaux tandis que ce n'est pas la cas pour la technique de pondération. Cependant, la technique de pondération, influencée par l'équation (4.18) et l'ordre sur les objets résultant de cette équation, est plus efficace pour trouver des solutions optimales. Selon les valeurs de s pour la technique de pondération et les algorithmes de coloration, on peut dire que les solutions des frontières Pareto de ces algorithmes sont bien réparties. Le temps de calcul des algorithmes de coloration est relativement plus élevé que celui des autres algorithmes sauf la recherche tabou. Ceci est dû au critère d'arrêt utilisé et qui repose sur le nombre d'itérations. Diminuer ce nombre d'itérations peut diminuer le temps de calcul mais sans aucune garantie d'obtenir les mêmes solutions.

Dans les tableaux 4.2 et 4.3 nous comparons les résultats obtenus par les différents algorithmes de génération de colonnes. Un temps limite d'une heure est donné à chacun de ces algorithmes. On note $\mathcal{GCM}_1$ l'algorithme de génération de colonnes utilisant le modèle linéaire de la section 4.4.2.1 pour le sous-problème de pricing résolu par CPLEX uniquement, et $\mathcal{GCM}_1$+$\mathcal{H}$ le même algorithme consistant à résoudre le pricing par les heuristiques décrites dans la section 4.4.2.3 et par CPLEX si nécessaire. L'algorithme $\mathcal{GCM}_2$ utilise le pricing décrit dans la section 4.4.2.2 et $\mathcal{GCM}_2$+$\mathcal{C}$ l'u-

tilise aussi ainsi que les coupes décrites dans la section 4.4.2.2. De même, $\mathcal{GCM}_2$+$\mathcal{H}$ et $\mathcal{GCM}_2$+$\mathcal{C}$+$\mathcal{H}$ consiste à résoudre le pricing par les heuristiques et par CPLEX si nécessaire.

Dans le tableau 4.2 on s'intéresse aux deux algorithmes $\mathcal{GCM}_1$ et $\mathcal{GCM}_2$. Le but de cette comparaison est de montrer l'efficacité de chacun des deux modèles de pricing. Pour chacun des deux algorithmes, on reporte le nombre de colonnes (*nb col*) générées afin de prouver l'optimalité de la borne inférieure ainsi que le temps de calcul requis (*sec.*). Le temps de calcul de l'algorithme $\mathcal{GCM}_2$ est réduit de 71% par rapport à celui de $\mathcal{GCM}_1$. Le nombre de colonnes générées est aussi réduit. Dans la suite, l'algorithme $\mathcal{GCM}_1$ ne sera plus utilisé car $\mathcal{GCM}_2$ fournit les mêmes résultats d'une façon sensiblement plus rapide en moyenne.

Dans le tableau 4.3 on compare les résultats obtenus par les algorithmes $\mathcal{GCM}_1$+$\mathcal{H}$, $\mathcal{GCM}_2$, $\mathcal{GCM}_2$+$\mathcal{H}$, $\mathcal{GCM}_2$+$\mathcal{C}$ et $\mathcal{GCM}_2$+$\mathcal{C}$+$\mathcal{H}$. La colonne (%) *heur* reporte le pourcentage de colonnes générées par les heuristiques pour chacun des algorithmes. En comparant les résultats de $\mathcal{GCM}_1$+$\mathcal{H}$ dans le tableau 4.3 à ceux de $\mathcal{GCM}_1$ dans le tableau 4.2, on peut remarquer une réduction du temps de calcul moyen (1360.47 secondes $\rightarrow$ 649.39 secondes) qui vaut presque 52% du temps de calcul initial bien que le nombre de colonnes générées ait augmenté de 25.58%. De même, le temps de calcul moyen de $\mathcal{GCM}_2$+$\mathcal{H}$ est réduit de 26,63% par rapport à $\mathcal{GCM}_2$ et de 15,24% dans le cas de $\mathcal{GCM}_2$+$\mathcal{C}$+$\mathcal{H}$ comparé à $\mathcal{GCM}_2$+$\mathcal{C}$. On peut conclure que l'idée qui consiste à générer le plus de colonnes possible avec des heuristiques plutôt que CPLEX est une stratégie efficace qui prouve ici sa capacité de réduire le temps de calcul. Les coupes proposées dans la section se sont révélées efficaces vu la réduction du temps de calcul de $\mathcal{GCM}_2$+$\mathcal{C}$ comparé à celui de $\mathcal{GCM}_2$. Pour conclure, on peut noter que $\mathcal{GCM}_2$+$\mathcal{C}$+$\mathcal{H}$ est le meilleur algorithme parmi les 6 algorithmes de génération de colonnes proposées dans ce chapitre.

4.7 Conclusion

Nous avons étudié un problème d'optimisation multi-objectif particulier difficile à résoudre. Nous avons résolu ce problème par des bornes inférieures et supérieures non triviales, impliquant des techniques de raisonnement basées sur la frontière pareto ainsi qu'une recherche locale à base d'oscillation stratégique et une méthode de génération de colonnes.

Les résultats obtenus montrent l'efficacité des algorithmes proposés, qui sont capables d'obtenir de bons résultats. Le sous-problème de pricing s'est révélé très difficile à résoudre. Ce sous-problème est donc un point sur lequel il faudra focaliser afin de pouvoir développer une méthode exacte de type branch & price pour ce problème.

TABLE 4.1 – *Résultats numériques obtenus par chacune des méthodes, any fit, coloration, pondération , descente et tabou*

			any fit				pondération				coloration				descente				tabou			
instance	*n*	$\lvert\mathcal{Z}^*\rvert$	N^*	*g*	s	*sec.*	N^*	*g*	s	*sec.*	N^*	*g*	s	*sec.*	N^*	*g*	s	*sec.*	N^*	*g*	s	*sec.*
5_4_10	60	2	1	40	5	<0.01	2	0	1	0.02	1	10	1.75	0.26	2	0	1	1.02	2	0	1	3.02
5_4_2	60	7	5	4.42	1.17	<0.01	7	0	1	0.02	6	1.11	0.79	1.55	7	0	1	1.74	7	0	1	5.17
5_4_6	60	9	1	53.2	2.12	<0.01	1	33.8	1.5	0.03	1	21.8	1.16	3.22	9	0	1	2.94	9	0	1	8.44
5_4_8	60	4	1	51	2.33	<0.01	1	39.2	1.67	0.02	1	30.2	1.42	0.90	1	30.8	1.67	1.82	1	30.8	1.67	6.71
5_6_10	60	16	1	42.3	1.8	<0.01	3	20.2	1.53	0.03	4	15.7	1.1	5.03	15	0.39	1.07	2.62	16	0	1	10.4
5_6_1	60	10	1	42	1.89	<0.01	2	22.4	2	0.03	3	17.7	1.36	3.03	8	6.3	1.56	2.27	9	1	1.11	9.93
5_6_2	60	16	1	54	1.87	<0.01	1	39.2	1.67	0.03	1	23.3	1.13	6.79	16	0	1	3.28	16	0	1	9.4
5_6_3	60	9	1	62.2	2.62	<0.01	1	54.2	2.12	0.03	1	38.1	1.62	2.45	1	35.9	1.75	2.22	1	31.8	1.25	10.9
5_6_4	60	11	1	58	2	0	2	36.6	1.7	0.0336	2	24.1	1.2	3.63	9	1.74	1.1	2.49	10	0.826	1.1	10.2
5_6_5	60	11	1	37.7	1.9	<0.01	2	19.2	1.4	0.03	2	17.7	1.1	2.86	2	13.8	1.1	1.96	11	0	1	9.36
5_6_6	60	20	1	36.9	1.68	0	6	9.91	1.42	0.03	4	12.2	1.05	7.67	15	1.86	1.11	2.89	19	0.47	1.11	15.4
5_6_7	60	11	1	54.1	2.1	0	2	33.8	1.6	0.03	3	22	1.18	3.83	11	0	1	2.66	11	0	1	8.59
5_6_8	60	17	1	40.8	1.56	0.01	2	13	1.38	0.03	2	13.9	1.05	5.62	15	1.87	1.25	2.51	16	1.18	1.25	10.8
5_6_9	60	11	1	61.5	2.5	0.0	1	45.7	1.8	0.03	1	34.7	1.52	4.28	1	31.7	1.8	3.03	1	20.6	1.5	14.8
5_7_10	60	21	1	41.4	1.75	0.0	2	23.1	1.65	0.03	2	19	1.16	5.98	2	11.5	1.25	2.16	19	1.75	1.25	15.5
5_7_1	60	18	3	25.4	1.59	0.01	14	4.78	1.29	0.03	9	7.74	1.01	4.5	17	0.83	1.18	1.95	18	0	1	9.58
5_7_2	60	20	1	42.4	2	0	1	24.1	1.74	0.03	2	17.3	1.29	8.92	17	2.54	1.42	3.63	17	1.11	1.32	15.9
5_7_3	60	18	1	44	1.82	0.01	2	22	1.59	0.03	4	17.1	1.21	5.64	16	2.24	1.41	2.54	17	1.26	1.29	12.5
5_7_4	60	17	1	52	1.81	0.01	1	35.8	1.62	0.03	2	22.5	1.12	7.95	12	1.98	1.06	3.8	15	0.714	1.06	13.7
5_7_5	60	16	1	42	1.73	0	1	20.2	1.27	0.03	1	19.4	1.03	3.92	1	15.2	1.13	1.92	1	15.2	1.13	7.68
5_7_6	60	24	1	39.4	1.74	0	2	20	1.57	0.04	3	15	1.13	9.39	20	0.70	1.22	2.75	21	0.554	1.17	11
5_7_7	60	20	1	42.9	1.68	0	2	19.2	1.47	0.03	3	15.6	1.07	6.06	19	0.47	1.11	2.42	19	0.25	1.05	14.2
5_7_8	60	22	1	44.1	1.81	0.01	1	30.6	1.62	0.03	1	22.2	1.21	6.57	1	14.1	1.43	2.17	1	12.4	1.24	16.3
5_7_9	60	17	1	39.4	1.81	0.01	1	15.3	1.62	0.03	1	14.4	1.17	6.42	12	2.7	1.25	3.2	12	2.42	1.25	13.4
1_4_10	120	7	1	74.4	5.33	0.0	1	25.5	1.5	0.0929	1	31.3	2.08	1.66	1	25.5	1.5	1.81	1	25.5	1.5	5.35
1_4_2	120	4	1	67.6	7	0	1	43.5	2.33	0.085	1	37.1	2.83	0.829	1	37.5	2	1.57	2	25	2	10.5
1_4_3	120	7	1	75.4	5.67	0	1	22.8	1.17	0.0971	1	30.2	2	1.67	1	22.8	1.17	1.81	1	22.8	1.17	5.36
1_4_6	120	2	1	45.8	24	0	1	16.7	3	0.055	1	15.6	7.25	0.265	2	0	2	1	2	0	2	4
1_4_7	120	6	1	65.2	3.2	0	1	26.2	1.4	0.0933	1	28.9	1.45	1.37	1	24.2	1.2	1.75	1	24.2	1.2	7.39
1_4_8	120	9	1	67.9	2.88	0.0	3	13.1	1.25	0.1	1	20.9	1.31	2.8	7	2.62	1.12	2.41	7	2.62	1.12	9.33
1_4_9	120	3	1	61.6	10.5	0	2	16.7	2	0.0767	1	22.4	3.5	0.542	2	11.1	1.5	1.38	3	0	1	7.37
1_5_10	120	18	1	72.3	2.94	0.0	1	23.5	2.24	0.114	1	28.3	1.76	6.28	1	17.3	1.88	2.44	1	17.3	1.88	8.89
1_5_1	120	16	1	57.6	2.4	0.0	13	3.82	1.47	0.112	4	16	1.33	4.22	14	2.77	1.47	2.01	14	2.2	1.4	12.5
1_5_2	120	18	1	65.8	2.12	0	1	21.6	1.35	0.116	1	23.1	1.21	9.9	13	4.86	1.35	4.89	13	4.16	1.35	16.5
1_5_3	120	18	1	68.3	2.94	0.0	3	24.1	2.06	0.113	2	26.9	1.54	8.59	3	15.2	1.18	3.5	3	15.2	1.18	9.93
1_5_4	120	8	1	71.8	3.71	0.0	1	53.8	3.29	0.0988	1	40.8	2.11	3.63	1	37.5	1.43	3.18	1	33.2	1.29	12.1
1_5_5	120	7	1	75.4	5.83	0	3	23.1	3	0.0971	1	27.9	2.58	1.9	4	13	1.5	2	4	13	1.5	8.23
1_5_6	120	16	1	61.5	2.4	0.0	6	14.5	1.53	0.112	3	19.3	1.25	7.46	13	1.25	1.07	3.33	13	1.25	1.07	9.61
1_5_7	120	16	1	63.1	2.53	0.0	10	6.69	1.47	0.114	2	18.4	1.32	5.06	13	3.59	1.27	2.3	13	3.59	1.27	8.85
1_5_8	120	21	1	56.7	2	0.0	1	21.8	1.45	0.121	1	22.8	1.12	10.1	1	12.6	1.05	3.49	1	12.6	1.05	9.8
1_5_9	120	14	1	69.9	3.38	0.0	1	33.6	2.38	0.111	1	27.4	1.75	6.45	9	5.95	1.23	3.21	9	5.95	1.23	10.8
1_7_10	120	37	1	52.2	2.17	0.0	1	25.8	1.78	0.133	1	23.7	1.35	23.8	1	16.6	1.47	4.48	1	14.2	1.47	16.9
1_7_1	120	38	1	50.9	1.84	0.0	1	17.6	1.7	0.139	1	20.8	1.27	18.2	1	14.5	1.54	3.08	1	12.8	1.54	19.1
1_7_2	120	42	1	39.5	1.61	0.0	4	10.2	1.49	0.141	2	14.2	1.12	15.3	4	6.89	1.37	2.56	4	6.38	1.29	13.7
1_7_3	120	41	1	47.3	2.1	0.0	10	13.1	1.7	0.138	4	17.5	1.36	15.6	10	9.45	1.62	2.79	10	9.05	1.6	22.3
1_7_4	120	32	1	48.9	1.68	0.0	1	25.8	1.48	0.133	1	24	1.11	15	1	21.5	1.29	3.15	1	15.4	1.29	36.5
1_7_5	120	35	1	57.3	1.88	0.0	5	18.9	1.68	0.135	2	22.2	1.31	20	5	12.6	1.68	3.8	5	12	1.62	23.3
1_7_6	120	41	1	49.8	1.77	0.0	5	11.9	1.65	0.14	2	16.9	1.19	25.5	7	6.05	1.35	3.93	7	5.25	1.35	15.4
1_7_7	120	37	1	52.9	1.89	0.0	10	16.8	1.61	0.136	4	20.3	1.28	23	10	11.5	1.61	4.07	10	9.78	1.42	14.8
1_7_8	120	38	1	47	1.81	0.0	1	28.5	1.51	0.14	1	24	1.21	18.8	1	20.5	1.51	3.3	1	20.4	1.51	18.5
1_7_9	120	42	1	43.2	1.59	0.0	20	5.13	1.41	0.143	8	12.7	1.04	25	20	2.29	1.17	3.7	20	2.09	1.17	20.3
Ttl.		920	57				168				110				376				418			
Avg.				52,165	3,048	0,000		22,490	1,688	0,076		21,300	1,518	7,635		10,514	1,341	2,684		8,593	1,278	12,160

TABLE 4.2 – *Comparaison des deux modèles mathématiques (4.28)-(4.32) et (4.33)-(4.36) pour générer les colonnes.*

			$\mathcal{GCM}_1$		$\mathcal{GCM}_2$	
instance	n	$\lvert\mathcal{Z}^*\rvert$	*nb col*	*sec.*	*nb col*	*sec.*
5_4_10	60	2	29	0.27	13	0.06
5_4_2	60	7	114	1.89	91	0.62
5_4_6	60	9	187	2.8	155	0.83
5_4_8	60	4	157	2.48	119	1.17
5_6_10	60	16	381	9.67	343	2.48
5_6_1	60	10	293	6.3	290	2.36
5_6_2	60	16	369	14.41	462	7.21
5_6_3	60	9	722	40.24	730	20.84
5_6_4	60	11	429	11.06	422	4.46
5_6_5	60	11	241	4.54	209	1.51
5_6_6	60	20	947	66.24	904	23.37
5_6_7	60	11	296	7.97	305	2.98
5_6_8	60	17	506	15.81	471	5.5
5_6_9	60	11	983	73.13	896	40.96
5_7_10	60	21	792	107.71	723	18.23
5_7_1	60	18	356	8.41	308	2.14
5_7_2	60	20	912	71.29	786	22.23
5_7_3	60	18	449	56.13	507	14.36
5_7_4	60	17	476	18.16	468	8.08
5_7_5	60	16	706	34.24	747	12.79
5_7_6	60	24	818	153.47	932	41.32
5_7_7	60	20	631	36.46	596	8.14
5_7_8	60	22	1037	153.19	1056	28.64
5_7_9	60	17	1081	131.51	1045	40.34
1_4_10	120	7	594	42.75	458	11.92
1_4_2	120	4	519	84.73	530	32.7
1_4_3	120	7	587	52.18	468	15.56
1_4_6	120	2	62	2.56	62	1.02
1_4_7	120	6	420	36.11	401	15.76
1_4_8	120	9	647	69.75	528	18.87
1_4_9	120	3	197	12.01	132	2.22
1_5_10	120	18	1600	220.07	1405	48.59
1_5_1	120	16	858	73.1	757	16.42
1_5_2	120	18	1396	166.73	1306	57.3
1_5_3	120	18	2137	500.64	2102	217.46
1_5_4	120	8	1334	305.25	1149	73.47
1_5_5	120	7	735	101.46	707	25.36
1_5_6	120	16	1377	230.52	1220	54.65
1_5_7	120	16	1218	158.73	1044	39.07
1_5_8	120	21	1339	172.34	1066	50.18
1_5_9	120	14	1361	365.49	1207	63.06
1_7_10	120	37	5270	6657.95	5566	2483.26
1_7_1	120	38	4017	3227.91	3979	845.57
1_7_2	120	42	3633	4267.5	3510	698.05
1_7_3	120	41	4414	5214.51	4211	2111.49
1_7_4	120	32	3641	3154.32	3652	822.35
1_7_5	120	35	3836	3459.07	3776	635.06
1_7_6	120	41	3651	3898.62	3671	713.28
1_7_7	120	37	3627	5000.8	3949	1399.57
1_7_8	120	38	5174	5273.11	5071	1708.44
1_7_9	120	42	2828	2481.44	2942	494.26
Avg.			1360,47	907	1322,490	254,226

TABLE 4.3 – *Comparaison des résultats obtenus par les différentes améliorations de la génération de colonnes.*

