N° d'ordre : 12 / 2005 –M / MT

République Algérienne Démocratique Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari Boumediene

Faculté des Mathématiques

Mémoire Présenté pour l'obtention du diplôme de

M A G I S T E R

EN : MATHEMATIQUES

Spécialité : **Recherche Opérationnelle (Mathématiques de Gestion)**

Par

Nazahet FELLAH

Le problème de placement unidimensionnel

Soutenue le : 16 / 05/ 2005, Devant le jury suivant :

- Mr A. KHELLADI	Professeur	USTHB	Président de jury.
- Mr M. AÏDER	Professeur	USTHB	Directeur de thèse.
- Mr M. MOULAI	Maître de Conférence	USTHB	Examinateur.
- Mr M. BOUDHAR	Maître de Conférence	USTHB	Examinateur.
- Mr R. OUAFI	Maître de conférences	USTHB	Examinateur.
- Mme. S.BENABDERAHMANE	Maître Assistante	USTHB	Invitée.

Remerciements

En premier lieu, je remercie Dieu le tout puissant de m'avoir prêté main pour la réalisation de ce humble travail.

A Mr M. Aïder, professeur à la faculté des mathématiques USTHB, J'exprime toute ma gratitude et ma reconnaissance pour la patience et la générosité avec lesquelles il a su me guider durant les travaux de recherches. Je le remercie vivement pour son aide, son assistance et sa sympathie dont il a fait preuve.

Je remercie également Mr A. Khelladi, Professeur à la faculté des mathématiques USTHB, qui a bien voulu me rendre honneur en acceptant de prendre en charge la présidence du jury.

Mes sincères remerciements vont aussi à Mr M. Moulai, Mr M. Boudhar,
Mr R Ouafi, maîtres de conférence et Mme S Benabderahmane, maître assistante pour leur gratitude et leur dévouement qui ont permis d'honorer le jury.

Ainsi Mr M.Daoudi chargé de cours à l'institut d'informatique de l'USTHB trouve ici toute ma reconnaissance pour le soutien dont il m'avait apporté le long de mes études en post-graduation.

Dédicaces

A ceux qui m'ont donnée la vie et que je leur témoigne mon amour et ma reconnaissance :

MAMAN et PAPA,

A mes chers frères :

RAFIK,

ABDERREZAK,

ATHMANE,

A toutes les personnes qui m'ont connue et qui ont cru en moi et à tout ceux qui m'ont soutenue de prés ou de loin,

Je dédie ce modeste travail.

Résumé

Le problème de placement unidimensionnel occupe une place très importante dans la classe des problèmes $\mathcal{NP}$-complets qui constitue le noyau dur des problèmes de l'optimisation combinatoire. Une forte raison qui le rendait un centre d'attraction de plusieurs chercheurs et par nous-mêmes que nous le traitons dans cette thèse.

Le problème de placement s'énonce d'une manière très simple : il s'agit de ranger un ensemble de pièces donné sous forme d'une liste $L = \{i : 1 \leq i \leq n\}$ chacune de taille $s(i)$;
$i = 1, \ldots, n$ dans un nombre minimum de boites toutes de taille 1.

Dans le cadre d'une variation de ce problème nous présentons, en premier lieu, le *problème de placement de taille uniforme sous la contrainte LIB* : *Largest Items at the Bottom* qui a été introduite par Manyem. Ensuite nous traitons *le problème de placement de taille variable sous la contrainte Largest Items at the Bottom* et ceci en analysant les résultats de l'algorithme *Next Fit using Largest bins only* et enfin, nous développons un nouvel algorithme online dénommé *Improved Best Fit* .

En second lieu, nous traitons une généralisation de ce problème introduite par nous-mêmes, appelée le *problème de placement avec graphe de conflits en tolérant un seul conflit par boite*. Nous proposons un algorithme de résolution pour ce problème qui est un hybride des méthodes appropriées à la théorie des graphes et les heuristiques classiques du problème de placement.

TABLE DE MATIERES

Résumé

Introduction générale 1

1. **Généralités sur l'optimisation combinatoire** 4

1.1. Introduction.... 4
1.2. Problématique de l'optimisation combinatoire.....5
1.3. Notions sur la complexité des algorithmes.....6
1.3.1. Comment mesurer l'efficacité ?.....6
1.3.2. Bons et mauvais algorithmes.....7
1.4. Problèmes d'optimisation combinatoire.....8
1.4.1. Problèmes combinatoires faciles et difficiles.....8
1.4.2. Exemples de problème faciles et difficiles 9
1.5. Notions de théorie de la complexité.....10
1.5.1. La théorie de la complexité.....10
1.5.2. Les classes de problèmes *P* et *NP*.....11
1.5.3. Les problèmes *NP*- complets.....12
1.5.4. La conjecture $P \neq NP$.....13
1.6. Les heuristiques.....14
1.6.1. Evaluation des heuristiques.....15
1.6.1.1. Evaluation a priori (performance relative au pire).....16
1.6.1.2. Evaluation a posteriori.....16
1.6.2. Les types d'heuristiques.....17
1.6.2.1. Heuristiques gloutonnes « greedy ».....17
1.6.2.2. Méthodes par recherches locales.....18
1.6.2.3. Méthode de recherche globale (méta-heuristiques).....19