			$\mathcal{GCM}_1$+$\mathcal{H}$			$\mathcal{GCM}_2$		$\mathcal{GCM}_2$+$\mathcal{H}$			$\mathcal{GCM}_2$+$\mathcal{C}$		$\mathcal{GCM}_2$+$\mathcal{C}$+$\mathcal{H}$		
instance	*n*	$\lvert\mathcal{Z}^*\rvert$	*nb col*	(%) *heur*	*sec.*	*nb col*	*sec.*	*nb col*	(%) *heur*	*sec.*	*nb col*	*sec.*	*nb col*	(%) *heur*	*sec.*
5_4_10.text		2	43	97.6744	0.09	13	0.06	43	97.6744	0.09	13	0.07	43	97.6744	0.08
5_4_2.text	60	7	117	94.0171	0.54	91	0.62	111	93.6937	0.31	97	0.65	111	93.6937	0.32
5_4_6.text	60	9	242	96.281	0.85	155	0.83	242	96.281	0.5	164	0.93	242	96.281	0.49
5_4_8.text	60	4	143	79.021	1.1	119	1.17	144	81.25	0.85	128	1.15	145	84.8276	0.68
5_6_10.text	60	16	514	96.8872	1.97	343	2.48	514	96.8872	0.89	357	2.73	514	96.8872	0.93
5_6_1.text	60	10	355	97.1831	0.95	290	2.36	355	97.1831	0.62	300	2.59	355	97.1831	0.6
5_6_2.text	60	16	491	92.4644	4.05	462	7.21	529	88.2798	3.25	580	6.87	569	91.5641	2.18
5_6_3.text	60	9	1075	84.5581	17.44	730	20.84	1074	81.0987	12.35	767	16.5	1061	85.3911	6.82
5_6_4.text	60	11	543	90.7919	3.77	422	4.46	537	90.3166	2.41	402	3.96	567	91.358	1.96
5_6_5.text	60	11	376	97.0745	1.1	209	1.51	383	96.8668	0.58	210	1.59	376	97.0745	0.61
5_6_6.text	60	20	1494	89.2236	37.25	904	23.37	1520	87.5658	15.63	853	16.69	1504	89.9601	10.6
5_6_7.text	60	11	384	97.3958	1.2	305	2.98	384	97.3958	0.66	294	2.72	384	97.3958	0.75
5_6_8.text	60	17	661	95.4614	3.64	471	5.5	673	95.5423	1.49	464	5.09	689	93.3237	3.08
5_6_9.text	60	11	1251	68.0256	60.38	896	40.96	1279	66.9273	37.64	917	24.35	1287	67.2883	19.29
5_7_10.text	60	21	1133	91.96	68.4	723	18.23	1129	91.5855	13.86	779	18.88	1116	92.20	11.6
5_7_1.text	60	18	537	96.27	2.31	308	2.14	537	96.2756	1.18	298	2.21	545	96.6972	1.26
5_7_2.text	60	20	1032	92.53	23.1	786	22.23	1013	91.9052	10.44	789	15.14	1021	91.67	6.68
5_7_3.text	60	18	627	90.11	30.61	507	14.36	626	90.2556	4.51	500	13.93	614	90.7166	4.19
5_7_4.text	60	17	727	92.15	19.22	468	8.08	730	92.0548	6.43	450	7.01	727	93.9477	3.21
5_7_5.text	60	16	963	92.93	8.33	747	12.79	987	93.5157	3.43	722	11.92	1004	93.0279	3.32
5_7_6.text	60	24	1171	91.97	147.87	932	41.32	699	92.2083	36.19	56742	21.27	814	92.85	20.33
5_7_7.text	60	20	818	95.84	16.2	596	8.14	816	96.201	2.47	610	7.84	820	95.9756	2.29
5_7_8.text	60	22	1366	87.33	91.67	1056	28.64	8039	85.5247	14.01	1098	25.84	1356	86.57	13.99
5_7_9.text	60	17	1604	83.04	88.36	1045	40.34	1557	83.7508	27.23	1006	26.78	1555	84.56	15.01
1_4_10.text	120	7	692	84.10	16.12	458	11.92	770	81.8182	11.69	557	15.4	745	84.42	9.16
1_4_2.text	120	4	504	67.46	45.29	530	32.7	503	63.6183	20.63	492	27.76	523	63.0975	17.59
1_4_3.text	120	7	645	85.11	17.25	468	15.56	777	87.2587	14.81	443	13.65	731	87.9617	10.6
1_4_6.text	120	2	56	94.64	0.5	62	1.02	56	94.6429	0.31	71	1.03	56	94.6429	0.32
1_4_7.text	120	6	559	77.28	25.33	401	15.76	578	82.0069	9.81	407	12.05	566	80.742	8.56
1_4_8.text	120	9	883	84.59	36.25	528	18.87	828	85.5072	14.05	608	24.31	850	84.2353	13.31
1_4_9.text	120	3	177	83.05	4.58	132	2.22	161	81.9876	1.86	128	2.14	142	80.9859	1.64
1_5_10.text	120	18	1961	89.75	63.46	1405	48.59	2042	89.9608	27.29	1405	51.76	2097	89.9857	27.4
1_5_1.text	120	16	1151	94.0052	17.79	757	16.42	1183	94.59	7.8	770	18.86	1187	94.0185	7.9
1_5_2.text	120	18	1907	83.9014	83.16	1306	57.3	1861	80.7093	41.53	1292	52.98	1791	84.5896	31.99
1_5_3.text	120	18	2822	84.231	205.46	2102	217.46	2896	78.4185	167.87	2227	179.35	2927	78.2371	110.3
1_5_4.text	120	8	1615	76.4087	166.6	1149	73.47	1652	72.4576	69.09	1305	76.51	1602	72.4719	51.12
1_5_5.text	120	7	910	79.4505	50.59	707	25.36	958	78.3925	16.33	707	26.95	899	82.3137	12.57
1_5_6.text	120	16	1862	84.4253	82.23	1220	54.65	1777	83.7366	31.76	1278	56.06	1905	80.4199	36.77
1_5_7.text	120	16	1655	89.7885	45.13	1044	39.07	1680	88.631	20.95	986	35.42	1630	89.8773	17.81
1_5_8.text	120	21	1845	88.2927	69.61	1066	50.18	1726	85.4577	38.46	1094	44.43	1860	84.3548	33.48
1_5_9.text	120	14	1547	86.2961	78.55	1207	63.06	1560	88.0128	23.89	1218	58.76	1525	87.2787	23.57
1_7_10.text	120	37	6720	75.3869	5407.73	5566	2483.26	6801	75.8418	1978.97	5345	1652.45	6700	76.2687	1505.15
1_7_1.text	120	38	5338	88.4414	1934.38	3979	845.57	5370	88.3799	535.01	3944	623.51	5431	88.7314	439.41
1_7_2.text	120	42	4957	88.1985	3060.21	3510	698.05	5120	85.8203	581.32	3542	609.86	5142	86.2505	524.41
1_7_3.text	120	41	6146	90.0423	3969.03	4211	2111.49	6177	89.1695	1762.77	4344	1975.14	6230	89.5024	1555.14
1_7_4.text	120	32	4899	85.9155	2863.09	3652	822.35	4909	85.6386	594.87	3424	591.98	5080	84.1535	485.86
1_7_5.text	120	35	5550	90.955	1690.16	3776	635.06	5573	89.9874	336.95	3880	612.86	5470	90.0731	299.79
1_7_6.text	120	41	5153	89.5013	2691.28	3671	713.28	5192	86.4214	523.53	3693	575.7	5132	86.9641	420.05
1_7_7.text	120	37	4857	88.4291	5121.97	3949	1399.57	4880	88.4631	942.15	3855	1118.93	4867	88.2063	861.07
1_7_8.text	120	38	6744	88.1376	2868.48	5071	1708.44	6755	86.5137	1153.49	5276	1096.95	6811	86.4778	603.33
1_7_9.text	120	42	4414	93.6565	1874.64	2942	494.26	4475	92.5363	387.29	2958	413	4516	92.7591	318.15
Avg.			1828,156	88,268	649,397	1322,490	254,226	1964.33	87.57	186.5	2427,235	200.08	1839.88	87.98	148.17

Chapitre 5

Un problème de bin packing avec objets fragiles

Le travail de ce chapitre a été réalisé en collaboration avec Manuel Iori et Mauro Dell'Amico de l'équipe de recherche opérationnelle de l'université de Reggio Emilia en Italie.

5.1 Introduction

Ce chapitre est consacré à des modèles mathématiques et des bornes inférieures et supérieures pour résoudre le problème de bin packing avec objets fragiles ($\mathcal{BPP}$-$\mathcal{FO}$, voir section 1.6). Ce problème généralise d'une certaine manière le problème de $\mathcal{BPP}$-$\mathcal{C}$.

Rappelons que dans ce problème, chaque article possède une taille w_i et une fragilité f_i. Dans la suite nous supposons, sans perte de généralité, que $w_i \leq f_i$ $(\forall i \in I)$ et sauf indication contraire que les objets sont triés par ordre croissant des fragilités et par ordre décroissant des tailles en cas d'égalité.

Le $\mathcal{BPP}$-$\mathcal{FO}$ apparaît dans le domaine de la télécommunication et en particulier au niveau de l'allocation des utilisateurs à des canaux de fréquences dans un réseau cellulaire (voir Bansal *et al.* [4] et Chan *et al.* [15]). Dans les systèmes d'accès multiple par répartition en code (*CDMA* pour ***C**ode **D**ivision **M**ultiple **A**ccess*), on dispose d'un nombre limité de canaux de fréquence. Chaque canal a une capacité beaucoup plus grande que la bande passante d'un utilisateur unique, il est ainsi possible d'affecter plusieurs utilisateurs au même canal. Toutefois, une telle affectation peut produire des interférences entre les différents utilisateurs partageant le même canal, ce qui peut endommager la qualité de la communication (se référer à la section 1.6.2 pour plus de détails).

A notre connaissance, nos travaux sont les premiers à traiter ce problème d'un point de vue pratique en fournissant des résultats numériques. Nous avons proposé plusieurs modèles différents, des heuristiques, une métaheurstique, ainsi que des bornes inférieures pour ce problème.

Nos bornes inférieures polynomiales sont essentiellement basées sur le concept de "fonctions dual-réalisables (DFF)" (voir section 1.4.3.1). Pour améliorer ces bornes, une méthode de génération de colonnes est aussi proposée. Cette méthode est basée sur différentes formulations mathématiques que nous proposons pour le $\mathcal{BPP}$-$\mathcal{FO}$.

Nous avons testé plusieurs algorithmes pour résoudre le sous-problème de pricing (deux modèles de programmation mathématique et un algorithme de programmation dynamique). Nous avons aussi stabilisé la génération de colonnes par le biais d'un ensemble de *coupes duales* [113].

Nous proposons une recherche locale à voisinage variable (VNS) qui permet d'obtenir des solutions faisables de très bonne qualité. Cette recherche est initialisée par une solution fournie par un ensemble de méthodes heuristiques gloutonnes que nous proposons pour résoudre rapidement le $\mathcal{BPP}$-$\mathcal{FO}$.

Afin de pouvoir tester l'efficacité de tous ces algorithmes, un jeu d'essai est nécessaire. Aucun n'a été proposé jusqu'à présent, nous proposons donc dans ce chapitre un jeu d'essai, que nous analysons en détail.

Le chapitre est organisé comme suit. Dans la section 5.2 nous proposons des modèles mathématiques pour le $\mathcal{BPP}$-$\mathcal{FO}$. La section 5.3 est dédiée à des techniques de prétraitement visant à modifier une instance donnée de $\mathcal{BPP}$-$\mathcal{FO}$. Dans la section 5.4 nous proposons un ensemble de bornes inférieures polynomiales. La section 5.5 est consacrée à un ensemble de méthodes heuristiques à utiliser pour avoir des solutions rapides pour une instance de $\mathcal{BPP}$-$\mathcal{FO}$, ainsi qu'un algorithme de recherche locale à base de voisinage variable. Dans la section 5.6 nous présentons un algorithme de génération de colonnes pour résoudre le modèle à base de couverture d'ensemble proposé dans la section 5.2. Pour accélérer la convergence de cet algorithme, nous proposons plusieurs techniques pour résoudre le sous-problème. Pour tester l'efficacité de ces algorithmes, nous les testons sur un jeu d'essai que nous proposons dans la section 5.7. Nous finissons avec quelques conclusions et perspectives dans la section 5.8.

5.2 Modèles mathématiques

Nous présentons dans cette section deux modèles compacts nécessitant un nombre polynomial de variables et de contraintes, et un troisième modèle nécessitant un nombre exponentiel de variables.

5.2.1 Un modèle compact simple

Soit $f_{\max} = \max_{j=1,\dots,n}\{f_j\}$ la fragilité maximale parmi celles des objets de l'instance considérée. La variable binaire y_i prend la valeur 1 si le bin i ($i = 1,\dots,n$) est utilisé et 0 sinon. La variable binaire x_{ji} est égale à 1 si l'objet j ($i, j = 1,\dots,n$) est placé dans le bin i et 0 sinon. Le $\mathcal{BPP}$-$\mathcal{FO}$ peut être modélisé avec le programme linéaire suivant :

$$
\begin{aligned}
\min \quad & \sum_{i=1}^{n} y_i && && (5.1)\\
\text{t.q.} \quad & \sum_{i=1}^{n} x_{ji} = 1 && j = 1,\dots,n && (5.2)\\
& \sum_{j=1}^{n} w_j x_{ji} \leq f_{\max} + x_{ki}(f_k - f_{\max}) && i,k = 1,\dots,n && (5.3)\\
& x_{ji} \leq y_i && i,j = 1,\dots,n && (5.4)\\
& y_j \in \{0,1\} && j = 1,\dots,n && (5.5)\\
& x_{ji} \in \{0,1\} && i,j = 1,\dots,n. && (5.6)
\end{aligned}
$$

Les contraintes (5.2) imposent que chaque objet soit placé dans un et un seul bin. Les contraintes (5.3) vérifient que la somme des tailles des objets placés dans un bin ne dépasse la fragilité d'aucun de ces objets (si l'objet k est placé dans le bin i, le côté droit de la contrainte est égal à f_k, sinon la contrainte est redondante). Les contraintes (5.4) sont utilisées pour renforcer la relaxation linéaire du modèle.

Le modèle (5.1)–(5.6) dérive du modèle classique du $\mathcal{BPP}$. Son principale inconvénient réside dans les contraintes (5.3) dont la complexité est de l'ordre de $\mathcal{O}(n^2)$ qui modélisent la contrainte non-linéaire (1.35) en utilisant une grande valeur ($f_{\max}$). Cette valeur peut dégrader la qualité de la relaxation linéaire du modèle et le rendre très dépendant de la fragilité f_{max}.

5.2.2 Une meilleure formulation

Rappelons que les objets sont triés par ordre croissant de fragilité, et par ordre décroissant de taille en cas d'égalité. Un objet est dit "objet témoin" s'il a la fragilité la plus petite du bin où il est placé. On définit une variable booléenne y_i qui prend la valeur 1 si l'objet i est un objet témoin et 0 sinon ($i = 1,\dots,n$). On définit aussi la variable booléenne x_{ji} qui prend la valeur 1 si l'objet j est placé dans le bin contenant un objet témoin i et 0 sinon ($i = 1,\dots,n, j = i+1,\dots,n$). Le $\mathcal{BPP}$-$\mathcal{FO}$ peut être modélisé sous la forme du modèle linéaire en nombre entiers suivant :

$$
\begin{aligned}
\min \quad & \sum_{i=1}^{n} y_i && && (5.7)\\
\text{t.q.} \quad & y_i + \sum_{j=1}^{i-1} x_{ij} = 1 && i = 1,\dots,n && (5.8)\\
& \sum_{j=i+1}^{n} w_j x_{ji} \leq (f_i - w_i) y_i && i = 1,\dots,n && (5.9)\\
& x_{ji} \leq y_i && i = 1,\dots,n, j = i+1,\dots,n && (5.10)\\
& y_i \in \{0,1\} && i = 1,\dots,n && (5.11)\\
& x_{ji} \in \{0,1\} && i = 1,\dots,n, j = i+1,\dots,n. && (5.12)
\end{aligned}
$$

Les contraintes (5.8) contrôlent le fait qu'un objet est témoin ou non. Les contraintes (5.9) vérifient que la somme des tailles des objets placés dans un bin ne dépasse pas la fragilité du témoin.

Le modèle (5.7)–(5.12) partage quelques similarités avec le modèle proposé par Ceselli et Righini [13] pour une variante du problème de bin packing dite *Ordered Open-End Bin Packing Problem* (OOEBPP) (voir section 1.6.2).

Malgré les similarités dans les modèles, le $\mathcal{BPP}$-$\mathcal{FO}$ et le OOEBPP sont deux problèmes très différents. L'idée de OOEBPP est d'avoir un grand objet en dernier dans chaque bin afin de garder de l'espace pour d'autres objets. Au contraire, dans le $\mathcal{BPP}$-$\mathcal{FO}$, la contrainte de sac-à-dos doit être satisfaite, et il est préférable d'avoir des objets similaires, surtout en terme de fragilité, placés dans le même bin. Toutefois, il existe un cas dans lequel les deux problèmes sont équivalents : le cas spécial de $\mathcal{BPP}$-$\mathcal{FO}$ où $f_j = w_j + C$ pour tout $j = 1, \ldots, n$ est équivalent au cas spécial de OOEBPP où l'ordre d'arrivée des objets correspond à un tri par ordre décroissant des tailles.

5.2.3 Une formulation à base de couverture d'ensemble

Le modèle (1.10)-(1.12) basé sur la méthode de décomposition de Gilmore et Gomory [54, 55] peut être aussi utilisé pour résoudre le $\mathcal{BPP}$-$\mathcal{FO}$. Ici on considère P l'ensemble de patterns valides, *i.e.* l'ensemble des patterns p pour lesquels

$$\sum_{j=1}^{n} w_j a_{jp} \leq \min_{j=1,\ldots,n} \{f_j a_{jp}\}. \tag{5.13}$$

Le nombre de patterns possibles pouvant être très grand, résoudre la relaxation linéaire du modèle (1.10)–(1.12) peut se révéler difficile. Nous utilisons une méthode de génération de colonnes, décrite dans la section 5.6.

Le sous-problème à résoudre est le **problème de sac-à-dos binaire avec objets fragiles (KP01-FO)**, dans lequel on dispose de n objets j de profit p_j, de taille w_j et de fragilité f_j ($j = 1, \ldots, n$). On dispose aussi d'un sac de taille non définie. Le but est de trouver le sous-ensemble d'objets maximisant le profit total et dont la taille totale ne dépasse la fragilité d'aucun des objets placés dans le sac (1.35).

5.3 Prétraitements

Plusieurs prétraitements ont été proposés dans la littérature pour le $\mathcal{BPP}$ et le $\mathcal{BPP}$-$\mathcal{C}$. Malheureusement, ces prétraitements ne sont applicables directement sur une instance de $\mathcal{BPP}$-$\mathcal{FO}$. Considérons par exemple le prétraitement de Boschetti et Mingozzi [7] décrit dans la section 1.4.2 et prenons un contre exemple avec deux objets : objet 1 de taille $w_1 = 2$ et fragilité $f_1 = 4$, et objet 2 de taille $w_2 = 2$ et fragilité $f_2 = 5$. Ces objets peuvent être placés dans un même bin. En appliquant le prétraitement de [7] avec $j = 2$, on obtient une capacité résiduelle $w^* = f_2 - w_1 - w_2 = 1$ mais si on modifie la taille de l'objet 2 ($w_2 = w_2 + w^* = 3$), la solution n'est alors plus réalisable. Par conséquent, nous proposons le résultat suivant :

Proposition 5.3.1 *Si w^* est la capacité résiduelle minimale dans tout bin contenant l'objet j, la transformation suivante peut être alors appliquée à une instance de $\mathcal{BPP}$-$\mathcal{FO}$ sans violer*

la contrainte d'optimalité :

$$f_j \mapsto f_j - w^*. \tag{5.14}$$

Démonstration. Considérons un ensemble d'objets incluant j, et qui peut être placé dans un seul bin. Supposons qu'en modifiant la fragilité de j par (5.14) il est impossible de trouver un placement faisable de cet ensemble d'objets dans un seul bin. L'infaisabilité viendrait du fait que $\mathcal{W} > f_j - w^*$ où $\mathcal{W}$ est la somme totale des objets de cet ensemble. La capacité résiduelle après avoir placé cet ensemble dans un bin est égale à $(f_j - \mathcal{W}) \leq f_j - f_j + w^* = w^*$, ce qui contredit le fait que w^* soit l'espace minimal restant dans tout bin contenant l'objet j. □

En se basant sur la considération précédente, on propose une procédure de prétraitement composée de conditions imbriquées. A chaque objet j correspond un *ensemble compatible* de j noté $\gamma(j) = \{i \in I \setminus \{j\} : w_i + w_j \leq \min\{f_i, f_j\}\}$, $j \in I$. La première condition est :

a) si $\gamma(j) = \emptyset$ (aucun objet ne peut être placé avec j dans un même bin), alors on place l'objet j dans un bin seul et on l'enlève de l'instance.
Si la condition *a* n'est pas vérifiée, on résout un problème de sac-à-dos particulier dit KP01-FO dans lequel : 1) les objets sont $\{j\} \cup \gamma(j)$, 2) les profits des objets sont égaux à leur taille et 3) l'objet j est forcé à être dans la solution. Soit $z(j)$ la valeur de la solution optimale et $\overline{\gamma}(j)$ le sous-ensemble optimal d'objets placés dans le sac. Les deux conditions imbriquées sont utilisées pour prétraiter l'instance :

b) si $|\overline{\gamma}(j)| = |\gamma(j)|$ (j est ses objets compatibles peuvent être placés dans un même bin), alors on place $\{j\} \cup \gamma(j)$ dans un même bin et on les enlève de l'instance ; sinon

c) si $|\overline{\gamma}(j)| = 1$, on teste si un seul objet peut être placé avec j dans tous les bins faisables possibles. Pour ce faire, on résout un nouveau KP01-FO, équivalent à celui que l'on vient de résoudre mais avec des profits égaux à 1. Si la solution trouvée a la valeur 2 (i.e., un seul objet peut être placé avec j dans le sac), on teste alors s'il existe un objet dans $\gamma(j)$ plus difficile à placer que les autres objets. En particulier, on teste s'il existe un objet $\ell \in \gamma(j)$ tel que $w_\ell \geq w_k, \forall k \in \gamma(j)$ et $f_\ell \leq f_k, \forall k \in \gamma(j)$. Si un tel objet existe, on place j et ℓ dans un même bin et on les enlève de l'instance ; sinon

d) si $w_j + z(j) < f_j$ (la fragilité de j n'est pas complètement atteinte), on modifie f_j comme suit $f_j = w_j + z(j)$.

A chaque fois que l'une des quatre conditions est vérifiée, la procédure de prétraitement met à jour l'instance et recommence le même traitement avec l'objet j suivant. Il faut noter qu'une mise à jour d'un objet donné peut changer les ensembles compatibles d'autres objets. Cela peut entraîner des améliorations qui n'étaient pas possibles avant la mise à jour. La procédure est donc réitérée tant que des améliorations sont possibles.

L'algorithme (12) donne une explication étape par étape de la procédure de prétraitement.

Algorithme 12 : Prétraitements pour le $\mathcal{BPP}$-$\mathcal{FO}$

entrée : Un ensemble d'objets I
sortie : Une valeur égale à la quantité d'objets éliminés par les prétraitements

```
1  stop ← false;
2  update ← true;
3  tant que stop = false faire
4      stop ← true;
5      pivot ← 1;
6      tant que pivot ≤ |I| faire
           // I_pivot = {i ∈ I \ pivot | w_i + w_pivot ≤ min{f_i, f_pivot}}
7          si |I_pivot| = 0 alors
8              valeur ← valeur + 1;
9              I ← I \ {i};
10             update ← true;
11         sinon
12             si Σ_{i∈I_pivot} w_i + w_pivot ≤ min{min_{i∈I_pivot}{f_i}, f_pivot} alors
13                 valeur ← valeur + 1;
14                 I ← I \ {i ∪ I_pivot};
15                 update ← true;
16             sinon
17                 s ← KP(I_pivot, w_pivot, f_pivot, "p = w")
18                 pivot_placé ← false;
19                 si |s| = 2 alors
20                     s' ← KP(I_pivot, w_pivot, f_pivot, "p = 1")
21                     si s' = 2 alors
                           // voisin_dominant = i ∈ I_pivot
22                         si ∃ i ∈ I_pivot | ∀j ∈ I_pivot \ i, w_i ≥ w_j, f_i ≤ f_j alors
23                             valeur ← valeur + 1;
24                             I ← I \ {i ∪ voisin_dominant};
25                             update ← true;
26                             pivot_placé ← true;
27                 si pivot_placé = false ET s ≤ f_pivot alors
28                     f_pivot ← s;
29                     update ← true;
30         si update = false alors
31             i ← i + 1;
32         sinon
33             stop ← false;
```

5.4 Bornes inférieures polynomiales

Dans cette section, nous proposons un ensemble de bornes inférieures rapides, et qui reposent sur la structure combinatoire du $\mathcal{BPP}$-$\mathcal{FO}$. Dans la suite, la lettre L est utilisée pour représenter une méthode de calcul de borne inférieure ainsi que la valeur fournie par cette méthode.

Pour évaluer la performance d'une méthode de calcul de borne inférieure, on définit la notion de *rapport de performance dans le pire des cas* comme étant une valeur réelle r telle que $r \leq L(P)/z(P)$, où P est une instance d'un problème d'optimisation combinatoire, $z(P)$ la valeur de la solution optimale pour l'instance P, et $L(P)$ la valeur d'une borne inférieure pour l'instance P. On dit qu'un rapport de performance dans le pire des cas est *serré* si la valeur r est atteinte au moins par une instance.

5.4.1 Bornes inférieures fractionnaires

Le rapport de performance dans le pire de cas de la borne inférieure L_0 décrite dans le chapitre d'état de l'art est arbitrairement mauvais. Il suffit de considérer une instance de $\mathcal{BPP}$-$\mathcal{FO}$ avec n objets, les premiers $n-1$ objets de taille 1 et fragilité 1 et le dernier de taille 1 et fragilité n. La solution optimale a une valeur $z(P) = n$ car chaque objet doit être placé dans un bin seul, tandis que $L_0 = 1$. Quand n tend vers l'infini, le rapport de performance dans le pire des cas de L_0 tend vers 0.

Le rapport de performance dans le pire des cas de la borne inférieure L_1 décrite dans le chapitre d'état de l'art est aussi arbitrairement mauvais. Considérons l'instance qui contient les n objets suivants :

- objet 1 de taille $1-(n-1)\times\varepsilon$ et de fragilité n,
- tout autre objet j est de taille $(1+\varepsilon)\times n^j$ et de fragilité n^{j+1}, $j = 2,\ldots,n$.

Chacun des objets de cette instance doit être placé dans un bin seul, tandis que

$$L_1 = \frac{1-(n-1)\times\varepsilon}{n} + \sum_{i=2}^{n} \frac{(1+\varepsilon)\times n^j}{n^{j+1}} = \frac{1-(n-1)\times\varepsilon}{n} + (n-1)\frac{(1+\varepsilon)}{n} = 1. \quad (5.15)$$

Le rapport de performance dans le pire des cas de L_1 tend vers 0 quand n tend vers l'infini.

La borne L_2 décrite dans la section 1.6.4 a meilleur rapport de performance, puisqu'il est égal à $1/2$.

Proposition 5.4.1 *Le rapport de performance dans le pire des cas de la borne inférieure L_2 décrite dans le chapitre d'état de l'art est $1/2$ et cette valeur peut être atteinte.*

Démonstration. La preuve est implicite dans Bansal *et al.* [4]. Pour des raisons de clarté, nous présentons cette preuve d'une façon plus concise. On construit une solution heuristique nécessitant $2L_2$ bins par l'algorithme 4. Le premier bin contient au plus un objet fractionnaire. Si c'est le cas, on enlève les deux portions de cet objet fractionnaire des deux premiers bins, et on le place dans un bin seul. Le même traitement est réitéré avec le deuxième bin, qui contient maintenant au plus un seul objet fractionnaire, et

ainsi de suite. Pour chaque bin dans l'ensemble original de L_2 bins, on ouvre au plus un seul nouveau bin, d'où la solution heuristique nécessite $2L_2$ bins. Le rapport de performance dans les pire des cas est de $L_2/2L_2 = 1/2$.

Pour vérifier que la valeur est serrée, il suffit de considérer une instance de n objets, chacun de taille égale à $C/2 + \varepsilon$, avec ε une petite valeur positive, et de fragilité égale à C. La solution optimale nécessite n bins, tandis que $L_2 = n/2 - 1$. □

5.4.2 Bornes inférieures basées sur la compatibilité entre objets

On peut associer un graphe de conflits $G = (I, E)$ à toute instance de $\mathcal{BPP}$-$\mathcal{FO}$, une arête $(i, j) \in E$ est créée pour tout couple d'objets $\{i, j\} \in I$ vérifiant la condition suivante : $w_i + w_j > \min\{f_i, f_j\}$.

Toute borne inférieure pour le $\mathcal{BPP}$-$\mathcal{C}$ peut ainsi être utilisée pour calculer des bornes inférieures pour le $\mathcal{BPP}$-$\mathcal{FO}$. Une instance de $\mathcal{BPP}$-$\mathcal{FO}$ est transformée en une instance de $\mathcal{BPP}$-$\mathcal{C}$ en construisant un graphe de conflit G comme indiqué au début de cette section, et à mettre à jour la fragilité de chaque objet $i \in I$ comme suit $f_i = f_{\max}$.

Dans les résultats numériques rapportés à la fin de ce chapitre, nous utilisons la version améliorée de L_{cp} que nous avons proposé dans le chapitre 2.

5.4.3 Fonctions dual-réalisables

Dans la suite, on note BPFO-DFF toute DFF dédiée à une instance de $\mathcal{BPP}$-$\mathcal{FO}$ et CS-DFF toute DFF dédiée au cutting-stock.

5.4.3.1 Définitions et propriétés

On définit maintenant d'une façon formelle le concept de *fonctions dual-réalisables pour le BPFO* (BPFO-DFF).

Definition 5.4.1 *Un mapping g définie de I dans $[0, 1]$ est une BPFO-DFF ssi*

$$\forall S \subseteq I, \sum_{i \in S} w_i \leq \min_{j \in S}\{f_j\} \implies \sum_{i \in S} g(i) \leq 1 \tag{5.16}$$

Une BPFO-DFF peut être utilisée directement pour calculer des bornes inférieures pour le BPFI.

Proposition 5.4.2 *Si g est une BPFO-DFF et I un ensemble d'objets à placer dans une instance D de BPFI, $L_g = \lceil \sum_{i \in I} g(i) \rceil$ est une borne inférieure valide sur le nombre minimum de bins nécessaires pour placer les objets de D.*

Démonstration. Considérons une solution duale π du modèle (1.10)–(1.12) telle que $\pi_i = g(i)$ pour $i \in I$. Selon la définition de BPFO-DFF, pour chaque pattern valide P de BPFI, $\sum_{i \in P} g(i) \leq 1$. Par conséquent, π est une solution duale valide de (1.10)–(1.12). La valeur de $\sum_{i \in I} g(i)$ est donc une borne inférieure valide pour l'instance de BPFI considérée. □

La proposition suivante découle des équations (1.16) et (5.16).

Proposition 5.4.3 *Si λ une CS-DFF et g une BPFO-DFF, $\lambda \circ g$ est une BPFO-DFF.*

Une CS-DFF peut donc être appliquée comme un post-traitement après l'application d'une BPFO-DFF sur l'instance originale et peut améliorer les résultats.

Pour qu'une DFF classique soit utile, la propriété de *superadditivité* est souhaitable. Une fonction g est superadditive si $g(x) + g(y) \leq g(x + y)$ pour toute valeur x, y. Nous prouvons maintenant le lemme suivant qui nous sera utile dans la suite.

Lemme 5.1 *Si g est superadditive, pour chaque liste de k valeurs $x_1, \ldots, x_k$, $g(x_1) + g(x_2) + \ldots + g(x_k) \leq g(x_1 + x_2 + \ldots + x_k)$.*

Démonstration. Nous prouvons ce résultat par récurrence sur le nombre d'éléments. S'il existe un seul élément x_1, le résultat est évidemment vrai. Supposons que le résultat est vrai pour k éléments et considérons une liste de $k+1$ éléments. Par hypothèse, $g(x_1) + \ldots + g(x_k) \leq g(x_1 + \ldots + x_k)$, et puisque g est superadditive, $g(x_1) + \ldots + g(x_k) + g(x_{k+1}) \leq g(x_1 + \ldots + x_k) + g(x_{k+1}) \leq g(x_1 + \ldots + x_k + x_{k+1})$. □

Les résultats suivants permettent de montrer le lien entre les fonctions superadditives et les BPFO-DFF. Dans la suite, nous supposons que les fragilités des objets sont strictement plus grandes que 0.

Proposition 5.4.4 *Soit λ une fonction superadditive croissante telle que $\lambda(0) = 0$. La fonction suivante g est une BPFO-DFF.*

$$g : i \mapsto \lambda(w_i)/\lambda(f_i) \tag{5.17}$$

Démonstration. Dans un premier temps, notons que $0 \leq \lambda(w_i)/\lambda(f_i) \leq 1$ car $\lambda(0) = 0$ et λ est croissante. Considérons un pattern valide quelconque P. Par définition, $\sum_{i \in P} w_i \leq \min_{j \in P}\{f_j\}$. Puisque λ est croissante, on a :

$$\lambda(\sum_{i \in P} w_i) \leq \lambda(\min_{j \in P}\{f_j\}) \tag{5.18}$$

et vu que λ est superadditive (voir Lemme 5.1) et croissante, on obtient :

$$\sum_{i \in P} \lambda(w_i) \leq \lambda(\sum_{i \in P} w_i) \leq \lambda(\min_{j \in P}\{f_j\}) = \min_{j \in P}\{\lambda(f_j)\} \tag{5.19}$$

Par conséquent on a

$$\sum_{i \in P} \frac{\lambda(w_i)}{\min_{j \in P}\{\lambda(f_j)\}} \leq 1 \tag{5.20}$$

En considérant le fait que, par définition, pour chaque i dans P, on a $f_i \geq \min_{j \in P}\{f_j\}$ et donc (puisque λ est croissante) $\lambda(f_i) \geq \lambda(\min_{j \in P}\{f_j\})$, on obtient

$$\sum_{i \in P} g(i) = \sum_{i \in P} \frac{\lambda(w_i)}{\lambda(f_i)} \leq \sum_{i \in P} \frac{\lambda(w_i)}{\min_{j \in P}\{\lambda(f_j)\}} \leq 1 \tag{5.21}$$

ce qui est le résultat souhaité. □

Précisons que toutes les CS-DFF maximales sont superadditives et croissantes (voir [20]). Par conséquent, toutes ces fonctions définies pour le cutting-stock peuvent être utilisées pour le bin packing avec objets fragiles. Si elles sont définies indépendemment de la taille du bin, elles peuvent être appliquées directement. Toutefois, certaines CS-DFF sont définies par rapport à la taille du bin et dans ce cas la valeur $f_{\max}$ est utilisé comme une taille fictive du bin.

Le résultat de la proposition 5.4.4 peut être améliorée en donnant à l'image des grands objets ($i \in I$, $w_i > f_i/2$) une valeur égale à la plus grande taille qu'il peut prendre lorsqu'on a transformé les petits objets par la fonction λ.

Proposition 5.4.5 *Soit λ une fonction croissante superadditive telle que $\lambda(0) = 0$. La fonction suivante $\bar{g}$ est une BPFO-DFF*

$$\bar{g} : i \mapsto \begin{cases} 1 - \max\limits_{\rho=0,\dots,f_i-w_i} \{\frac{\lambda(\rho)}{\lambda(w_i+\rho)}\} & si\ w_i > f_i/2 \\ \frac{\lambda(w_i)}{\lambda(f_i)} & si\ w_i \leq f_i/2. \end{cases} \tag{5.22}$$

Démonstration. Considérons un pattern valide quelconque P. S'il n'existe pas d'objet i tel que $w_i > f_i/2$, le résultat est équivalent à la proposition 5.4.4. Par conséquent, on s'intéresse seulement au cas où il existe un tel objet i.