2. Le problème de placement **23**

2.1. Introduction.....23

2.2. Présentation du problème de placement 24
2.3. Formulations du problème de placement 24
2.3.1. Le problème de placement et la programmation mathématique....24
2.3.2. Le problème de placement et la programmation linéaire....26
2.3.3. Le problème de placement et les hypergraphes....27
2.3.4. Le problème de placement et les graphes 28
2.4. La complexité du problème de placement....29
2.5. Les différentes variantes du problème de placement 31
2.5.1. Problème de placement unidimensionnel....32
2.5.2. Problème de placement bidimensionnel....32
2.5.3. Problème de placement tridimensionnel....33
2.5.4. Placement des rectangles en maximisant les profits....34
2.5.5. Problème de placement des lots de pièces....35
2.5.6. Problème de placement avec conflits 35
2.6. Conclusion....35

3. Résolution du problème de placement 36

3.1. Introduction. 36
3.2. Les méthodes exactes....37
3.2.1. Les méthodes arborescentes....37
3.2.1.1. Principes généraux....37
3.2.1.2. Les méthodes arborescentes et le problème de placement....38
3.3. Les méthodes approchées....41
3.3.1. Algorithmes online....43
3.3.2. Algorithmes off-line....48
3.4. Conclusion....49

4. Le problème de placement sous la contrainte *LIB* : version online 50

4.1. Introduction....50
4.2. Présentation du problème....51
4.3. *NP*-complétude du *PPLIB*....51

4.4. Le *PPLIB* de taille uniforme : version online......52
4.4.1. Approches de résolution......53
4.4.1.1. *Next Fit*......53
4.4.1.2. *First Fit*......55
4.5. Le *PPLIB* de taille variable : version online......56
4.5.1. Définition du problème......57
4.5.2. Approches de résolution......58
4.5.2.1. *Next Fit using largest bins only*......58
4.5.2.2. *Improved Best Fit*......61
4.5.2.3. Ratio de performance de *Improved Best Fit*......62
4.6. Conclusions et perspectives......64

5. Le **problème de placement avec graphe de conflits ayant un seul conflit par boite** **65**

5.1. Introduction......65
5.2. Présentation du problème (*PPGC)*......66
5.3. *NP*-complétude du problème......66
5.4. Approches de résolution du (PPGC)......67
5.4.1. Méthodes d'approximation classiques du problème de placement......67
5.4.2. Méthode de coloration 68
5.4.3. Méthode de précoloration......69
5.5. Présentation du problème (*PPGC)* en tolérant un seul conflit par boite......71
5.6. *NP*-complétude du *PPGC avec un seul conflit*......71
5.7. Approches de résolution du (PPGC) avec un seul conflit......71
5.7. 1. Méthodes classiques du problème de placement......72
5.7. 2. Méthode de couplage maximal et précoloration......74
5.8. Conclusion et perspectives......78

Conclusion générale 79
Bibliographie 81

Introduction générale

La Recherche Opérationnelle est une technique récente, datant au plus de la seconde guerre mondiale dont elle a été utilisée dans le domaine de la stratégie militaire ensuite elle a été développée dans des domaines différents autres qu'industriel.

L'idée à retenir c'est que *la Recherche Opérationnelle* ne s'occupe pas des problèmes dans lesquels une solution de bon sens intervient tout naturellement. Son domaine réservé est celui des situations dans lesquelles, pour une raison quelconque, le sens commun se révèle faible ou impuissant.

Tels sont :

a) les problèmes combinatoires
b) les domaines de l'aléatoire
c) les situations de concurrence.

Il est bien connu que l'homme envisage difficilement la multiplicité des combinaisons qui se présentent, dans les moindres faits de la vie, lorsque plusieurs variables peuvent prendre, chacune, des états différents.

Dans tous les problèmes fortement combinatoires, malheureusement, l'esprit humain ne peut envisager le nombre astronomique des arrangements, permutations, combinaisons.
Il lui faut un fil d'*Ariane* s'il veut choisir, entre telle ou telle de ces dispositions, relativement à un critère quelconque, car, même avec une puissante machine, le principe fondamental en matière combinatoire demeure d'éviter toute énumération.

Les problèmes *NP-complets* occupent une place très privilégiée en Recherche Opérationnelle. Leur importance se justifie, d'une part par la grande difficulté à les résoudre (on ne possède pas à ce jour de solution algorithmique complète et efficace pour des instances de grande taille), d'autre part par leurs nombreuses applications pratiques, (optimisation des réseaux, les problèmes de placement d'objets, l'ordonnancement, etc....)

Cette thèse porte sur l'étude du problème de placement unidimensionnel. L'accent sera mis sur l'aspect algorithmique. Ce document est divisé en cinq grandes parties : une partie introductive au contexte théorique dans lequel fait parti le problème, deux parties sur la présentation du problème de placement et les différentes approches optées pour son résolution et deux parties qui représentent notre contribution dans l'étude de deux extensions de ce problème et ceci à travers les analyses des résultats sur les comportements des différents algorithmes qui permettront de conclure sur l'efficacité de chaque algorithme .