Puisque $\sum_{j\in P\setminus\{i\}} w_j + w_i \leq f_l, \forall l \in P \setminus \{i\}$ et λ est croissante, on a $\lambda(f_l) \geq \lambda(\sum_{j\in P\setminus\{i\}} w_j + w_i)$, $\forall l \in P \setminus \{i\}$.

$$\sum_{j\in P\setminus\{i\}} \frac{\lambda(w_j)}{\lambda(f_j)} \leq \frac{\sum_{j\in P\setminus\{i\}} \lambda(w_j)}{\lambda(\sum_{l\in P\setminus\{i\}} w_l + w_i)} \tag{5.23}$$

Puisque λ est superadditive, et d'après le Lemme 5.1,

$$\sum_{j\in P\setminus\{i\}} \lambda(w_j) \leq \lambda(\sum_{j\in P\setminus\{i\}} w_j) \tag{5.24}$$

Par conséquent, on obtient

$$\sum_{j\in P\setminus\{i\}} \frac{\lambda(w_j)}{\lambda(f_j)} \leq \frac{\lambda(\sum_{j\in P\setminus\{i\}} w_j)}{\lambda(\sum_{l\in P\setminus\{i\}} w_l + w_i)} \tag{5.25}$$

Puisque P est un pattern valide, $0 \leq \sum_{j\in P\setminus\{i\}} w_j \leq f_i$. Par construction, la partie droite de l'équation (5.25) est plus petite que $\max\limits_{\rho=1,\dots,f_i-w_i} \{\frac{\lambda(\rho)}{\lambda(w_i+\rho)}\}$ (car $\sum_{j\in P\setminus\{i\}} w_j$ est une valeur possible de ρ). Par conséquent, on a

$$\sum_{j\in P\setminus\{i\}} \frac{\lambda(w_j)}{\lambda(f_j)} \leq \max_{\rho=1,\dots,f_i-w_i} \left\{ \frac{\lambda(\rho)}{\lambda(w_i+\rho)} \right\} \tag{5.26}$$

Ceci donne

$$\sum_{j\in P\setminus\{i\}} \bar{g}(j)+\bar{g}(i) = \sum_{j\in P\setminus\{i\}} \frac{\lambda(w_j)}{\lambda(f_j)} + 1 - \max_{\rho=1,\dots,f_i-w_i} \left\{ \frac{\lambda(\rho)}{\lambda(w_i+\rho)} \right\} \leq 1 \tag{5.27}$$

qui est le résultat souhaité.

□

5.4.3.2 Une instance de BPFO-DFF

Pour créer des BPFO-DFF à partir des CS-DFF classiques, on peut utiliser les propositions 5.4.3,5.4.4 et 5.4.5. Nous appliquons ces résultats théoriques sur une simple fonction superadditive : "$\lfloor\ \rfloor$". Nous montrons aussi comment utiliser deux différentes BPFO-DFF sur une instance de BPFI en utilisant une méthode de décomposition.

Corollaire 5.1 *Soit k un paramètre donné ($1 \leq k < \min_{j\in I}\{f_j\}$), la fonction suivante est une BPFO-DFF*

$$g_2^k : i \mapsto \frac{\lfloor w_i/k \rfloor}{\lfloor f_i/k \rfloor} \tag{5.28}$$

Démonstration. La fonction "$\lfloor\ \rfloor$" est croissante et superadditive, et $\lfloor 0 \rfloor = 0$. Par simple application de la proposition 5.4.4 avec $\lambda = \lfloor x/k \rfloor$ on obtient le résultat attendu. □

Précisons que l'utilisation de g_2^k aboutit à des bornes qui dominent L_0 et L_1. En fait, il suffit de choisir $k = 1$ pour avoir les résultats de L_1. Les bornes peuvent aussi être strictement meilleures que L_2. Supposons qu'on dispose de n objets de taille $C/2+\epsilon$ et fragilité C. Dans ce cas, $L_2 = n/2 - 1$, tandis que la borne obtenue par $g_2^{C/2+\epsilon}$ est égale à $\sum_{i\in I} 1/1 = n$ (résultat optimal).

La fonction g_2^k peut être améliorée en donnant à l'image des grands objets ($i \in I$, $w_i > f_i/2$) une valeur égale au plus grand espace résiduel après avoir transformé les autres objets par g_2^k.

Corollaire 5.2

$$\bar{g}_2^k : i \mapsto \begin{cases} 1 - \max\limits_{\rho=1,\dots,f_i-w_i} \left\{ \frac{\lfloor \rho/k \rfloor}{\lfloor (w_i+\rho)/k \rfloor} \right\} & si\ w_i > f_i/2 \\ \frac{\lfloor w_i/k \rfloor}{\lfloor f_i/k \rfloor} & si\ w_i \leq f_i/2 \end{cases} \tag{5.29}$$

Démonstration. la fonction "$\lfloor\ \rfloor$" est croissante et superadditive et $\lfloor 0 \rfloor = 0$. Par simple application de la proposition 5.4.5 avec $\lambda = \lfloor x/k \rfloor$ on obtient le résultat recherché. □

Nous montrons maintenant par un exemple les améliorations obtenues en utilisant une BPFO-DFF. Considérons une instance de 100 objets de taille 5 et de fragilité 8 et 100 objets de taille 2 et de fragilité 7.

- $L_1 = \lceil 100 \times 5/8 + 100 \times 2/7 \rceil = 91$

- $L_2 = 92$ (les 28 premiers bins sont utilisés pour placer les 98 premiers objets de taille 2, ensuite, un bin contient deux objets de taille 2 et 3 unités d'un objet de taille 5. Les objets ou portions d'objets restants de taille 5 sont placés dans 63 bins :
- $L_{g_2^k}$ avec $k = 2 : \lceil 100 \times \frac{\lfloor 5/2 \rfloor}{\lfloor 8/2 \rfloor} + 100 \times \frac{\lfloor 2/2 \rfloor}{\lfloor 7/2 \rfloor} \rceil = \lceil 100 \times 1/2 + 100 \times 1/3 \rceil = 84$
- $L_{\bar{g}_2^k}$ avec $k = 2 : \lceil 100 \times (1 - \frac{\lfloor 2/2 \rfloor}{\lfloor (5+2)/2 \rfloor}) + 100 \times \frac{\lfloor 2/2 \rfloor}{\lfloor 7/2 \rfloor} \rceil = \lceil 100 \times 2/3 + 100 \times 1/3 \rceil = 100$

5.4.3.3 Une BPFO-DFF basée sur la décomposition d'instance

La fonction décrite dans cette section est utilisée pour diviser l'instance en deux sous-instances, ainsi deux différentes BPFO-DFF peuvent être appliquées sur chacune de ces sous-instances.

Proposition 5.4.6 *Soit h_1 et h_2 deux BPFO-DFF. Pour un paramètre donné k, la fonction suivante g_3^k est une BPFO-DFF*

$$g_3^k : i \mapsto \begin{cases} h_1(i) & si\ w_i > k \\ 0 & si\ f_i > k\ et\ w_i \leq k \\ h_2(i) & si\ f_i \leq k. \end{cases} \tag{5.30}$$

Démonstration. Supposons que P est un pattern valide. Soit $P_1 = \{i \in P : w_i > k\}$, $P_2 = \{i \in P : f_i > k, w_i \leq k\}$ et $P_3 = \{i \in P : f_i \leq k\}$. Nous considérons deux cas différents selon qu'il existe des objets dans P_1 ou non :

- si $|P_1| > 0$, alors $|P_3 = 0|$. Par conséquent, $\sum_{i \in P} g_3^k(i) = \sum_{i \in P_1} h_1(i) + \sum_{i \in P_2} 0$. Par hypothèse, h_1 est une BPFO-DFF valide, donc g_3^k est valide dans ce cas.
- si $|P_1| = 0$, $\sum_{i \in P} g_3^k(i) = \sum_{i \in P_2} 0 + \sum_{i \in P_3} h_2(i)$. Par hypothèse, h_2 est valide, alors g_3^k est aussi valide dans ce cas.

□

5.5 Recherche à voisinage variable (VNS)

Afin d'obtenir des solutions heuristiques de bonne qualité rapidement, nous utilisons une méthode de *recherche locale à voisinage variable* (VNS). L'idée principale de cette méthode, proposée par Hansen et Mladenovic [92] en 1997, est de changer itérativement la meilleure solution en utilisant un voisinage initialement petit et qui croît au fil des itérations. Chaque nouvelle solution obtenue de cette façon est améliorée par une sous-routine de recherche locale et remplace la meilleure solution si elle est meilleure. Les algorithmes de type VNS ont été largement utilisés pour résoudre un grand nombre de problèmes d'optimisation combinatoire. Un état de l'art complet sur le VNS a été récemment établi par Hansen *et al.* [59]. Une application réussie du VNS dans le cas de bin packing a été proposée par Fleszar et Hindi [46]. Pour implémenter

notre VNS, nous avons utilisé les composants logiciels de la plateforme ParadisEO [10].

L'algorithme de VNS proposé dans cette section effectue des mouvements dans un espace de recherche contenant des solutions faisables et non faisables. Le pseudo-code de ce VNS est décrit dans l'algorithme 13. La figure 5.1 donne un exemple illustratif du concept de voisinages multiples utilisé par le VNS.

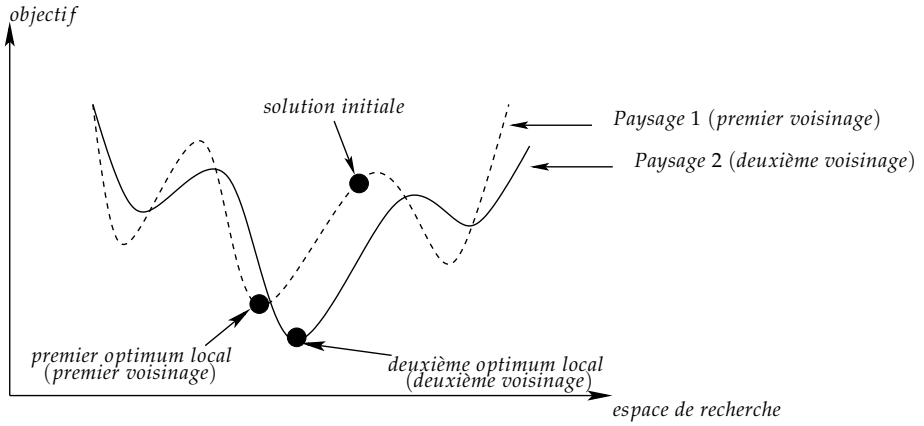


FIGURE 5.1 – *[109] Un exemple de recherche à voisinage variable utilisant deux voisnages. Le premier optimum local est obtenu dans le premier voisnage et le deuxième optimum local est obtenu dans le deuxième voisnage.*

On commence par calculer une solution heuristique, dite σ, nécessitant $U(\sigma)$ bins. Pour ce faire, on utilise un ensemble d'heuristiques gloutonnes détaillées dans la section 5.5.1. On modifie ensuite σ par une méthode de perturbation pendant un certain nombre d'itérations qui dépend d'un paramètre k initialement égal à 1. Les détails de la méthode de perturbation sont présentés dans la section 5.5.2 mais l'idée est la suivante. On note $N_k(\sigma)$ le voisinage contenant l'ensemble des solutions obtenues en : (i) éliminant k de σ et (ii) redistribuant le contenu de ces k bins sur $U(\sigma) - 1$ même si on est obligé de violer la contrainte de fragilité. Différentes stratégies ont été conçues pour chacune des étapes (i) et (ii). On note σ' la solution obtenue et qui appartient à $N_k(\sigma)$, le $k^{ème}$ voisinage de σ.

Il est possible que la nouvelle solution σ' soit infaisable étant donné qu'on est autorisé à violer la contrainte de fragilité en redistribuant le contenu des bins supprimés sur les $U(\sigma) - 1$ bins restants. On essaye ensuite de minimiser cet excès de contenu en utilisant la procédure de recherche locale décrite dans la section 5.5.3. On note σ'' la nouvelle solution obtenue de cette manière.

Si la recherche locale réussit à rendre σ'' faisable, on obtient alors une nouvelle solution heuristique avec un bin de moins, on met à jour la meilleure solution et on réitère avec $k = 1$. Sinon, on réitère tant que le nombre maximum d'itérations n_{iter} n'a pas été atteint. Si σ'' est toujours infaisable, on incrémente la valeur de k par 1 afin d'étendre le voisinage. Le paramètre k est réinitialisé à 1 une fois que sa valeur atteint un seuil maximum $k_{\max}$. En se basant sur des expérimentations préliminaires, nous avons choisi $k_{\max} = n$.

L'algorithme est arrêté une fois que la valeur de la meilleure solution est égale à la valeur d'une borne inférieure ou si un temps limite de 300 secondes est écoulé. L'idée de se déplacer dans un espace de recherche contenant des solutions infaisables et d'essayer de les rendre faisables a été étudiée dans plusieurs travaux et a montré son efficacité (*e.g.* Talbi [109]).

Algorithme 13 : Recherche à voisinage variable pour le BPP-FO

entrée : Une instance de BPP-FO
sortie : Une solution heuristique σ utilisant $U(\sigma)$ bins

1 *Solution de départ* : Calculer une solution de départ σ d'une façon heuristique ;
2 $k = 1$;
3 **tant que** (condition d'arrêt non vérifiée) **faire**
4 $t = 1$;
5 **tant que** ($t \leq n_{iter}$ et σ'' non faisable) **faire**
6 *Perturbation* : Générer une nouvelle solution $\sigma' \in N_k(x)$ avec $U(\sigma) - 1$ bins ;
7 *Recherche locale* : Améliorer σ' afin d'obtenir une solution σ'' ;
8 $t = t + 1$;
9 **si** (σ'' non faisable) **alors**
10 $\sigma \leftarrow \sigma''$;
11 $k = 1$;
12 $k = k + 1$;
13 **si** ($k > k_{\max}$) **alors** $k = 1$
14 **retourner** σ;

5.5.1 Solution initiale

Une solution initiale étant nécessaire afin de faire fonctionner notre VNS, nous avons adapté un ensemble de méthodes heuristiques de la littérature dédiées au problème de bin packing classique et au problème de coloration de graphe. Cette adaptation vise à permettre à ces algorithmes de tenir compte des contraintes spécifiques au $\mathcal{BPP}$-$\mathcal{FO}$. La solution utilisant le moins de bins parmi celles produites par les différents algorithmes sera utilisée comme une solution initiale pour le VNS.

5.5.1.1 Heuristique de type AnyFit

Afin d'adapter toute méthode de type "Fit" (voir section 1.4.4.1) pour le $\mathcal{BPP}$-$\mathcal{FO}$, on remplace tout simplement la contrainte de capacité du bin par la contrainte de fragilité. Pour ce faire : (i) on fixe la capacité du bin à la fragilité du premier objet placé dans le bin, et (ii) on met à jour cette fragilité à chaque fois qu'un nouveau objet de fragilité plus petite que celle du bin est placé dans le bin.

Nous avons adapté les algorithmes classiques *Best Fit*, *Worst Fit* et *Next Fit*. Nous avons obtenu au total 12 solutions heuristiques en considérant les ordres suivants sur les objets :

1. ordre croissant des f_j, ordre décroissant des w_j en cas d'égalité ;
2. ordre décroissant des w_j, ordre croissant des f_j en cas d'égalité ;
3. ordre croissant des rapports f_j/w_j.

5.5.1.2 Heuristique à base de clique

Dans la section 5.4.2, nous avons présenté une borne inférieure en construisant un graphe de conflits G à partir d'une instance de BPP-FO et en calculant une clique maximale K de G avec l'heuristique de Johnson [69]. On reprend cette idée en calculant une borne supérieure valide de la manière suivante. On choisit un premier objet $j \in K$, on détermine son ensemble compatible $\gamma(j)$ et le sous-ensemble $\overline{\gamma}(j)$ des objets les plus grands pouvant être placés avec j (cette dernière étape est réalisée en résolvant le même KP01-FO de la section 5.3). On place j et $\overline{\gamma}(j)$ dans un bin et on réitère avec l'objet suivant dans la clique, jusqu'à ce que tous les objets de $|K|$ soient traités. Si tous les objets de l'instance sont placés dans $|K|$ bins, la solution obtenue est donc optimale étant donné que $L_k = |K|$. Par contre, s'il reste des objets non placés, on les place en utilisant l'heuristique de type First Fit.

5.5.1.3 Heuristique de fusion

L'idée derrière cette heuristique est de fusionner itérativement des couples d'objets pour faire des objets plus grands tant que c'est possible. L'ensemble final d'objets obtenus de cette façon représente une solution heuristique dans laquelle, chaque objet est placé dans un bin seul. On procède de la façon suivante : (i) on choisit deux objets *compatibles* j et k tels que $w_j + w_k \leq \min\{f_j, f_k\}$ et on les fusionne pour avoir un nouvel objet h de taille $w_h = w_j + w_k$ et de fragilité $f_h = \min\{f_j, f_k\}$; (ii) on répète le processus tant qu'il existe des couples à fusionner.

Plusieurs variantes de cette heuristique sont envisageables selon la stratégie adoptée pour la sélection de couples d'objets à fusionner. En particulier, nous adoptons les stratégies suivantes :

1. sélectionner le couple (j,k) avec une valeur minimum pour $|f_j - f_k|$;
2. sélectionner le couple (j,k) avec une valeur minimum pour $(\min\{f_j, f_k\} - w_j - w_k)$.

5.5.1.4 Heuristique à base de sac-à-dos

L'idée utilisée dans la section 5.3 pour la phase de prétraitement peut être utilisée d'une façon itérative simple afin de produire une solution faisable. En particulier, à chaque itération, soit j l'objet de plus petite fragilité et soit $\gamma(j)$ son ensemble compatible. On résout une instance de KP01 avec l'ensemble d'objets $\gamma(j)$ dont les profits

sont les tailles et une capacité de sac égale $f_j - w_j$. On place j et les objets appartenant à la solution optimale de KP01 dans un bin. Le processus est réitéré tant qu'il existe des objets non placés. On tient à préciser qu'en choisissant à chaque itération l'objet j de plus petite fragilité, on peut résoudre un KP01 au lieu d'un KP01-FO.

5.5.2 Procédure de perturbation

Le but de cette méthode de perturbation est de modifier la meilleure solution σ afin de générer une nouvelle solution σ' appartenant au voisinage $N_k(\sigma)$. Ce voisinage est l'ensemble de solutions qui peuvent être obtenues à partir de σ en (i) éliminant k bins et (ii) redistribuant le contenu de ces k bins sur $U(\sigma) - 1$ bins.

La solution σ' est construite comme suit : les $U(\sigma) - k$ bins restants de σ après avoir éliminé k bins sont copiés directement dans σ' ; on ajoute $k - 1$ bins vides à σ' ; le contenu des k bins éliminés est ensuite distribué sur les $U(\sigma) - 1$ bins de σ' en se permettant de violer la contrainte de fragilité.

Nous avons conçu différentes stratégies pour effectuer les étapes (i) et (ii) décrites plus haut afin d'obtenir des méthodes de diversification. Nous rappelons que $J(i)$ est l'ensemble d'objets affectés à un bin i. La première étape de la procédure de perturbation consiste à :

1. éliminer k avec une probabilité proportionnelle à $\min_{j \in J(i)}\{f_j\}$;
2. éliminer k avec une probabilité proportionnelle à $\left(\min_{j \in J(i)}\{f_j\} - \sum_{j \in J(i)} w_j\right)$;
3. éliminer k aléatoire.