Le premier chapitre présente le contexte théorique dans lequel s'inscrit le problème du placement. L'*optimisation combinatoire* qui fait en effet, l'objet de ce chapitre est abordée d'une manière explicite. Nous décrivons la problématique de l'optimisation combinatoire sous ses différentes facettes comme la complexité des algorithmes, la théorie de la complexité des problèmes ainsi que leur classification.

Dans le second chapitre, la problématique du problème de placement est présentée sous ses différentes formulations. Nous parlons ensuite de sa NP-complétude et exhibons ses différentes variantes ainsi que quelques extensions.

Le troisième chapitre est consacré aux méthodes de résolution du problème de placement. Différents algorithmes, sont exposés qu'ils soient exacts ou approchés. Pour ces deux approches, de très nombreuses variantes ont été développées. Nous nous contentons d'en çiter quelques unes.

Après avoir décrit le problème de placement d'une manière assez suffisante, nous présentons à partir du quatrième chapitre notre contribution personnelle.

Le chapitre quatre traite d'une variante du problème de placement appelée « *problème de placement sous la contrainte LIB* » (*Largest Items at the Bottom*) qui a été introduite par Manyem.

Nous présentons les travaux de Manyem, concernant l'analyse des deux algorithmes usuels : *Next Fit* et *First Fit* pour la résolution du problème de placement sous la contrainte *LIB* de taille uniforme. Nous évoquons ensuite une des perspectives proposées par Manyem : *le problème de placement de taille variable sous la contrainte Largest Items at the Bottom*, en étudiant le comportement de l'algorithme *Next Fit using Largest bins only* (*NFL*). Nous développons enfin un nouvel algorithme online dénommé *Improved Best Fit*.

Dans le cinquième et dernier chapitre, nous nous sommes intéressés à une autre extension du problème de placement nommée « *problème de placement avec graphe de conflits* » (*PPGC*) qui a été étudié pour la première fois par K. Jansen et S. Öhring. Nous énonçons le problème (*PPGC*) et exposer les différentes approches de résolution établies par les deux auteurs.

Nous traitons une généralisation de ce problème où on autorise l'existence d'un seul conflit par boite appelée « *problème de placement avec graphe de conflits en tolérant un seul conflit par boite* ». Nous proposons un algorithme pour résoudre ce problème et qui dépend de la recherche d'un couplage maximal dans le graphe de conflits initial, d'une coloration minimale du graphe contracté et l'application d'un algorithme approché du problème de placement sur chaque ensemble de pièces ayant la même couleur.

Chapitre 1

Généralités sur l'optimisation combinatoire

1.1 Introduction

Notre but à travers ce chapitre, est de fournir au lecteur non averti les différentes notions de base de *l'optimisation combinatoire*, dont fait partie notre thème de recherche en lui offrant le bagage nécessaire à la compréhension de notre travail.

Nous décrivons la problématique de *l'optimisation* combinatoire, en abordant ses différentes facettes : *la complexité des algorithmes*, *la théorie de la complexité* des problèmes, les problèmes complets et les problèmes $\mathcal{NP}$- *complets.*

Nous terminons en présentant des méthodes générales de résolution des problèmes $\mathcal{NP}$-*complets*, exactes et approchées.

1.2 Problématique de l'optimisation combinatoire

L'optimisation combinatoire est le noyau fédérateur autour duquel s'est développé l'essentiel de l'activité de la Recherche Opérationnelle : elle se situe au carrefour de la Théorie des graphes, de la Programmation mathématique et de l'Informatique théorique (algorithmique et théorie de la complexité).

Si on décrypte les deux termes de « *optimisation combinatoire* », on en tirera que le premier désigne la détermination d'un optimum d'une fonction qui sert à modéliser un choix optimal, et le second représente la discipline des mathématiques concernée par les structures " discrètes " ou " finies ". Citons quelques branches de cette discipline : la théorie des graphes, la combinatoire énumérative.

Donc, L' " *Optimisation combinatoire* " consiste à :

Trouver le meilleur élément parmi un nombre fini de choix.

Autrement dit, minimiser une fonction, avec contraintes, sur un ensemble fini.

Définition 1.1 Une *instance* appelée parfois (*cas, donnée,…)* d'un problème d'optimisation combinatoire est défini par la donnée d'un ensemble fini S et d'une application $f : S \to IR$.
Il s'agit de déterminer un élément s^* dans S, tel que $f(s^*) \leq f(s)$, pour tout élément s de S.
(problème de minimisation).

Définition 1.2 un problème d'*optimisation combinatoire* est composé de l'ensemble de toutes les instances de ce problème.

Définition 1.3 On associe à toute instance I d'un problème (P), un nombre $\mathcal{T}(I)$ qui mesure la longueur des données de cette instance, appelé la *taille de l'instance I.* Elle représente la quantité de mémoire pour la stocker.

1.3 Notions sur la complexité des algorithmes

1.3.1 Comment mesurer l'efficacité ?

La première idée qui vient à l'esprit pour évaluer et comparer des algorithmes consiste à les programmer, puis mesurer leurs durées d'exécution. En fait, le temps de calcul est une mesure imparfaite car elle dépend trop de la machine, du langage de programmation et des données.

On préfère en pratique compter le *nombre d'opérations élémentaires* de l'algorithme à évaluer : ce nombre ne dépend ni de la machine, ni du langage, et peut s'évaluer sur le papier. On l'appelle *complexité de l'algorithme*. Il s'agit d'un terme technique n'ayant rien à voir avec la complexité de la structure ou la difficulté de la programmation !