On utilise une fonction objectif à deux niveaux pour redistribuer le contenu des bins éliminés. Supposons qu'on ait besoin d'évaluer le placement d'un objet j dans un bin i, notre fonction objectif, notée $\Omega(j,i)$, calcule sur un premier niveau le nombre d'objets en conflit avec j dans le bin i, *i.e.* $\left|\{k \in J(i) : w_j + w_k > \min\{f_j, f_k\}\}\right|$. En cas d'égalité, $\Omega(j,i)$ calcule sur un deuxième niveau l'excès de contenu , *i.e.* $\max\left(0; \sum_{j \in J(i)} w_j - \min_{j \in J(i)}\{f_j\}\right)$. Un placement d'un objet j dans un bin i est faisable si et seulement si les valeurs obtenues aux deux niveaux sont égales à 0.

Les différentes stratégies adoptées pour effectuer la deuxième étape de notre méthode de perturbation sont :

1. *meilleur-meilleur* : à chaque itération, on considère toutes les affectations possibles d'un objet j à un bin i et on choisit celle minimisant $\Omega(j,i)$;
2. *premier-meilleur* : à chaque itération, on choisit l'objet j ayant la fragilité minimum parmi ceux appartenant aux bins éliminés et on le place dans le bin i minimisant $\Omega(j,i)$;
3. *premier-premier* : similaire à *first-best* sauf qu'ci, on s'arrête une fois qu'on trouve une affectation faisable d'un objet j. Sinon, on choisit le bin i minimisant $\Omega(j,i)$;
4. *aléatoire* : affecter chaque objet à un bin aléatoire.

L'impact de chacune de ces stratégies sur les résultats du VNS est discuté dans la section 5.7.2.

5.5.3 Procédure de recherche locale

La solution obtenue après l'exécution de la procédure de perturbation peut être infaisable. En effetn un excès de contenu peut se produire dans un ou plusieurs bins. On essaye de rendre faisable la solution en utilisant un méthode de recherche locale qui *échange* des objets provenant de deux bins différents. Soit

$$\widetilde{W}(i) = \max\left(0; \sum_{j \in J(i)} w_j - \min_{j \in J(i)}\{f_j\}\right) \tag{5.31}$$

l'excès de contenu associé à un bin i. Soit i_1 et i_2 deux bins, tels qu'au moins un des deux a un excès de contenu strictement positif, *i.e.* $\max\{\widetilde{W}(i_1), \widetilde{W}(i_2)\} > 0$. Notre procédure, notée *swap* (ℓ_1, ℓ_2) dans la suite, échange ℓ_1 objets placés dans i_1 avec ℓ_2 objets placés dans i_2. Un mouvement est dit *améliorant* s'il réussit à réduire la valeur de $\max\{\widetilde{W}(i_1), \widetilde{W}(i_2)\}$.

Durant l'exécution d'un swap (ℓ_1, ℓ_2), on garde la liste des bins triés par ordre croissant de $\widetilde{W}(i)$ et on tente tout d'abord d'échanger des objets provenant du premier et du deuxième bin. On considère ensuite le premier et le troisième bin et ainsi de suite. On effectue des vérifications rapides des tailles des objets et l'excès de contenu des bins afin de ne tenter que des swaps améliorants. Une fois un swap tenté on a besoin de réévaluer les fragilités des bins impliqués dans ce mouvement. Cette évaluation est effectuée d'une façon rapide en gardant une liste d'objets placés dans chacun des bins triée par ordre décroissant de fragilité (ordre initial donc pas besoin de trier). La stratégie de sélection d'un mouvement parmi les mouvements possibles est la stratégie dite *première amélioration*, *i.e.* on choisit le premier mouvement améliorant et on relance.

Nous avons développé l'ensemble des swaps suivants $\{(1,0), (1,1), (1,2), (2,1), (2,2)\}$. Les différents swaps sont appliqués par ordre croissant de leur complexité, commençant ainsi par $(1,0)$. La phase de recherche locale est arrêtée une fois que le problème d'excès de contenu est réglé ou bien si aucun mouvement améliorant n'a été trouvé. Pour évaluer le meilleur compromis entre la précision de l'approche de recherche locale et de sa complexité, nous avons testé différentes stratégies, notamment :

1. swap $(1,0)$;
2. swaps $(1,0)$ et $(1,1)$;
3. swaps $(1,0)$, $(1,1)$, $(1,2)$ et $(2,1)$;
4. swaps $(1,0)$, $(1,1)$, $(1,2)$, $(2,1)$ et $(2,2)$.

Nous analysons expérimentalement l'impact de chacune de ces stratégies sur le comportement du VNS dans la section 5.7.2.

5.6 Algorithme de génération de colonnes

Dans la section 5.2.3, nous avons mentionné que le modèle (1.10)-(1.12) nécessite un nombre exponentiel de variables, nous proposons ainsi dans cette section une méthode de génération de colonnes afin de résoudre ce modèle.

Nous initialisons le modèle par un ensemble $\widetilde{P} \subseteq P$ de patterns. Nous relâchons ensuite les contraintes d'intégrité (1.12) en les remplaçant par

$$z_p \geq 0, \forall p \in \widetilde{P}. \tag{5.32}$$

On se permet d'éliminer les contraintes $z_p \leq 1$ puisqu'elles sont redondantes. En fait, on peut toujours remplacer une solution où telle que $z_p > 1$ avec une meilleure solution telle que $z_p = 1$. On associe les variables duales π_j $(j = 1, \ldots, n)$ aux contraintes (1.11).

On procède de la façon itérative suivante : on résout le problème maître décrit en haut en l'initialisant par un sous-ensemble de colonnes et on teste s'il existe une nouvelle colonne à coût réduit négatif. Dans ce cas, on ajoute cette colonne au problème maître et on réitère ; sinon, l'optimalité du modèle est atteinte. Le coût réduit d'une colonne p est défini par

$$\bar{c}_p = 1 - \sum_{j=1}^{n} \pi_j a_{jp}. \tag{5.33}$$

Une colonne est ajoutée problème maître si elle est valide et si coût réduit est négatif. L'existence d'un tel pattern est déterminé en résolvant un KP01-FO (voir section 5.1) avec la fonction objectif suivante

$$\max \sum_{j=1}^{n} \pi_j a_j \tag{5.34}$$

sous les contraintes

$$\sum_{j=1}^{n} w_j a_j \leq \min_{j=1,\ldots,n} \{f_j a_j\} \tag{5.35}$$

$$a_j \geq 0, j = 1, 2, \ldots, n. \tag{5.36}$$

Dans la section 5.6.1, on présente différentes méthodes à utiliser pour résoudre le KP01-FO. Dans la section 5.6.2 on propose quelques idées heuristiques à utiliser pour initialiser $\widetilde{P}$ et pour améliorer le processus de génération de colonnes par le biais de coupes duales.

5.6.1 Résoudre le sous-problème de pricing

Le sous-problème de pricing consiste en une instance de KP01-FO, pour lequel on propose ici deux modèles linéaires en nombres entiers et un algorithme de programmation dynamique. Les résultats numériques obtenus en utilisant ces trois approches figurent dans la section 5.7.

5.6.1.1 Modèles mathématiques

Dans la section 5.2.1, nous avons présenté deux modèles compacts de programmation linéaire en nombres entiers pour résoudre le $\mathcal{BPP}$-$\mathcal{FO}$, le deuxième modèle (5.7)-(5.12) étant meilleur que le premier modèle (5.1)-(5.6).

Un premier modèle compact est obtenu en utilisant une variable binaire ξ_j prenant la valeur 1 *si l'objet $j \in I$ est placé dans le sac* et 0 sinon. Le KP01-FO est modélisé comme suit :

$$\max \sum_{j=1}^{n} \pi_j \xi_j \tag{5.37}$$

$$\sum_{j=1}^{n} w_j \xi_j \leq f_{\max} + \xi_k (f_k - f_{\max}) \quad k = 1, \ldots, n \tag{5.38}$$

$$\xi_j \in \{0,1\} \quad j = 1, \ldots, n. \tag{5.39}$$

La différence principale par rapport au problème de sac-à-dos classique KP01 réside dans le fait qu'ici, on a besoin de n contraintes pour imposer la capacité maximum. En effet, à chaque fois qu'un objet k est placé dans le sac, les contraintes (5.38) imposent que la somme des tailles des objets placés dans le sac ne dépasse pas f_k.

Une meilleure formulation est obtenue comme suit. Dans un premier temps, on rappelle que les objets sont triés par ordre croissant de f_j, et par ordre décroissant de w_j en cas d'égalité. On définit un variable binaire β_j qui prend la valeur 1 *si l'objet $j \in I$ est l'objet de plus petite fragilité dans le sac* et 0 sinon. On définit aussi une variable binaire α_j qui prend la valeur 1 *si l'objet $j \in I$ est placé dans le sac sans qu'il ait la plus petite fragilité*. On obtient ainsi :

$$\max \sum_{j=1}^{n} \pi_j (\alpha_j + \beta_j) \tag{5.40}$$

$$\sum_{j=1}^{n} \beta_j = 1 \tag{5.41}$$

$$\sum_{j=1}^{n} w_j \alpha_j \leq \sum_{j=1}^{n} \beta_j (f_j - w_j) \tag{5.42}$$

$$\alpha_k + \sum_{j=k}^{n} \beta_j \leq 1 \quad k = 1, \ldots, n \tag{5.43}$$

$$\alpha_j \in \{0,1\} \quad j = 1, \ldots, n \tag{5.44}$$

$$\beta_j \in \{0,1\} \quad j = 1, \ldots, n. \tag{5.45}$$

Les contraintes (5.41) imposent le fait qu'un seul objet de fragilité minimum soit placé dans le sac. Les contraintes (5.42) imposent que la somme des tailles des objets placés dans le sac ne dépasse la fragilité d'aucun de ces objets.

Par hypothèse, $\alpha_j + \beta_j \leq 1$, pour $j = 1, \ldots, n$. Cette considération est exprimée par les contraintes (5.43), ce qui impose le fait que si un objet k est placé dans le sac sans

qu'il soit de plus petite fragilité (*i.e.* si $\alpha_k = 1$), alors aucun autre objet de fragilité plus grande ne peut être considéré comme l'objet de plus petite fragilité.

5.6.1.2 Programmation dynamique

Le KP01-FO peut être aussi résolu par une approche de programmation dynamique. Une méthode triviale, pas nécessairement efficace, consiste à considérer à tour de rôle toutes les valeurs possibles de f_i et résoudre à chaque fois le KP01 lié à l'ensemble d'objets $\{i : w_i \leq f_i\}$ et à un bin de taille f_i. Dans le pire des cas, la programmation dynamique est exécutée n fois, une fois par valeur de fragilité. La complexité temporelle de cette procédure est de l'ordre de $\mathcal{O}(n^2 \times f_{\max})$.

Une méthode plus efficace peut être appliquée si les objets sont triés par ordre décroissant de fragilité. L'idée est la suivante : on construit un tableau de programmation dynamique classique, et on prend la valeur maximum parmi toutes les valeurs obtenues sur les états qui correspondent à un changement de fragilité.

Soit $\psi(i, C)$ la solution optimale de KP01 pour les i premiers objets et un sac de taille C. Pour simplifier la notation, nous considérons que $\psi(i, C) = -\infty$ si $C < 0$) ou si l'état (i, C) ne peut pas être atteint. On note $OPT(KP01 - FO)$ la solution optimale du problème de sac-à-dos avec conflits.

Proposition 5.6.1 *Si les objets sont triés par ordre décroissant de fragilité, le résultat suivant est valide :*

$$OPT(KP01{-}FO) = \max_{\alpha=0,\ldots,f_{\max}} \{\psi(\max\{i : f_i \geq \alpha\}, \alpha)\} \tag{5.46}$$

Démonstration. Nous montrons que si une solution optimale été trouvée à un état donné, une autre solution équivalente est trouvée qui respecte l'équation 5.46. Deux cas à considérer :

- Supposons qu'une solution optimale a été trouvée au niveau de l'état (j, α) tel que $j < \max\{i : f_i \geq \alpha\}$. En parcourant les objets $j+1, \ldots, i$, l'état $(\max\{i : f_i \geq \alpha\}, \alpha)$ de même coût est obtenu.
- Supposons qu'une solution optimale a été trouvée au niveau de l'état(j, α) tel que $j > \max\{i : f_i \geq alpha\}$. Il n'existe pas un objet i tel que $f_i < \alpha$, sinon le pattern ne serait pas valide. Par conséquent, il n'existe pas d'objet de label $\max\{i : f_i \geq \alpha\} + 1, \ldots, j$ qui appartient au sac-à-dos et donc l'état $(\max\{i : f_i \geq \alpha\}, \alpha)$ a le même coût que (j, α).

□

Chaque état (j, α) n'est exploré qu'une seule fois. En utilisant une formulation récursive comme dans le cas de KP01, on obtient une complexité temporelle de l'ordre de $\mathcal{O}(n \log(n) + f_{\max} \times n)$ ou $\mathcal{O}(f_{\max} \times n)$ si les objets sont initialement triés par ordre décroissant de fragilité.

La figure 5.2 montre la tableau de programmation dynamique utilisé pour résoudre l'instance de BP01-FO du tableau 5.1. Les états liés à l'objet 4 (resp. 5) et aux tailles plus grandes que f_4 (resp f_5) sont interdites. En se basant sur le résultat de la

proposition 5.46, on sait qu'un état optimal se trouve parmi les états les plus à droite du tableau.

TABLE 5.1 – *Une instance de KP01-FO*

i	w_i	f_i	π_i
1	2	8	1
2	4	8	1
3	6	8	1
4	4	6	2
5	2	4	1

Si un objet 1 est considéré, deux états seulement sont possibles : placer 1 (profit 1) ou non (profit 0). Une fois la première colonne calculée, la deuxième peut être calculée en ajoutant ou non l'objet 2 à chaque état de la première colonne 1, et ainsi de suite.

5.6.2 Stabilisation de la génération de colonnes

La génération de colonnes assure une rapide amélioration de la valeur du problème maître durant les premières itérations. Cependant, elle peut passer beaucoup de temps à générer des colonnes qui n'améliorent que faiblement la valeur de l'objectif. Cette observation est appelé l'effet d'amortissement ou l'effet de tailing-off.

Les problèmes de convergence de la génération de colonnes proviennent essentiellement de la dégénérescence primale et de la qualité des colonnes générées qui n'appartiendront pas à la base optimale. Sous l'aspect dual, ces problèmes de convergence se traduisent par des oscillations des valeurs des variables duales.

Il existe dans la littérature plusieurs approches dites de stabilisation qui visent à diminuer les fluctuations des variables duales. Ces approches guident la méthode de génération de colonnes pour générer des solutions proches de la meilleure solution trouvée jusqu'à l'itération courante. Dans [111], les auteurs font un état de l'art détaillé sur les différentes méthodes de stabilisation de la méthode de génération de colonnes.

La méthode des *coupes duales* introduite par Valerio de Carvalho [113] est l'une des idées simples et efficaces pour stabiliser la génération de colonnes. L'idée consiste à ajouter des coupes duales au problème maître (des colonnes dans le primal) afin d'éliminer des solutions duales dominées par d'autres solutions.

Le concept de coupes duales peut être adapté au cas de bin packing avec objets fragiles comme suit.

Proposition 5.6.2 *Pour un objet donné j, s'il existe un ensemble $S \subset I$ tel que $\sum_{i\in S} w_i \leq w_j$ et $\min_{i\in S} f_i \geq f_j$ alors*

$$\sum_{i\in S} \pi_i \leq \pi_j \tag{5.47}$$

est une coupe duale valide pour (1.10)-(1.12).

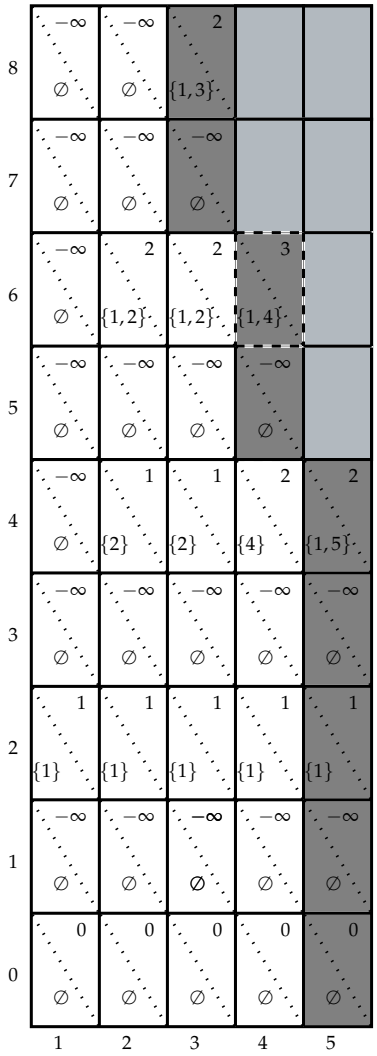


FIGURE 5.2 – *Un tableau illustrant les états trouvés par l'algorithme de programmation dynamique en résolvant l'instance de KP01-FO décrite dans le tableau 5.1. Chaque ligne correspond à un niveau de fragilité allant de* 0 *jusqu'à* f_{max} *et chaque colonne i correspond aux i premiers objets de I. Chaque cellule est divisée en deux sous-cellules : dans celle du bas nous montrons l'ensemble des objets placés dans le sac et dans celle du haut nous montrons le profit rapporté par ces objets. Les cellules à couleur grise foncée représentent les états dominants (états contenant une solution optimale), et la couleur gris clair pour les cellules interdites.*

Démonstration. La preuve de [113] sur la validité des coupes duales repose sur le fait que dans tout pattern valide contenant un objet j, j peut être remplacé par l'intégralité des objets de S. La preuve que nous proposons est similaire, nous montrons donc uniquement que la condition est vérifiée : "Soit P un pattern valide contenant un objet j, et S l'ensemble défini ci-dessus. Si P est un pattern valide contenant j, $\sum_{k\in P\setminus\{j\}} w_k + w_j \leq \min\{\min_{k\in P\setminus\{j\}}\{f_k\}, f_j\}$."

En remplaçant j par l'ensemble S, et en considérant les hypothèses sur l'ensemble S, on obtient : $\sum_{k\in P\setminus\{j\}} w_k + \sum_{i\in S} w_i \leq \min\{\min_{k\in P\setminus\{j\}}\{f_k\}, \min_{i\in S}\{f_i\}\}$, ce qui signifie que le pattern obtenu est valide. □

Plusieurs coupes de ce type peuvent être appliquées. D'une manière pratique, nous utilisons les coupes de Type I et II décrites dans [113]. Dans les coupes de I, l'ensemble S est de taille 1. Les coupes de type II considèrent deux objets dans S. Si le nombre de coupes de type II est très grand, nous n'appliquons qu'un sous-ensemble d'elles. Pour le $\mathcal{BPP}$-$\mathcal{FO}$, cette taille n'est pas très grande car les conditions imposées sur S sont plus restrictives que dans le cas du cutting-stock. Par conséquent, nous appliquons toutes les coupes de type II.