Définition 1.4 La complexité d'un algorithme $\mathcal{A}$ de résolution d'un problème d'optimisation combinatoire (P) est une fonction $C_{(A,I)}(N)$, donnant le nombre minimum d'opérations nécessaires à $\mathcal{A}$ pour résoudre la pire des instances de (P) de taille N.

1.3.2 Bons et mauvais algorithmes

Une classification grossière mais reconnue distingue les algorithmes *polynomiaux* (bons) des autres dits *exponentiels* (mauvais). Les fondateurs de la théorie de la complexité Stephen Cook [6] et Richard Karp [33] reconnaissent à Jack Edmonds [7] la première idée de la caractérisation des algorithmes polynomiaux.

Définition 1.5 Un algorithme $\mathcal{A}$ de résolution d'un problème (P) est dit *polynomial* si pour toute instance *I* de taille *N* de ce problème, il existe un polynôme *Q* tel que : $C_{(A,I)}(N) \leq Q(N)$. Un algorithme est considéré comme *efficace* ou *bon* si, et seulement si, il est polynomial.

Des exemples de complexité polynomiale sont $log\ n, n^{0.5},\ n\ log\ n,\ n^2 \ldots$ Une complexité exponentielle peut être une vrai exponentielle au sens mathématique (e^n, 2^n), mais aussi des fonctions comme $n^{\log n}$, la fonction factorielle *n !*, et n^n qui ne peuvent être majorées par un polynôme.

Pour résoudre une instance d'un problème d'optimisation combinatoire, il faut se méfier des méthodes énumératives conduisant à examiner tous les cas possibles. Elles conduisent à des algorithmes exponentiels.

Exemple. L'énumération des parties d'un ensemble de n éléments est de complexité 2^n.

1.4 Problèmes d'optimisation combinatoire

Définition 1.6 *Un problème de reconnaissance* (PR) consiste à chercher dans un ensemble fini S s'il existe un élément s vérifiant une certaine propriété P. Il peut être formulé par un énoncé et une question à réponse Oui- Non.

Il existe toujours un problème de reconnaissance associé à tout problème d'optimisation combinatoire

$$\begin{cases} Min \;\; f(s) \\ s \in S \end{cases}$$

qui est :

Pour un nombre entier k, existe-t-il un élément $\hat{s} \in S$ *tel que* $f(\hat{s}) \leq k$ *?*

Remarque 1.1 Si le problème de reconnaissance associé à un problème d'optimisation combinatoire est difficile, ce dernier l'est alors au moins autant. Car si on disposait d'un algorithme efficace pour le problème de reconnaissance, on pourrait l'utiliser pour résoudre le problème d'optimisation combinatoire, par dichotomie sur *k*.

1.4.1 Problèmes combinatoires faciles et difficiles

Les problèmes d'optimisation combinatoire posent un véritable défi à l'humanité. Les techniques de l'analyse mathématique sont en effet inopérantes pour minimiser une fonction sur un ensemble discret (fini).

Théoriquement, tout problème d'optimisation combinatoire ou problème de reconnaissance peut toujours être résolu par énumération complète : on construit toutes les solutions de S pour retenir la meilleure. Cette méthode se heurte au cardinal de S, qui est le plus souvent

énorme. L'énumération complète donne donc en général des algorithmes exponentiels, inutilisables en pratique, sauf sur les très petits cas.

On connaît pour certains problèmes des algorithmes efficaces (problèmes faciles). Malheureusement, pour d'autres problèmes (difficiles), on n'a pas réussi à trouver d'algorithmes polynomiaux . On ne dispose pour les résoudre que d'algorithmes exponentiels qui sont des sortes d'énumération complète.

1.4.2 Exemples de problèmes faciles et difficiles

Cette section donne des exemples classiques de problèmes avec ou sans algorithmes polynomiaux. Parmi les problèmes faciles, on en trouve ceux de :

- *Chemin de coût minimal.*
  - *Données* : $G = (X, U, C)$, un graphe orienté évalué, et deus sommets s et t de X.
  - *But* : Trouver un chemin de coût minimal de s à t.

- *Flot maximal.*
  - *Données* : Un réseau de transport $G = (X, U, C, s, t)$. Il s'agit d'un graphe valué où le poids C_{ij} sur l'arc (i, j) désigne une capacité entière, le sommet s est appelé *source* et t le *puits.*
  - *But* : Maximiser le débit du flot qui peut s'écouler dans le réseau entre s et t. Le flot φ_{ij} sur un arc (i, j) doit être compris entre 0 et la capacité C_{ij} de l'arc. En chaque nœud, le flot doit être conservé : la somme des flots entrants doit être égale à celle des flots sortants (*lois de Kirchhoff*).

Parmi les problèmes difficiles, on a :

- *Stable maximal.*

  Soit un graphe simple $G = (X, E)$. Un sous-ensemble de sommets S est un ensemble *stable* s'il n'y a pas d'arêtes entre deux sommets quelconques de S.

  Le but est de trouver un stable de G de cardinal maximal.

- *Clique maximale.*

  Dans un graphe simple $G = (X, E)$. Un sous-ensemble de sommets Q est une clique si toute paire de sommets de Q est reliée par une arête.

  Le problème de la clique maximale consiste à trouver une clique de G ayant le Maximum de sommets.

1.5 Notions de théorie de la complexité

1.5.1 La théorie de la complexité

Certains problèmes d'optimisation combinatoire disposent depuis longtemps d'algorithmes polynomiaux, tandis que d'autres n'en ont toujours pas.

Existe-t-il réellement une classe de problèmes combinatoires pour lesquelles on ne trouvera jamais d'algorithmes polynomiaux, ou est- ce que les problèmes difficiles ont en fait de tels algorithmes, mais non encore découverts ?

On conjecture depuis longtemps l'existence d'une classe de problèmes intrinsèquement difficiles.

La théorie de la complexité a été développée vers 1970 à travers les travaux de Stephen Cook [6] et Richard Karp [33], pour répondre à cette question. Elle se heurte à d'énormes difficultés théoriques et n'a pu obtenir pour l'instant que des résultats partiels.

Le principal résultat est que tous les problèmes difficiles sont liés : la découverte d'un algorithme polynomial pour un seul d'entre-eux permettrait de déduire des algorithmes polynomiaux pour tous les autres.

A cause des difficultés rencontrées, la théorie de la complexité ne traite que des problèmes de reconnaissance, car leur formalisme à réponse Oui- Non, Vrai- Faux a permis de les étudier avec les outils de la logique mathématique. Et comme tout problème d'optimisation combinatoire est au moins aussi difficile que son problème de reconnaissance associé, tout résultat établissant la difficulté du second concernera a fortiori le premier.

1.5.2 Les classes de problèmes $\mathcal{P}$ et $\mathcal{NP}$

Définition 1.7 L'ensemble des problèmes qui admettent des algorithmes polynomiaux forme *la classe $\mathcal{P}$.*

La Théorie de la Complexité se concentre sur les problèmes de reconnaissance qui ont une importance pratique. Le critère de praticabilité retenu est de pouvoir vérifier en temps polynomial une proposition de réponse Oui. Ce critère est appelé parfois *principe du superviseur*, d'où la donnée de la première définition de la classe $\mathcal{NP}$: ayant trouvé que la solution à un problème de reconnaissance est Oui (par un algorithme polynomial ou pas), peut-on en convaincre « facilement » (polynomialement), une tierce personne (le superviseur) qui, elle, n'a pas cherché la solution ?

Le symbole $\mathcal{NP}$ provient de ce que les problèmes de cette classe sont ceux qui peuvent être résolus par une machine de Turing *non déterministe* en temps polynomial (Non determinist Polynomial). Les algorithmes *non déterministes* diffèrent des algorithmes *déterministes* en ce qu'ils ne peuvent être mis en œuvre sur ordinateur.

Définition 1.8 Un algorithme *non déterministe* est un algorithme contenant une instruction « choix » qui, opérant sur un ensemble fini, choisit un élément, sans spécifier comment ce choix est effectué. Il est caractérisé par le fait que s'il existe une manière (au moins) d'effectuer le choix qui conduise à la réponse Oui, c'est suivant cette manière que le choix est fait.

Par conséquent, sur le plan théorique, les algorithmes non déterministes permettent de caractériser la classe $\mathcal{NP}$:

Définition 1.9 Un problème d'optimisation combinatoire est dit appartenir à *La classe* $\mathcal{NP}$ s'il peut être résolu en temps polynomial par un algorithme non déterministe.

Comme tout problème de *la classe* $\mathcal{P}$ possède un algorithme polynomial pour sa résolution, on en déduit que $\mathcal{P} \subset \mathcal{NP}$.

1.5.3 Les problèmes $\mathcal{NP}$ – complets

Il s'agit des problèmes les plus difficiles de la classe $\mathcal{NP}$, « le noyau dur » en quelque sorte. La notion de problème $\mathcal{NP}$*- complet* est basée sur celle de *la réduction polynomiale*.

Définition 1.10 Un problème de reconnaissance (P_1) *se réduit polynomialement* à un autre (P_2), s'il existe un algorithme pour (P_1) qui fait appel à un algorithme de résolution de (P_2) et si cet algorithme de résolution de (P_1) est polynomial lorsque la résolution de (P_2) est comptabilisée comme une opération élémentaire.

Définition 1.11 Un problème $\mathcal{NP}$*- complet* est un problème de $\mathcal{NP}$ en lequel se réduit polynomialement tout autre problème de $\mathcal{NP}$.

Remarque 1.2 On trouve souvent dans la littérature le terme « $\mathcal{NP}$*- difficile* » qui diffère du terme « $\mathcal{NP}$*- complet » :* Un problème d'optimisation combinatoire est $\mathcal{NP}$*- difficile* si le problème de reconnaissance associé est $\mathcal{NP}$*- complet*.

Les problèmes $\mathcal{NP}$- complets représentent le noyau dur de $\mathcal{NP}$, car si on trouvait un algorithme polynomial $\mathcal{A}$ pour un seul problème $\mathcal{NP}$*- complet* (X), on pourrait en déduire un

autre pour tout autre problème difficile (Y) de $\mathcal{NP}$. En effet, ce dernier algorithme consisterait à réduire polynomialement les données pour (Y) en données pour (X), puis à exécuter l'algorithme pour (X).