5.7 Partie expérimentale

Dans cette section, nous présentons les résultats numériques obtenus par les différents algorithmes décrits dans ce chapitre. Toutes les méthodes ont été codées en C++ et exécutées sur un processeur Intel(R) Core(TM)2 Duo CPU T6400 @ 2.13GHz avec 2GO de RAM. Étant donné que ce travail est le premier travail qui traite le $\mathcal{BPP}$-$\mathcal{FO}$ d'un point de vue numérique, nous proposons dans la section 5.7.1 un jeu d'essai que nous avons généré pour ce problème. Dans la section 5.7.2 nous décrivons les étapes effectuées afin de paramètrer le VNS. Ce VNS a été implémenté en se basant sur les composants logiciels de la plateforme ParadisEO [10]. Les résultats globaux des algorithmes sont détaillés dans la section 5.7.3.

5.7.1 Jeu d'essai compétitif

Il est d'une grande importance de comprendre l'impact de la contrainte de fragilité sur la difficulté d'instance de $\mathcal{BPP}$. Pour ce faire, nous avons construit un ensemble d'instances de $\mathcal{BPP}$-$\mathcal{FO}$ à base d'instances de $\mathcal{BPP}$ en essayant différentes fonctions pour générer des fragilités pour les objets.

Nous avons utilisé le jeu d'essai classique "*set 1*" de Scholl *et al.* [104] dédié au $\mathcal{BPP}$. Ce jeu d'essai a été utilisé dans de nombreux articles traitant des méthodes exactes et heuristiques pour résoudre le $\mathcal{BPP}$. Ce jeu d'essai peut être téléchargé à l'adresse `http://www.fe.up.pt/~esicup`, et comprend des instances avec un nombre d'objets $n = 50, 100, 200, 500$, un bin de capacité $C = 100, 120, 150$, et des tailles d'objets uniformément distribuées sur les intervalles $[1, 100]$, $[20, 100]$, $[30, 100]$. Pour chaque configuration comprenant un nombre d'objets n, une capacité de bin et des tailles d'objets, 20 instances sont générées, donnant un total de 720 instances. Pour notre jeu, nous nous intéressons aux 5 premières instances parmi les 20 pour chaque configuration, sans tenir compte des instances des cas où $n = 500$, car, comme nous allons le prouver dans la suite, les instances de petite taille sont déjà très difficiles à résoudre. Nous obtenons ainsi un total de 135 instances.

Pour générer le jeu d'essai pour $\mathcal{BPP}$-$\mathcal{FO}$, nous considérons les tailles des objets des instances de $\mathcal{BPP}$ et nous générons des fragilités selon différentes stratégies. Nous nous concentrons sur la relation entre la fragilité f_j d'un objet j et sa taille w_j, car ceci peut avoir un impact important sur la difficulté de l'instance résultante. Nous

avons créé différentes configurations dans lesquelles la fragilité d'un objet est corrélée ou non avec sa taille :

instances non corrélées : soit $w_{\max} = \max_{j=1,\dots,n}\{w_j\}$ et multiplions la taille C d'un bin par une valeur entière k_2. Nous générons ensuite des fragilités appartenant à différentes sous-intervalles de l'intervalle $[w_{\max}, k_2C]$. Plus précisément, nous fixons f_j à une valeur entière sélectionnée avec une distribution uniforme dans l'intervalle $[w_{\max} + k_1(k_2C - w_{\max})/5, k_2C]$, for $j = 1,\dots,n$, et on teste toutes les valeurs $k_1 = 0, 1, \dots, 4$ et $k_2 = 1, 2, \dots, 5$;

instances faiblement corrélées : nous fixons f_j à une valeur entière sélectionnée avec une distribution uniforme dans l'intervalle $[k_1w_j, k_2C]$, for $j = 1,\dots,n$. Nous testons les valeurs $k_1 = 1, 2, \dots, 5$ et $k_2 = k_1, k_1 + 1, \dots, 5$;

instances fortement corrélées : nous fixons $f_j = k_1w_j$, for $j = 1,\dots,n$, et testons les valeurs $k_1 = 1, 2, \dots, 5$.

En considérant les différentes valeurs de k_1 et k_2, nous obtenons un total de 50 différents cas. Pour tester la difficulté des ces cas, nous nous concentrons sur les instances de $\mathcal{BPP}$ avec $n = 100$ objets, car elles se sont avérées être un jeu d'essai significatif. Nous avons obtenus de cette façon $45 \times 50 = 2025$ instances de $\mathcal{BPP}$-$\mathcal{FO}$.

Nous avons tenté de résoudre ce jeu d'essai en résolvant le modèle linéaire (5.7)–(5.12) avec Cplex 10 avec un temps de calcul limite de 5 minutes CPU. Les résultats obtenus sont présentés dans le tableau 5.2. La partie gauche du tableau montre le rapport d'instances résolues à l'optimalité sur 45 instances. La partie droite montre le pourcentage moyen de gap, évalué par $100(U - L)/U$, où U (resp. L) est une borne supérieure (inférieure) obtenue par Cplex au bout de 5 minutes CPU.

Les instances fortement corrélées sont très faciles à résoudre et ont été toutes résolues par Cplex sans dépasser le temps limite, sauf pour le cas d'une instance avec $k_1 = 4$. Les instances non corrélées sont plus difficiles à résoudre. On peut voir sur le tableau que, dans le cas où les valeurs de k_1 et k_2 sont petites ou bien grandes, les instances sont moins difficiles que les autres. Ceci peut aussi être vu dans la partie droite du tableau. Dans le cas de deux classes, Cplex n'a pas été capable de résoudre 20% des instances à l'optimalité. Les instances faiblement corrélées deviennent de plus en plus difficiles, avec trois classes pour lesquelles un rapport de solutions optimales de 20% n'a pas été atteint. Les pourcentages moyens des gaps sont un peu plus élevés que ceux dans le cas des instances non corrélées.

En se basant sur les résultats ci-dessus, nous avons choisi les cinq classes difficiles suivantes :

Classe 1 : non corrélée avec $k_1 = 1$ et $k_2 = 3$;

Classe 2 : non corrélée avec $k_1 = 1$ et $k_2 = 5$;

Classe 3 : faiblement corrélée avec $k_1 = 1$ et $k_2 = 5$;

Classe 4 : faiblement corrélée avec $k_1 = 3$ and $k_2 = 3$;

Classe 5 : faiblement corrélée avec $k_1 = 3$ and $k_2 = 4$.

TABLE 5.2 – *Résultats obtenus par le modèle* (5.7)–(5.12) *sur* 2025 *instances avec* $n = 100$. *Chaque cellule contient un résultat moyen sur* 45 *instances.*

		rapport de solutions optimales					pourcentage de gap				
	$k_2 \backslash k_1$	*0*	*1*	*2*	*3*	*4*	*0*	*1*	*2*	*3*	*4*
non corrélées	*1*	0.93	0.89	0.87	0.89	0.89	0.15	0.26	0.32	0.28	0.29
	2	0.69	0.38	0.31	0.20	0.38	0.92	2.09	2.64	2.85	2.37
	3	0.33	0.11	0.31	0.42	0.51	2.16	3.74	3.07	2.82	2.48
	4	0.33	0.27	0.29	0.60	0.78	2.55	3.82	3.88	2.40	1.52
	5	0.51	0.18	0.49	0.67	0.78	2.22	4.74	3.20	2.48	1.75
	$k_2 \backslash k_1$	*1*	*2*	*3*	*4*	*5*	*1*	*2*	*3*	*4*	*5*
faiblement corrélées	*1*	0.44					1.70				
	2	0.24	0.22				2.62	3.96			
	3	0.22	0.38	0.07			3.29	3.43	4.62		
	4	0.20	0.44	0.18	0.36		3.67	3.28	4.08	3.53	
	5	0.07	0.53	0.29	0.29	0.44	4.32	3.03	3.91	4.17	3.60
	k_1	*1*	*2*	*3*	*4*	*5*	*1*	*2*	*3*	*4*	*5*
fortement corrélées		1.00	1.00	1.00	0.98	1.00	0.00	0.00	0.00	0.10	0.00

Nous avons considéré les 135 instances de $\mathcal{BPP}$ et avons obtenu ainsi $135 \times 5 = 675$ instances de $\mathcal{BPP}$-$\mathcal{FO}$. Ces instances sont disponibles pour le téléchargement et l'utilisation libre sur la page web `http://www.or.unimore.it`.

Pour insister sur le fait que les instances générées sont difficiles, nous avons tenté de les résoudre en les traitant avec la procédure de prétraitement décrite dans la section 5.3 et en résolvant ensuite le modèle (5.7)–(5.12). Le temps limite est encore 5 minutes CPU, temps de prétraitement inclus. Le tableau 5.3 montre les résultats obtenus. Les entêtes des tableaux sont les mêmes que celles du tableau 5.2.

Dans presque 50 cas, le prétraitement a pu améliorer le comportement du modèle compact, en terme de nombre de solutions optimales obtenues et du gap moyen. En regardant le tableau 5.3, on peut remarquer que les classes difficiles choisies dans le tableau 5.2 reste difficiles même après avoir appliqué le prétraitement sur les instances avant de les résoudre. En particulier, considérons le cas remarquable de la classe faiblement corrélée avec $k_1 = 3$ et $k_2 = 3$, on peut noter que le rapport de solutions optimales reste le même (7%) tandis que le gap moyen augmente de 4.62% à 4.89%.

Nous pouvons conclure que le prétraitement est efficace car il aide la résolution des instances de $\mathcal{BPP}$-$\mathcal{FO}$. Le nombre d'objets a aussi été réduit de 100 à 92, en moyenne. De plus, le gap moyen passe de 2.41% à 2.18%, et la nombre total de solutions optimales trouvées passe de 1006 à 1140. Parmi ces 1140 instances, 115 ont été résolues par le prétraitement seul.

TABLE 5.3 – *Résultats obtenus par la procédure de prétraitement suivie du modèle* (5.7)–(5.12) *sur* 2025 *instances avec* $n = 100$. *Chaque cellule contient un résultat moyen sur* 45 *instances.*

		ratio of optimal solutions					percentage gaps				
	$k_2 \backslash k_1$	*0*	*1*	*2*	*3*	*4*	*0*	*1*	*2*	*3*	*4*
non corrélées	*1*	1.00	0.96	0.96	1.00	0.93	0.00	0.12	0.12	0.00	0.19
	2	0.67	0.42	0.49	0.31	0.53	1.01	2.06	1.91	2.74	1.87
	3	0.40	0.20	0.20	0.49	0.44	2.17	3.37	3.82	2.55	3.08
	4	0.38	0.29	0.40	0.73	0.71	2.61	3.68	3.48	1.75	1.93
	5	0.53	0.38	0.58	0.76	0.84	2.22	3.48	2.71	1.87	1.34
	$k_2 \backslash k_1$	*1*	*2*	*3*	*4*	*5*	*1*	*2*	*3*	*4*	*5*
faiblement corrélées	*1*	0.67					1.12				
	2	0.31	0.40				2.43	3.21			
	3	0.27	0.29	0.07			3.17	4.14	4.89		
	4	0.29	0.58	0.20	0.42		3.27	2.70	4.20	3.33	
	5	0.20	0.56	0.42	0.33	0.73	3.61	3.08	3.31	4.04	1.84
	k_1	*1*	*2*	*3*	*4*	*5*	*1*	*2*	*3*	*4*	*5*
fortement corrélées		1.00	1.00	1.00	1.00	1.00	0.00	0.00	0.00	0.00	0.00

5.7.2 Paramétrage du VNS

Cette section est dédiée à une étude de paramétrage du VNS proposé dans la section 5.5. Nous nous concentrons sur la procédure de perturbation, la méthode de recherche locale et le nombre d'itérations pour chaque valeur de k. Pour ce faire, nous avons effectué des expérimentations sur la première instance de chaque groupe de cinq instances ayant le même nombre d'objets, la même capacité du bin, les mêmes distributions pour les tailles des objets et la même classe de fragilité. Nous avons essayé de résoudre ces 135 instances avec différents paramétrages du VNS. Sur ces 135 instances, 65 ont été résolues à l'optimalité par les prétraitements, les bornes inférieures et la phase initiale du VNS. Ces 65 instances n'entrent pas en jeu dans la phase de paramétrage et nous nous concentrons ainsi sur les 70 instances restantes en essayant différentes configurations du VNS avec un temps d'exécution limite d'une minute CPU.

Le tableau 5.4 montre les premiers résultats obtenus. Chaque ligne correspond à une configuration particulière du VNS. La colonne U_{VNS} représente la somme, pour 70 instances, de la borne supérieure trouvée par la configuration correspondante. La colonne *sec* représente le temps de calcul total requis et la colonne *opt* représente le nombre total d'instances résolues à l'optimalité.

Pour déterminer la meilleure stratégie de perturbation ainsi que la meilleure méthode de recherche locale, nous avons défini une configuration *initiale* du VNS. Dans cette configuration, on utilise une recherche locale de type 4, *i.e.* une recherche locale qui utilise tous les swaps possibles $(1,0)$, $(1,1)$, $(1,2)$, $(2,1)$ et $(2,2)$. Le nombre d'itérations est $n_{iter} = 12$ et nous appliquons à chaque itération une des 12 options

TABLE 5.4 – *Evaluation des différentes options de la procédure de perturbation et de recherche locale du VNS. Chaque cellule contient un résultat sur un total de 70 instances.*

éliminer bins	redistribuer objets	recherche locale	U_{VNS}	*sec*	*opt*
les 12 options, une à la fois		4	2196	3949.6	9
1	1	4	2197	3897.3	9
1	2	4	2198	4031.2	6
1	3	4	2192	3907.2	11
1	4	4	2200	4156.3	5
2	1	4	2203	4107.9	3
2	2	4	2203	4167.4	2
2	3	4	2203	4107.7	3
2	4	4	2199	4080.1	6
3	1	4	2198	3957.0	7
3	2	4	2199	4078.9	6
3	3	4	2192	3941.0	10
3	4	4	2199	4044.4	6
les 12 options, une à la fois		1	2203	4092.1	3
les 12 options, une à la fois		2	2196	3849.3	9
les 12 options, une à la fois		3	2194	3786.0	11

pour la méthode de perturbation dans l'ordre. Cette configuration qui correspond à la première ligne du tableau a été capable de résoudre 9 instances à l'optimalité et d'obtenir un total de 2196 bins. La somme des meilleures bornes inférieures obtenues pour chacune des instances par les algorithmes de la section 5.4 est 2120, et la somme des bornes supérieures obtenues par les heuristiques de la section 5.5.1 est 2206.

Le deuxième groupe de lignes dans le tableau 5.4 montre les résultats obtenus par différentes options pour la procédure de perturbation (voir section 5.5.2 pour mieux comprendre la signification des nombres représentant les options). D'après le tableau, nous pouvons remarquer que l'option 2 pour l'élimination de bins, *i.e.* éliminer k bins avec une probabilité proportionnelle à $(\min_{j\in J(i)}\{f_j\} - \sum_{j\in J(i)} w_j)$, n'est pas efficace. Les meilleures options sont définies par les couples $(1,1)$ (éliminer k bins avec une probabilité proportionnelle à $\min_{j\in J(i)}\{f_j\}$, *meilleur-meilleur*), $(1,3)$ (éliminer k bins avec une probabilité proportionnelle à $\min_{j\in J(i)}\{f_j\}$, *premier-premier*) et $(3,3)$ (éliminer k bins aléatoires, *premier-premier*).

Le troisième groupe de lignes dans le tableau 5.4 montre les résultats obtenus par différentes stratégies de recherche locale. Il est tout à fait normal que la configuration qui correspond à l'option 3, *i.e.* swaps $(1,0)$, $(1,1)$, $(1,2)$ et $(2,1)$, soit celle qui obtient les meilleurs résultats. Nous avons choisi de ne pas appliquer le swap $(2,2)$ car le temps de calcul entraîné par ce swap est trop important par rapport à l'amélioration apportée.

En se basant sur les observations ci-dessus, la configuration finale du VNS consiste

à perturber les solutions courantes en utilisant à chaque fois une des trois options $(1,1)$, $(1,3)$ et $(3,3)$, et à appliquer l'option 3 pour la recherche locale. Cette configuration a été testée sur les 70 instances avec un temps limite d'une minute CPU, en variant le nombre d'itérations n_{iter} pour chaque valeur de k. Le tableau 5.5 montre les résultats obtenus par cette exécution.

TABLE 5.5 – *Evaluation des différentes options pour le nombre d'itérations du VNS. Chaque cellule contient un résultat pour un total de* 70 *instances.*

éliminer bins	redistribuer items	recherche locale	n_{iter}	U_{VNS}	*sec*	*opt*
options (1,1), (1,3) et (3,3), une à la fois		3	3	2190	3606.8	15
options (1,1), (1,3) et (3,3), une à la fois		3	6	2191	3690.7	11
options (1,1), (1,3) et (3,3), une à la fois		3	9	2190	3634.6	13
options (1,1), (1,3) et (3,3), une à la fois		3	12	2191	3727.0	12
options (1,1), (1,3) et (3,3), une à la fois		3	15	2193	3754.3	10
options (1,1), (1,3) et (3,3), une à la fois		3	18	2191	3702.8	12

Nous pouvons remarquer d'après ce test que les choix adoptés par les configurations précédentes ont assuré une amélioration en terme de solutions optimales obtenues, qui a augmenté de 11 (voir tableau 5.4) à 15. Le meilleur choix consiste à choisir $n_{iter} = 3$.

En se basant sur tous ces résultats, nous adoptons la configuration qui utilise $n_{iter} = 3$ itérations pour chaque valeur de k, et une option parmi $(1,1)$, $(1,3)$ et $(3,3)$ dans l'ordre à chaque itération, et qui essaye d'améliorer la solution courante par l'option 3 pour la recherche locale. Après avoir effectué plusieurs tests avec plusieurs valeurs pour le temps de calcul, nous avons choisi de fixer ce temps à 300 secondes CPU.

Nous avons aussi testé le VNS pour un temps limite de 120 secondes CPU sur le jeu d'essai classique de bin packing. Les résultats sont rapportés dans le tableau 5.6. Les instances sont disponibles sur la page de ESICUP, `http://www.fe.up.pt/~esicup`. Dans ce tableau, nous donnons le nom du jeu d'essai, le nombre d'instances pour chaque groupe (colonne # inst), la somme des bornes inférieures (colonne L), la somme des bornes supérieures obtenues par le VNS (colonne U), le temps de calcule moyen en secondes CPU (colonne sec), le nombre total de solutions optimales obtenues et le pourcentage de gap (colonne opt et % gap, respectivement). Malgré le fait que le VNS a été conçu pour résoudre un problème d'optimisation différent, son comportement est satisfaisant dans le cas de $\mathcal{BPP}$, particulièrement en ce qui concerne les jeux d'essai set1, set2 et 0120, u500 et u1000, où le VNS atteint presque toutes les solutions optimales. Cependant, le mauvais comportement sur les instances de t60, t120, t249 et t501 peut être justifié par la structure particulière de ces instances (seuls 3 objets peuvent être placés dans un bin), ce qui n'est pris en compte par aucune de nos heuristiques.