Une question immédiate est de savoir si de tels problèmes existent réellement dans $\mathcal{NP}$.

Définition 1.12 *Le problème de satisfaisabilité* est défini par la donnée d'un ensemble $X = \{x_1, x_2, \ldots, x_n\}$ de variables booléennes et d'une expression booléenne $E = C_1 \wedge C_1 \wedge \ldots \wedge C_m$ en termes de ces variables où chaque « clause » C_i (i = 1, 2,…, m) est une expression $C_i = u_1 \vee u_2 \vee \ldots \vee u_{k(i)}$ dont u_j est l'une des variables x_k complémentée ou non. Le problème consiste à chercher s'il existe une affectation des variables x_k à 0 ou 1 telle que $E = 1$.

Le logicien Cook a montré en 1970 que le problème de *satisfaisabilité* est $\mathcal{NP}$–complet : tout autre problème de $\mathcal{NP}$ peut s'y réduire polynomialement. Il existe donc bien au moins un problème dans cette classe. Depuis cette date, les chercheurs ont montré que la plupart des problèmes de reconnaissance associés aux problèmes d'optimisation combinatoire sans algorithmes polynomiaux connus sont en fait $\mathcal{NP}$–complets.

1.5.4 La conjecture $\mathcal{P} \neq \mathcal{NP}$

La question cruciale de la Théorie de la Complexité est de savoir si les problèmes $\mathcal{NP}$–complets peuvent être résolus polynomialement, auquel cas $\mathcal{P} = \mathcal{NP}$, ou bien si on ne leur trouvera jamais d'algorithmes polynomiaux, auquel cas $\mathcal{P} \neq \mathcal{NP}$. Cette question n'est pas définitivement tranchée. Et comme des centaines de problèmes $\mathcal{NP}$–complets résistent en bloc à l'assaut des chercheurs, on conjecture aujourd'hui que $\mathcal{P} \neq \mathcal{NP}$ et la preuve définitive nécessitera probablement l'invention de nouvelles mathématiques.

Un répertoire d'environ trois cents problèmes peut être trouvé dans le livre de Garey et Johnson [14]. Si votre problème est $\mathcal{NP}$–complet, il vaut mieux abandonner l'espoir de

trouver un algorithme polynomial. Si vous trouverez un, ce qui est peu probable, vous deviendrez célèbre (à défaut de devenir riche) : vous aurez infirmé la conjecture $\mathcal{P} \neq \mathcal{NP}$!

Mais, *comment résoudre ce genre de problèmes ?*

En pratique, on « résout » le problème en tirant parti :

• De la *taille des données.* L'énumération complète peut être valable sur de petits cas.

• De la « *complexité moyenne* ». Un algorithme exponentiel sur son pire cas peut être assez rapide en moyenne (comme l'algorithme du simplexe en programmation linéaire).

• Des *méthodes diminuant le combinatoire*. Il s'agit des méthodes arborescentes et de la programmation dynamique, sortes de méthodes d'énumération intelligentes. Ces méthodes sont actuellement les plus efficaces pour résoudre optimalement des problèmes $\mathcal{NP}$- difficiles de taille moyenne.

• Des *méthodes heuristiques*. Ces méthodes donnent des solutions avec plus ou moins de garantie de leur qualité et elles s'évertuent à trouver une solution possible parmi un ensemble de solutions.

1.6 Les heuristiques

Définition 1.13 Une *heuristique* de résolution d'un problème (P), est un algorithme donnant une solution réalisable avec plus ou moins de garantie de sa qualité. Ainsi, il est tout à fait possible qu'un algorithme de ce type donne une solution très éloignée de la solution optimale comme il est possible qu'il donne la meilleure solution. Ces algorithmes donc s'évertuent à trouver une solution possible parmi un ensemble de solutions.

Définition 1.14 Un algorithme est *approximatif* ou *approché* de la résolution d'un problème (P) lorsqu'il donne une solution s'approchant de la solution désirée (optimale). On doit évidemment connaître la valeur réelle de cette solution ou du moins, on peut garantir un

certain degré de précision. Il a donc la double capacité de donner une solution réalisable au problème et son évaluation par rapport à l'optimum.

Il faut noter que les deux types d'algorithmes ne sont pas étanches. Pour certains sous-ensembles d'instances d'une classe de problèmes, un algorithme heuristique peut devenir approximatif.

Le plus souvent dans la littérature, on omettra cette nuance en utilisant carrément le terme *heuristique* dans la mesure où :

- Elle est de complexité raisonnable (idéalement polynomiale).
- Elle fournit le plus souvent une solution proche de l'optimum.
- La probabilité d'obtenir une solution de mauvaise qualité est faible.
- Elle est simple à mettre en œuvre.

1.6.1 Evaluation des heuristiques

Le problème de l'évaluation des heuristiques est crucial. Les heuristiques n'offrent aucune garantie d'optimalité : elles peuvent trouver l'optimum pour certaines instances, ou en être très éloignées. Supposons qu'on étudie un problème combinatoire pour lequel on dispose déjà d'une méthode exacte (optimale) de référence. Pour une heuristique H et une instance I, on note $H(I)$ le coût de la solution heuristique et $OPT(I)$ le coût optimal.