TABLE 5.6 – *Comportement du VNS sur des jeux d'essai de $\mathcal{BPP}$.*

Jeu d'essai	# inst	L	U	sec	opt	%gap	% opt
set1	720	78378	78412	5474.08	686	0.04	95.28
set2	480	20246	20271	4419.79	461	0.12	96.04
set3	10	562	574	3136.87	1	2.09	10.00
t60	20	400	416	2110.38	4	3.85	20.00
t120	20	800	820	2408.71	0	2.44	0.00
t249	20	1660	1680	2483.16	0	1.19	0.00
t501	20	3340	3360	2724.75	0	0.60	0.00
u120	20	981	981	10.97	20	0.00	100.00
u500	20	4024	4027	505.77	17	0.07	85.00
u1000	20	8011	8016	1464.91	17	0.06	85.00
moyenne/somme	1350				1206	1.05	89.33

5.7.3 Résultats numériques

L'algorithme que nous proposons pour résoudre le $\mathcal{BPP}$-$\mathcal{FO}$ commence par applique la phase de prétraitement, ensuite il calcule les bornes inférieures polynomiales et les heuristiques gloutonnes, puis il déclenche la méthode de génération de colonnes suivi par le VNS. Les résultats obtenus par cet algorithme sont rapportés dans le tableau 5.7. Nous nous concentrons sur 135 instances parmi les 675, en considérant la première instance de chaque groupe de cinq instances ayant la même classe et le même nombre d'objets.

Chaque cellule du tableau donne un résultat moyenné sur neuf instances. Pour le prétraitement, nous donnons le nombre d'objets dont la fragilité a été modifiée (n_{dec}), le pourcentage moyen de modification en terme de fragilité (%dec) et le temps CPU en secondes (sec). Pour les bornes inférieures et supérieures, nous donnons les valeurs moyennes obtenues par les différentes procédures, avec le temps de calcul en secondes CPU et le nombre de solutions optimales obtenues (opt). Pour la génération de colonnes, nous donnons le nombre moyen de colonnes (col) ajoutées au problème maître tout au long des itérations, ainsi que le nombre de coupes duales (cuts) ajoutées initialement au problème maître.

En 15 secondes, le prétraitement décrémente, en moyenne, la fragilité de 10 objets d'un facteur de 2.5%. La borne inférieure obtenue par L_2 est d'une très bonne qualité, les bornes inférieures à base de compatibilité entre objets (L_{cp}) et de fonctions dual-réalisables (L_{dff}) n'ont pas pu faire mieux que L_2. Les heuristiques gloutonnes, exécutées dans la première phase du VNS, obtiennent une borne supérieure moyenne égale à 25.7 bins. Cette phase de l'algorithme est très rapide et efficace, vu qu'elle réussit à atteindre 54 solutions optimales en moins de 0.2 secondes en moyenne.

La génération de colonnes augmente le nombre de solutions optimales obtenues en utilisant 10 secondes de plus en moyenne. La valeur moyenne de la borne inférieure a augmenté de 24.97 à 25.09. Le nombre de coupes duales ajoutées au problème

maître est très grand mais cela permet de réduire sensiblement le nombre d'itérations. Le pricing a été résolu par la méthode de programmation dynamique, qui a été en moyenne 20 fois plus rapide que les modèles mathématiques (5.40)–(5.45) résolus par Cplex. La faible différence entre L_2 et la génération de colonnes expliquent le fait que les bornes basées sur les DFF soient égales à L_2.

Le VNS utilise 138 secondes suplémantaires, en moyenne, pour augmenter le nombre de solutions optimales jusqu'à 78. La borne supérieure moyenne a été réduite de 25.7 à 25.53.

Dans le tableau 5.8 nous comparons l'algorithme global avec le modèle compact (5.7)–(5.12). Le modèle a été résolu par Cplex, avec un temps limite de 10 minutes CPU. Pour chacun des deux algorithmes, nous rapportons la valeur moyenne des bornes supérieures et trouvées, le pourcentage gap évalué comme suit %gap=$100(U - L)/U$, le temps de calcul en secondes CPU et le nombre total de solutions optimales.

Notre algorithme globale est meilleur que l'utilisation des modèles mathématiques, car il trouve 78 solutions optimales parmi 135, contre 51 trouvées par les modèles. De plus, l'effort de calcul est plus faible (163 seconds contre 380) et le pourcentage moyen de gap est réduit quasiment de moitié. Le modèle mathématique est très efficace pour résoudre des instances de petite taille avec 50 objets, mais il est très faible pour les instances de plus grande taille. En revanche, l'algorithme proposé est capable de résoudre un grand nombre d'instances de grande taille.

Les cinq classes partagent le même degré de difficulté, et pour chacune d'elle, nous n'avons pas réussi à résoudre les 27 instances à l'optimalité.

5.8 Conclusions

Nous avons étudié un problème d'optimisation combinatoire difficile à résoudre apparaissant dans le domaine de la télécommunication. Nous avons attaqué ce problème à l'aide de bornes inférieures et supérieures non triviales, impliquant des fonctions dual-réalisables ainsi qu'une recherche locale à voisinage variable et une méthode de génération de colonnes.

Ce travail étant le premier travail qui traite ce problème d'un point de vue numérique, nous avons proposé un jeu d'essai qui s'est avéré difficile à résoudre. Ces instances sont maintenant en ligne pour libre utilisation.

Les résultats obtenus montrent l'efficacité des algorithmes proposés, qui sont capables d'améliorer les résultats obtenus par une formulation compacte du problème. En particulier, parmi 135 instances, 78 solutions optimales ont été obtenues en moins de trois minutes en moyenne, contre 51 solutions optimales obtenues par le modèle compact en un temps de calcul plus élevé. Le bon comportement de la méthode de génération de colonnes nous encourage à développer un algorithme de branch & price afin de générer les solutions optimales des instances restantes.

TABLE 5.7 – *Evaluation des quatre phases de l'algorithme proposé.*

instance		prétraitement			bornes inférieures et supérieures						Génération de colonnes					VNS		
classe	n	n_{dec}	%dec	sec	L_2	L_{cp}	L_{dff}	U	sec	opt	L	col	cuts	sec	opt	U	sec	opt
1	50	9.22	4.57	0.84	12.44	8.00	12.44	12.89	0.05	5	12.67	33	1515	0.09	7	12.78	33.46	8
	100	8.56	1.94	4.35	24.78	12.78	24.78	25.78	0.10	1	25.11	164	16390	2.13	3	25.56	165.90	5
	200	10.67	1.55	39.70	49.67	30.44	49.67	51.22	0.43	1	50.00	351	94709	54.82	1	50.89	265.62	2
2	50	15.00	5.28	1.04	8.44	5.56	8.44	8.56	0.06	8	8.44	8	288	0.05	8	8.56	33.35	8
	100	13.00	3.16	4.88	17.33	10.22	17.33	17.78	0.12	5	17.33	97	6406	0.85	5	17.56	97.93	7
	200	11.33	1.96	31.80	33.44	20.00	33.44	34.00	0.48	4	33.44	267	62857	49.39	4	34.00	169.34	4
3	50	9.00	2.43	0.89	11.00	9.22	11.00	11.33	0.05	6	11.00	31	301	0.05	6	11.33	100.01	6
	100	8.11	1.55	4.21	23.56	17.44	23.56	24.56	0.09	1	23.67	164	3843	0.49	1	24.11	164.17	5
	200	11.89	0.92	42.70	45.67	34.44	45.67	46.89	0.40	2	46.00	308	15423	4.98	2	46.89	236.63	2
4	50	8.67	2.28	1.00	11.00	9.11	11.00	11.33	0.05	6	11.00	28	272	0.05	6	11.22	66.75	7
	100	10.89	1.46	6.33	22.78	16.89	22.78	23.67	0.10	1	22.78	158	2208	0.33	1	23.22	140.76	5
	200	9.00	1.01	40.87	45.11	34.67	45.11	46.56	0.39	0	45.44	406	36193	16.79	0	46.22	265.79	2
5	50	10.00	5.58	0.83	10.00	8.00	10.00	10.11	0.05	8	10.00	11	148	0.04	8	10.00	0.37	9
	100	9.11	2.00	3.99	19.78	13.67	19.78	20.44	0.11	3	19.78	135	5181	0.71	3	20.22	134.99	5
	200	10.00	1.53	39.78	39.56	26.22	39.56	40.33	0.42	3	39.67	287	23461	10.83	3	40.33	202.78	3
moyenne/somme		10.30	2.48	14.88	24.97	17.11	24.97	25.70	0.19	54	25.09	163	17946	9.44	58	25.53	138.52	78

TABLE 5.8 – *Comparaison avec le modèle mathématique compact.*

Instance		Modèle compact					Algorithme global				
classe	n	L	U	%gap	sec	opt	L	U	%gap	sec	opt
1	50	12.67	12.67	0.00	0.45	9	12.67	12.78	1.11	34.43	8
	100	24.78	25.78	3.78	541.79	1	25.11	25.56	1.81	172.49	5
	200	49.56	51.67	4.14	600.10	0	50.00	50.89	1.62	360.57	2
2	50	8.44	8.44	0.00	0.68	9	8.44	8.56	1.11	34.50	8
	100	17.22	17.89	3.50	460.80	3	17.33	17.56	1.35	103.77	7
	200	33.44	34.89	3.91	495.06	2	33.44	34.00	1.46	251.01	4
3	50	11.00	11.22	1.65	136.31	7	11.00	11.33	2.57	101.00	6
	100	23.33	24.44	4.54	600.03	0	23.67	24.11	1.86	168.96	5
	200	45.00	47.33	4.95	600.13	0	46.00	46.89	1.82	284.70	2
4	50	11.00	11.11	0.93	67.41	8	11.00	11.22	1.85	67.85	7
	100	22.56	23.33	3.33	477.07	2	22.78	23.22	1.81	147.51	5
	200	44.33	46.56	4.81	600.10	0	45.44	46.22	1.63	323.83	2
5	50	10.00	10.00	0.00	1.05	9	10.00	10.00	0.00	1.30	9
	100	19.67	20.56	4.35	533.67	1	19.78	20.22	2.09	139.79	5
	200	39.33	41.44	5.11	600.09	0	39.67	40.33	1.56	253.80	3
moyenne/somme		24.82	25.82	3.00	380.98	51	25.09	25.53	1.58	163.03	78

Conclusion générale

Dans cette thèse, nous avons étudié une variété de problèmes de bin packing avec contraintes pratiques liées à des conflits entre articles (conflits simples ou fragilités).

En ce qui concerne les méthodes de résolution approchées, nous avons montré que les méthodes basées sur des stratégies d'oscillation impliquant des phases de construction et de destruction étaient particulièrement bien adaptées aux problèmes de packing avec conflits. La possibilité de violer ce type de contraintes permet d'explorer le voisinage d'une manière très efficace, que ce soit à l'aide d'une recherche tabou ou d'une recherche à voisinage variable.

Nos contributions en rapport avec les fonctions dual-réalisables confirment le fait que ces méthodes peuvent être efficaces lorsque le modèle de génération de colonnes associé permet de calculer de bonnes bornes. Pour le problème de bin-packing avec conflits, les DFF ne donnent pas en elles-mêmes des résultats satisfaisants, puisque nous avons dû utiliser des bornes reposant sur des modèles de transport pour les améliorer. En revanche, pour les objets fragiles, ces fonctions permettent d'obtenir de très bonnes évaluations.

Nous avons aussi décrit comment des algorithmes de résolution de problèmes de bin packing avec conflits pouvaient tirer partie de la structure du graphe de conflits à l'aide du concept de décomposition arborescente. Les résultats que nous avons obtenus sur la variante bi-dimensionnelle du problème de bin packing avec conflits montrent que ces algorithmes peuvent être notablement plus rapides et plus efficaces que les algorithmes classiques lorsque la largeur arborescente du graphe de compatibilité est faible. Ils représentent une nouvelle preuve de l'utilité du concept de décomposition arborescente. La méthode que nous proposons est générique. L'approche de décomposition permet de mettre au point des méthodes hybrides utilisant résolutions exactes et approchées. Elle se prête bien au parallélisme, que ce soit au niveau de la séparation des clusters, ou de la résolution des clusters séparés.

Au cours de nos travaux sur les méthodes de génération de colonnes pour résoudre des problèmes de le bin packing, nous avons testé différentes formulations mathématiques du sous-problème de pricing. Dans le cas du bin-packing bi-objectif, le sous-problème s'est avéré très difficile à résoudre en raison de sa structure quadratique. Pour faire face à cette difficulté, nous avons utilisé des techniques hybrides de génération de colonnes : l'idée est de générer le plus possible de colonnes avec des méthodes heuristiques et méta-heuristiques rapides et par la programmation mathé-

matique si nécessaire. Même avec ces améliorations, le temps nécessaire à la génération des colonnes ne permet pas la mise au point d'une méthode de branch-and-price dans l'immédiat. Une perspective à court terme est de tester de nouvelles modélisations du sous-problème et de mettre au point des hybridations plus avancées de ces méthodes. En revanche, pour le cas des objets fragiles, nous avons réussi à proposer une méthode de programmation dynamique très rapide et beaucoup plus efficace que la programmation mathématique. Par ailleurs, nous avons adapté des techniques classiques de stabilisation de la génération de colonnes, ce qui a largement diminué le nombre d'itérations réalisées.

D'un point de vue général, cette thèse confirme la difficulté des problèmes de packing, surtout lorsqu'on considère des contraintes pratiques. Nous avons montré que cette difficulté pouvait être abordée à l'aide de méthodes de décomposition qui mettent en œuvre programmation mathématique, heuristiques et métaheuristiques. Les prochaines avancées dans ce domaine ne reposent pas uniquement sur l'amélioration de chacun des éléments de cette résolution, mais aussi sur des hybridations plus fines de ceux-ci.

Bibliographie

[1] H. Adorf and M. Johnston. A discrete stochastic neural network algorithm for constraint satisfaction problems. In *Proceedings of the International Joint Conference on Neural Networks, San Diego, CA*, 1990. (Cité page 84.)

[2] A.E. Fernandes Muritiba, M. Iori, E. Malaguti, and P. Toth. Algorithms for the bin packing problem with conflicts. *Informs Journal on Computing, Published online in Articles in Advance, DOI : 10.1287/ijoc.1090.0355*, 2009. (Cité pages ix, x, 2, 27, 28, 29, 31, 33, 34, 35, 41, 42, 51, 52, 53, 54, 55, 56, 57, 58, 59, 84, 86 et 96.)

[3] N. Bansal, Z. Liu, and A. Sankar. Bin-packing with fragile objects. In R. Baeza-Yates, U. Montanari, and N. Santoro, editors, *IFIP conference proceedings*, volume 223, pages 38–46. Kluwer academic, Dordrecht, 2002. (Cité page 36.)

[4] N. Bansal, Z. Liu, and A. Sankar. Bin-packing with fragile objects and frequency allocation in cellular networks. *Wireless Networks*, 15(6) :821–830, 2009. (Cité pages 37, 38, 39, 103 et 109.)

[5] B. Bengtsson. Packing rectangular pieces - a heuristic approach. *The computer journal*, 25 :353–357, 1982. (Cité page 21.)

[6] J. Berkey and P. Wang. Two dimensional finite bin packing algorithms. *Journal of Operational Research*, 38 :423–429, 1987. (Cité pages x, 25, 55, 59, 74 et 76.)

[7] M. Boschetti and A. Mingozzi. The two-dimensional finite bin packing problem. part I : New lower bounds for the oriented case. *4OR*, 1 :27–42, 2003. (Cité pages 19, 20, 22, 23, 43 et 106.)

[8] M. Boschetti and A. Mingozzi. The two-dimensional finite bin packing problem. part II : New lower and upper bounds. *4OR*, 1 :135–147, 2003. (Cité page 26.)

[9] D. Brélaz. New methods to color the vertices of a graph. *Communications of the Association for Computing Machinery*, 22 :251–261, 1979. (Cité pages 9, 10, 11 et 12.)

[10] S. Cahon, N. Melab, and E. Talbi. ParadisEO : A framework for the reusable design of parallel and distributed metaheuristics. *Journal of Heuristics*, 10(3) :357–380, 2004. (Cité pages 68, 73, 91, 93, 96, 115 et 125.)

[11] A. Caprara and P. Toth. Lower bounds and algorithms for the 2-dimensional vector packing problem. *Discrete Applied Mathematics*, 111 :231–262, 2001. (Cité page 30.)

[12] J. Carlier, F. Clautiaux, and A. Moukrim. New reduction procedures and lower bounds for the two-dimensional bin-packing problem with fixed orientation. *Computers & Operations Research*, 34(8) :2223–2250, 2007. (Cité pages 2, 21, 22, 23, 45 et 46.)

[13] A. Ceselli and G. Righini. An optimisation algorithm for the orderded open-end bin-packing problem. *Operations Research*, 56 :423–436, 2008. (Cité pages 38 et 106.)

[14] M. Chams, A. Herz, and D. de Wera. Some experiments with simulated annealing for coloring graphs. *European Journal of Operational Research*, 32 :260–266, 1987. (Cité pages 9, 10 et 11.)

[15] W. Chan, F.-L. Chin, D. Ye, G. Zhang, and Y. Zhang. Online bin packing of fragile objects with application in cellular networks. *Journal of Combinatorial Optimization*, 14 :427–435, 2007. (Cité pages 36, 37, 38, 39 et 103.)

[16] H. Chao, M. Harper, and R. Quong. A tight lower bound for optimal bin packing. *Operational Research Letters*, 18 :133–138, 1995. (Cité page 21.)

[17] B. Chazelle. The bottom-left bin-packing heuristic : an efficient implementation. *IEEE Transactions on Computers*, C-32 :697–707, 1983. (Cité page 25.)

[18] C. Chen, S. Lee, and Q. Shen. An analytical model for the container loading problem. *European Journal of Operational Research*, 80 :68–76, 1995. (Cité page 19.)

[19] N. Christofides. An algorithm for the chromatic number of a graph. *Computer Journal*, 14 :38–39, 1971. (Cité page 9.)

[20] F. Clautiaux, C. Alves, and J. V. de Carvalho. A survey of dual-feasible functions for bin-packing problems. *Annals of Operations Research*, 2008. to appear. (Cité pages 21 et 112.)

[21] F. Clautiaux, J. Carlier, and A. Moukrim. A new exact method for the two-dimensional orthogonal packing problem. *European Journal of Operational Research*, 183(3) :1196–1211, 2007. (Cité page 19.)