Définition 1.15 On appelle *performance relative* de H sur I le quotient : $R_H(I) = \frac{H(I)}{OPT(I)}$.

Pour un problème de minimisation, $R_H(I) \geq 1$. La performance relative peut être bornée *a priori* ou être imprévisible et constatée *a posteriori*, c'est-à-dire après l'exécution de l'algorithme.

1.6.1.1. Evaluation a priori (Performance relative au pire)

Définition 1.16 On appelle *performance relative au pire des cas (*worst case performance ratio) P_H d'une heuristique *H*, sa plus mauvaise performance relative sur l'ensemble de toutes les instances possibles : $P_H = Max_I[R_H(I)]$.

Il s'agit d'une *garantie de performance*, qu'on obtient mathématiquement par analyse théorique de *H*, et non par des statistiques ou par énumération des instances possibles. Les démonstrations se font souvent en deux temps : on borne $R_H(I)$ pour toute instance *I*, puis on construit une instance montrant que ce pire des cas peut être effectivement atteint.
Ce genre de résultats est en général difficile à obtenir, et on n'en connaît que pour quelques heuristiques. Heureusement, les pires écarts à l'optimum sont souvent obtenus sur des problèmes théoriques construits exprès : les heuristiques peuvent être très bonnes en moyenne sur des problèmes pratiques.

1.6.1.2. Evaluation a posteriori

a) Bornes inférieures de l'optimum

Le plus souvent, on ne sait pas établir mathématiquement le comportement au pire des cas. On ne peut alors évaluer les résultats qu'après exécution de l'heuristique *H*. Pour évaluer le résultat pour une instance donnée *I*, on peut calculer le ratio de performance relative $R_H(I)$ à condition de connaître l'optimum qui n'est pas calculable en une durée acceptable pour les problèmes de grande taille. On peut cependant obtenir une évaluation moins serrée si on dispose d'une *évaluation par défaut* (minorant) *B(I)* pour *OPT(I).*
Alors : $R_H(I) = \dfrac{H(I)}{OPT(I)} \leq \dfrac{H(I)}{B(I)}$.

On peut toujours trouver des bornes inférieures, même grossières, de l'optimum *OPT(I).*

b) Evaluation statistique

On est souvent contraint à cette extrémité pour les grands problèmes réels. L'évaluation la plus rudimentaire consiste à comparer nos solutions heuristiques avec celles des méthodes actuelles (informatisées ou manuelles). Toute amélioration relative est économiquement intéressante. Même si on connaît la performance relative au pire des cas, cette dernière peut être très mauvaise, ou atteinte seulement sur des cas pathologiques. Et l'heuristique peut être bonne en moyenne ce qui recommande une *évaluation statistique*.

1.6.2. Les types d'heuristiques

1.6.2.1 Heuristiques gloutonnes « greedy »

Les algorithmes gloutons sont caractérisés par deux idées :

- Ce type d'algorithmes *ne revient jamais en arrière*, de là leur nom ; ils sont gloutons.
- L'examen des données dans *un ordre prédéfini* afin de construire une solution de manière incrémentale.

Il existe fréquemment un « *bon ordre* » des éléments d'une instance *I*, qui garantit que l'heuristique gloutonne donnera la solution optimale. Mais ce n'est pas toujours le cas :
les algorithmes gloutons donnent la solution optimale si l'ensemble des solutions possède une structure de *matroïde* [46], [36].

Définition 1.17 Etant donnés un ensemble fini E et une famille $\mathcal{F}$ de parties de E, si

1. $\varnothing \in \mathcal{F}$
2. $I \in \mathcal{F} \Rightarrow \forall J \subseteq I, J \in \mathcal{F}$

Alors, le système $(E, \mathcal{F})$ est appelé *système d'indépendance.*

Définition 1.18 On appelle *matroïde* tout système d'indépendance vérifiant la propriété suivante :

3. $\forall A \subseteq E \Rightarrow$ tous les indépendants maximaux de A ont le même cardinal.

Algorithme glouton Soient (E, $\mathcal{F}$) un système d'indépendance et une application poids $w \rightarrow \mathrm{IR}^{+}$, Le problème consiste à déterminer un indépendant de pois maximum.

Début

1. Ordonner les éléments de E par ordre décroissant de leurs poids :
   $w(e_1) \geq w(e_2) \geq \ldots \geq w(e_n)$. Poser
   $I := \emptyset$, j := 1,
2. Si $I + e_j \in \mathcal{F}$, alors poser $I := I + e_j$,
3. Si j < n, alors poser j := j + 1, aller en (1), sinon, terminer. Sélectionner I.

Fin.

Théorème 1.1 [36] *L'heuristique gloutonne donne un indépendant de pois maximum si et seulement le système* (E, $\mathcal{F}$) *est un matroïde.*

1.6.2.2 Méthodes par recherches locales

Soit un problème d'optimisation combinatoire (S, f), où S est l'ensemble des solutions réalisables et f la fonction objectif de S dans *IR*. *Un algorithme de recherche locale* a la structure générale suivante, dans laquelle *V(s)* est un *voisinage* de la solution *s* :

s := une solution initiale de *S* (calculée par toute heuristique)

z := *f*(s)

Tant qu'il existe *s'* dans *V(s)* avec *f(s')* < z

  s := *s'*

  z := *f(s')*

Fin tant que

Sortir *s* et z.