[22] F. Clautiaux, A. Jouglet, J. Carlier, and A. Moukrim. A new constraint programming approach for the orthogonal packing problem. *Computers & Operations Research*, 35(3) :944–959, 2008. (Cité page 64.)

[23] E. Coffman, M. Garey, and D. Johnson. Approximation algorithms for bin-packing – an updated survey. *Algorithm design for computer system design. Vienna : Springer*, pages 49–106, 1984. (Cité pages 24, 25, 33, 57 et 65.)

[24] E. Coffman, M. Garey, D. Johnson, and R. Tarjan. Performance bounds for level oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 9 :808–826, 1980. (Cité page 24.)

[25] E. G. Coffman, M. R. Garey, and D. S. Johnson. *Approximation Algorithms NP-Hard Problems*, chapter 2, pages 46–93. D. Hochbaum (ed.), PWS Publishing, Boston, 1996. (Cité page 23.)

[26] E. G. Coffman and P. W. Shor. Average-case analysis of cutting and packing in two dimensions. *European Journal of Operational Research*, 44 :134–144, 1990. (Cité page 24.)

[27] Y. Collette and P. Siarry. *Optimisation multiobjectif*. Eyrolles, 2002. (Cité page 8.)

[28] T. Crainic, G. Perboli, M. Pezzuto, and R. Tadei. New bin packing fast lower bounds. *Computers & Operations Research*, 34 :3439–3457, 2007. (Cité page 21.)

[29] G. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8 :101–111, 1960. (Cité pages 18, 19 et 88.)

[30] W. de la Vega. Random graphs almost optimally colorable in polynomial time. *Annals of Discrete Mathematics*, 1 :43–65, 1985. (Cité page 9.)

[31] D. de Werra. An introduction to timetabling. *European Journal of Operational Research*, 19 :151–162, 1985. (Cité page 9.)

[32] K. Deb. *Multi-objective optimization using evolutionary algorithms*. John Wiley, 2001. (Cité page 8.)

[33] I. Diaz and P. Zabala. A branch-and-cut algorithm for graph coloring. In *Computational Symposium on Graph Coloring ans its Generalization (COLOR02), Ithaca, New York*, 2002. (Cité page 12.)

[34] I. Diaz and P. Zabala. A branch-and-cut algorithm for graph coloring. *Optimization Online : Combinatorial Optimization*, 2002. (Cité page 12.)

[35] R. Dorne and J. Hao. Tabu search for graph coloring, *t*-coloring and set *t*-colorings. In S. Voss, S. Martello, I. Osman, and e. C. Roucairol, editors, *Meta-Heuristics : Advances and Trends in Local Search Paradigms for Optimization*, volume 22 of *Kluwer, Boston*, pages 33–47, 1996. (Cité pages 10 et 12.)

[36] D.S. Johnson and C.R. Aragon and L.A. McGeoh and C. Schevon. Optimization by simulated annealing : an experimental evaluation ; part ii, graph coloring and number partitionning. *Operations Research*, 39 :378–406, 1991. (Cité pages 9, 10 et 11.)

[37] F. Edgeworth. *Mathematical Physics*. P. Keagan, Londres, Angleterre, 1881. (Cité page 7.)

[38] M. Ehrgott. Multicriteria optimization. In Springer, editor, *Lecture Notes in Economics and Mathematical Systems*, volume 491, 2000. (Cité page 8.)

[39] M. Ehrgott and X. Gandibleux. A survey and annotated bibliography of multiobjective combinatorial optimization. *OR Spektrum*, 22 :425–460, 2000. (Cité page 8.)

[40] L. Epstein and A. Levin. On bin packing with conflicts. In *4th Workshop on Approximation and online Algorithms (WAOA2006)*, pages 160–173, 2006. (Cité page 32.)

[41] L. Epstein, A. Levin, and R. van Stee. Multi-dimensional packing with conflicts. *Acta Informatica*, 45 :155–175, 2008. (Cité page 32.)

[42] R. D. et J. Pearl. Tree clustering schemes for constraint processing. *Artificial Intelligence*, 38 :353–366, 1989. (Cité page 16.)

[43] S. Fekete and J. Schepers. New classes of fast lower bounds for bin packing problems. *Mathematical Programming*, 91 :11–31, 2001. (Cité page 21.)

[44] S. Fitzpatrick and L. Meertens. An experimental assessment of a stochastic, anytime, decentralized, soft colourer for sparse graphs. In *Proceedings of SAGA 2001, 1st Symposium on Stochastic Algorithms, Foundations and Applications, Berlin, Springer-Verlag.*, 2001. (Cité page 12.)

[45] S. Fitzpatrick and L. Meertens. Soft, real-time, distributed graph coloring using decentralized, synchronous, stochastic, iterative-repair, anytime algorithms. Technical report, Kerstel Institute, 2001. KES.U.01.05. (Cité page 12.)

[46] K. Fleszar and K. Hindi. New heuristics for one-dimensional bin-packing. *Computers & Operations Research*, 29 :821–839, 2002. (Cité page 114.)

[47] C. Fleurent and J. Ferland. Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research*, 36 :437–464, 1996. (Cité pages 9, 10 et 12.)

[48] C. Friden and D. de Werra. Tabaris : an exact algorithm based on tabu search for finding a maximum independent set in a graph. *Computers & Operations Research*, 17 :437–445, 1990. (Cité page 9.)

[49] D. Fulkerson and O. Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15 :835–855, 1965. (Cité page 15.)

[50] A. Gamst. Some lower bounds for a class of frequency assignment problems. *IEEE Transactions on Vehicular Technology*, 35 :8–14, 1986. (Cité page 9.)

[51] M. Garey and D. Johnson. *Computers and Intractability, a guide to the theory of NP-completeness*. Freeman, New York, 1979. (Cité pages 5, 10, 16 et 65.)

[52] M. Garey, D. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1 :237–267, 1979. (Cité page 9.)

[53] M. Gendreau, G. Laporte, and F. Semet. Heuristics and lower bounds for the bin packing problem with conflicts. *Computers & Operations Research*, 31 :347–358, 2004. (Cité pages ix, 27, 28, 32, 51, 52, 53, 54, 66 et 84.)

[54] P. Gilmore and R. Gomory. A linear programming approach to the cutting stock problem. *Operations Research*, 9 :849–859, 1961. (Cité pages 18, 19, 31 et 106.)

[55] P. Gilmore and R. Gomory. A linear programming approach to the cutting stock problem - part II. *Operations Research*, 11 :863–888, 1963. (Cité pages 18, 19, 31 et 106.)

[56] P. Gilmore and R. Gomory. Multistage cutting stock problems of two and more dimensions. *Operations Research*, 13 :94–120, 1965. (Cité page 19.)

[57] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1998. (Cité page 68.)

[58] A. Hammami. Modélisation technico-economique d'une chaîne logistique dans une entreprise réseau, 2003. PhD thesis, Université de Laval, Québec, Canada. (Cité page 8.)

[59] P. Hansen, N. Mladenović, and J. M. Pérez. Variable neighbourhood search : methods and applications. *Annals of Operations Research*, 175 :367–407, 2010. (Cité page 114.)

[60] J. E. Hayek, A. Moukrim, and S. Negre. New resolution algorithm and pretreatments for the two-dimensional bin-packing problem. *Computers & Operations Research*, 35(10) :3184 – 3201, 2008. (Cité page 26.)

[61] A. Herz and D. de Wera. Using tabu search techniques for graph coloring. *Computing*, 39 :345–351, 1987. (Cité pages 9, 10, 11 et 12.)

[62] M. Hifi and M. Michrafy. Reduction strategies and exact algorithms for the disjunctively constrained knapsack problem. *Computers & Operations Research*, 34(9) :2657 – 2673, 2007. (Cité page 32.)

[63] K. Jansen. An approximation scheme for bin packing with conflicts. *Journal of Combinatorial Optimization*, 3 :363–377, 1999. (Cité page 32.)

[64] K. Jansen and S. Öhring. Approximation algorithms for time constrained scheduling. *Information and Computation*, 132(2) :85–108, 1997. (Cité pages 27, 32 et 33.)

[65] B. Jarboui, S. Ibrahim, and A. Rebai. A new destructive bounding scheme for the bin packing problem. *Annals of Operations Research*, 2008. (Cité page 21.)

[66] P. Jégou, S. Ndiaye, and C. Terrioux. Computing and exploiting tree-decompositions for solving constraint networks. In *Proceedings of the Eleventh International Conference on Principles and Practice of Constraint Programming (CP-2005)*, pages 777–781, 2005. (Cité page 67.)

[67] J.L. González-Verlade and M. Laguna. Tabu search with simple ejection chains for coloring graphs. *Annals of Operations Research*, 36 :437–464, 1996. (Cité pages 10 et 12.)

[68] D. Johnson. Near optimal bin packing algorithms, 1973. Ph.D. Dissertation, Massachusetts Institute of Technology, Cambridge, Massachusetts. (Cité pages 21 et 24.)

[69] D. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Science*, 9 :256–278, 1974. (Cité pages 34 et 117.)

[70] L. Kantorovich. Mathematical methods of organizing and planning production. *Management Science*, 6 :363–422, 1960. (Cité page 18.)

[71] A. Khanafer, F. Clautiaux, and E. Talbi. New lower bounds for bin packing problems with conflicts. In *EURO Special Interest Group on Cutting and Packing*, Valencia, Spain, March 2009. (Cité page 41.)

[72] A. Khanafer, F. Clautiaux, and E. Talbi. Problèmes de bin packing avec conflits - nouvelles bornes inférieures. In *ROADEF'09 (dixième congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision)*, Nancy, France, February 2009. (Cité page 41.)

[73] A. Khanafer, F. Clautiaux, and E. Talbi. Tree-decomposition based tabu search for the bin packing problems with conflicts. *Metaheuristics International Conference, MIC09, Hamburg, Germany*, 2009. (Cité page 61.)

[74] A. Khanafer, F. Clautiaux, and E. Talbi. New lower bounds for bin packing problems with conflicts. *European Journal of Operational Research*, 206 :281–288, 2010. (Cité page 41.)

[75] A. Khanafer, F. Clautiaux, and E. Talbi. Recherche tabou à base de décomposition arborescente pour le problème de bin packing avec conflits. In *ROADEF'10 (onzième congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision)*, Toulouse, France, 24-26 Feb 2010. (Cité page 61.)

[76] A. Koster, H. Bodlaender, and S. van Hoesel. Treewidth : Computational experiments. *Fundamenta Informaticae*, 49 :301–312, 2001. (Cité page 16.)

[77] M. Kubale and B. Jackowski. A generalized implicit enumeration algorithm for graph coloring. *Communications of the Association for Computing Machinery*, 28 :412–418, 1985. (Cité page 12.)

[78] F. Leighton. A new graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau Standard*, 84 :489–505, 1979. (Cité pages 9, 10 et 11.)

[79] J. Leung, M. Dror, and G. Young. A note on an open-end bin packing problem. *Journal of Scheduling*, 4 :201–207, 2001. (Cité page 38.)

[80] D. Liu, K. Tan, S. Huang, C. Goh, and W. Ho. On solving multiobjective bin packing problems using evolutionary particle swarm optimization. *European Journal of Operational Research*, 190(2) :357–382, 2008. (Cité page 26.)

[81] A. Lodi, S. Martello, and D. Vigo. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. *Informs Journal on Computing*, 11 :345–357, 1999. (Cité page 25.)

[82] C. Lucet, F. Mendes, and A. Moukrim. An exact method for graph coloring. *Computers & Operations Research*, 33 :2189–2207, 2006. (Cité pages 12 et 61.)

[83] G. S. Lueker. Bin packing with items uniformly distributed over intervals [a,b]. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science (FOCS 83)*, pages 289–297. IEEE Computer Society, 1983. (Cité page 21.)

[84] E. Malaguti, M. Monaci, and P. Toth. A metaheuristic approach for the vertex coloring problem. *Informs Journal on Computing*, 20 :302–316, 2008. (Cité page 31.)

[85] S. Martello and P. Toth. *Knapsack Problems : Algorithms and Computer Implementations*. Wiley, Chichester, 1990. (Cité page 18.)

[86] S. Martello and D. Vigo. Exact solution of the two-dimensional finite bin packing problem. *Management Science*, 44 :388–399, 1998. (Cité pages x, 19, 21, 55, 59, 74 et 76.)

[87] B. McCloskey and A. Shankar. Approaches to bin packing with clique-graph conflicts. Technical report, 2005. (Cité page 32.)

[88] A. Mehrotra and M. Trick. A column generation approach for graph coloring. *Informs Journal on Computing*, 8 :344–354, 1996. (Cité pages 12 et 31.)

[89] F. Mendes. *Méthode de décomposition pour la coloration de graphes*. PhD thesis, Université de Picardie Jules Verne, 2005. (Cité page 10.)

[90] K. Miettinen. *Nonlinear multiobjective optimization*. Kluwer Academic Publishers, 1999. (Cité page 8.)

[91] S. Minton, M. Johnston, and P. Laird. Solving large scale constraint satisfaction and scheduling problems using a heuristic repair method. In *AAAI'90, Boston, MA*, 1990. (Cité page 84.)

[92] N. Mladenovic and P. Hansen. Variable neighbourhood search. *Computers & Operations Research*, 24 :1097–1100, 1997. (Cité page 114.)

[93] M. Monaci and P. Toth. A set-covering-based heuristic approach for bin-packing problems. *Informs Journal on Computing*, 18 :71–85, 2006. (Cité page 31.)

[94] M. Mongeau and C. Bes. Optimization of aircraft container loading. *IEEE Transactions on Aerospace and Electronic Systems*, 39 :140–150, 2003. (Cité page 79.)

[95] C. Morgenstern. Distributed ccoloration neighbourhood search. In e. Johnson DS, Trick MA, editor, *2nd DIMACS implementation challenge*, volume 22 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society*, pages 335–358, 1996. (Cité pages 10, 11 et 35.)

[96] Y. Oh and S. Son. On a constrained bin-packing problem. Technical report, 1995. (Cité page 32.)

[97] H. Onodera, Y. Taniguchi, and K. Tmaru. Branch-and-bound placement for building block layout. In *28th ACM/IEEE Design Automation Conference*, pages 433–439, 1991. (Cité page 19.)

[98] V. Pareto. *Cours d'économie politique*. Rouge, Lausanne, Suisse, 1986. (Cité page 7.)

[99] U. Pferschy and J. Schauer. The knapsack problem with conflict graphs. Technical report, University of Graz, Department of Sctatistics and Operations Research, 2008. (Cité page 32.)

[100] D. Pisinger and M. Sigurd. Using decomposition techniques and constraint programming for solving the two-dimensional bin packing problem. *Informs Journal on Computing*, 19 :36–51, 2007. (Cité page 64.)

[101] N. Robertson and P. Seymour. Graph minors. ii algorithmic aspects of tree-width. *Journal of Algorithms*, 7 :309–322, 1986. (Cité page 13.)

[102] R. Sadykov and F. Vanderbeck. Branch-and-price for bin packing with conflicts, 2009. ISMP. (Cité pages 32 et 35.)

[103] M. Sathe, O. Schenk, and H. Burkhart. Solving bi-objective many-constraint bin packing problems in automobile sheet metal forming processes. In S. Voss, S. Martello, I. Osman, and e. C. Roucairol, editors, *Proceedings of the 5th International Conference on Evolutionary Multi-Criterion Optimization, Nantes, France, April 2009*, volume 22 of *Lecture Notes in Computer Science, Springer, 5467*, pages 246–261, 2009. (Cité page 26.)

[104] A. Scholl, R. Klein, and C. Jürgens. Bison : a fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers & Operations Research*, 24(7) :627–645, 1997. (Cité page 125.)

[105] B. Selman, H. Levesque, and M. Mitchell. A new method for solving hard satisfiability problems. In *AAAI'92, San Diego, CA*, pages 440–446, 1992. (Cité page 85.)

[106] E. Sewell. An improved algorithm for exact graph coloring. In D. Johnson and M. Trick, editors, *Cliques, coloring and satisfiability*, Second DIMACS Implementation Challenge. American Mathematical Society, pages 359–373, 1993. (Cité page 12.)

[107] P. Soriano and M. Gendreau. Tabu search algorithms for the maximum clique problem. In D. Johnson and M. Trick, editors, *Cliques, coloring and satisfiability*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 221–242, 1996. (Cité page 54.)

[108] S.V. Amiouny and J.J. Bartholdi and J.H. Vande Vate and J.X. Zhang. Balanced loading. *Operations Research*, 40 :238–246, 1992. (Cité page 26.)

[109] E.-G. Talbi. *Metaheuristics from design to implementation*. John Wiley & Sons, New Jersey, 2009. (Cité pages ix, 115 et 116.)

[110] R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. In *SIAM Journal on Computing*, volume 13, pages 566–579, 1984. (Cité pages 15, 16 et 49.)

[111] N. Touati, L. Létocart, and A. Nagih. Sur l'accélération de la convergence de la génération de colonnes. Technical report, Laboratoire d'informatique de Paris-nord - LIPN - CNRS : UMR7030 - Université Paris-Nord - Paris XIII - Laboratoire d'Informatique Théorique et Appliquée - LITA - Université Paul Verlaine - Metz, 2006. (Cité page 123.)

[112] E. Ulungu and J. Teghem. Multi-objective combinatorial optimization : a survey. *Journal of Multi-Criteria Decision Analysis*, 3 :83–104, 1994. (Cité page 8.)

[113] J. Valério de Carvalho. Using extra dual cuts to accelerate column generation. *Informs Journal on Computing*, 17(2) :175–182, 2005. (Cité pages 104, 123, 124 et 125.)

[114] F. Vanderbeck. Computational study of a column generation algorithm for bin packing and cutting stock problems. *Mathematical Programming*, 86 :565–594, 1999. (Cité page 31.)

[115] M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM Journal on Algebraic and Discrete Methods*, 2 :77–79, 1981. (Cité page 14.)

Titre Algorithmes pour des problèmes de bin packing mono- et multi-objectif

Résumé Le problème de bin packing constitue un problème très général qui permet de représenter un grand nombre de problèmes combinatoires classiques et d'applications pratiques. Notre étude porte sur la résultion d'une variété de problèmes de bin packing.

Mots-clés bin packing, bin packing avec conflits, bin packing avec objets fragiles, bin packing bi-objectif, coloration de graphes, décomposition arborescente, bornes inférieures, heuristiques et méta-heuristiques, génération de colonnes.

Title Algorithms for mono- and multi-objective bin packing problems

Abstract The bin-packing problem is a general problem, which is able to model a large number of combinatorial problems. Our study is about some variants on the bin-packing problem.

Keywords bin packing, bin packing with conflicts, bin packing with fragiles objects, bi-objective bin packing, graph coloring, tree-decomposition, lower bounds, heuristics and meta-heuristics, column generation.