La solution initiale peut être générée aléatoirement ou être obtenue par une méthode gloutonne. En général, il vaut mieux partir d'une solution assez bonne. Pour le voisinage, si *V(s)* = *S* pour tout *s*, on se ramène à une énumération complète, en général exponentielle.
Si par contre |*V(s)* | est trop petit devant |*S*|, la recherche locale est facilement piégée dans un *minimum local*.

Le voisinage est en général un compromis choisi pour être calculable rapidement et permettre si possible un passage de toute solution à toute autre, même en plusieurs étapes. On peut chercher à chaque itération la meilleure solution du voisinage (technique de la plus grande pente) ou aller sur la première solution améliorante trouvée. La recherche locale se termine après un nombre variable d'itérations, quand elle arrive en une solution *s* qui est la meilleure de son voisinage.

1.6.2.3 Méthodes de recherche globale (Méta-heuristiques)

.

Ces méthodes sont en fait des recherches locales modifiées dans l'espoir d'éviter le piégeage dans un minimum local. Leur conception commence par l'étude d'une recherche locale, que

l'on promeut ensuite en recherche globale si elle s'avère insuffisante. Il s'agit donc de squelettes, de méthodes génériques, appelées aussi pour cette raison *méta-heuristique.*

a) Recuit simulé

Le *recuit simulé (simulated annealing)* a été inventé par les physiciens Kirkpatrick, Gellat et Vecchi en 1983 [34]. Ils ont ainsi pu résoudre de manière quasi optimale des problèmes du voyageur de commerce à 5000 sommets, avec, il est vrai, des heures de calcul.

L'analogie historique s'inspire du *recuit de métaux* en métallurgie. Un métal refroidi trop vite présente de nombreux défauts microscopiques, c'est l'équivalent d'un minimum local pour un problème combinatoire. Si on le refroidit lentement, les atomes se réarrangent, les défauts disparaissent, et le métal a alors une structure très ordonnée, équivalent du minimum global.

b) Méthode taboue

Les *recherches taboues* (*tabu* ou *taboo search)* ont été inventées par Glover vers 1985 [16]. Elles sont de conception plus récente que le recuit, n'ont aucun caractère stochastique et paraissent meilleure à temps d'exécution égal. Elles sont caractérisées par trois points fondamentaux :

- A chaque itération, on examine complètement le voisinage $V(s)$ de la solution actuelle s, et on va sur la meilleure solution s', même si le coût remonte.
- On *s'interdit de revenir sur une solution visitée dans un passé proche* grâce à une *liste taboue T (tabu list)* stockant de manière compacte la trajectoire parcourue. On cherche donc s' dans $V(s)- T$.

- On conserve la *meilleure solution trouvée* en cours de route car, contrairement au recuit, c'est rarement la dernière. On stoppe après un nombre maximal NMax d'itérations, ou après un nombre maximal d'itérations sans améliorer la meilleure solution trouvée, ou quand $V(s)- T = \varnothing$.

Ce dernier cas ne se produit que sur de très petits problèmes, pour lesquels le voisinage tout entier peut se retrouver enfermé dans T.

On voit qu'au cours de sa progression, une méthode taboue échappe aux minima locaux : même si *s* est un minimum local, l'heuristique va s'échapper de la région *V(s)* en empruntant un *col*.

c) Algorithmes génétiques

Cette classe de méthodes a été inventée par Holland dans les années 60, pour imiter les phénomènes d'adaptation des êtres vivants. L'application aux problèmes d'optimisation a été développée ensuite par Goldberg [17].

Par analogie avec la reproduction des êtres vivants, on part d'une *population* initiale de *N* solutions aléatoires. Le point délicat est d'arriver à coder chaque solution comme une chaîne de caractères, analogue à un *chromosome* d'une cellule vivante. Le chromosome est formé de sous-chaines appelées *gènes,* chacun codant une caractéristique de la solution.

Une itération est appelée *génération.* A chaque génération, on choisit au hasard $NC < N$ paires de chromosomes à reproduire, avec une certaine probabilité. Chaque paire (x, y) choisie subit une opération de *croisement (crossover)* : un gène est choisi aléatoirement dans x, puis permuté avec le gène de même position dans y. On pratique également un faible taux de *mutations* : *NM* chromosomes sont choisis et subissent une modification aléatoire d'un gène.

La nouvelle population est formée de la population précédente et des chromosomes nouveaux générés par mutation ou croisement. On définit pour chaque solution une *mesure d'adaptation au milieu (fitness).* Pour un problème d'optimisation, cette mesure est tout simplement la

fonction objectif. On élimine alors les solutions les moins adaptées pour maintenir un effectif constant de *N* solutions. On recommence le même processus pour l'itération suivante.
L'intérêt de ces méthodes est que les bonnes solutions sont encouragées à échanger par croisement leurs caractéristiques et à engendrer les solutions encore meilleures. De plus, les solutions finales vont être concentrées autour des minima locaux, et la méthode fournira donc un choix de plusieurs solutions possibles au décideur.

Les algorithmes génétiques demandent un effort de calcul important. Ils commencent à être appliqués aux problèmes d'optimisation, mais avec des résultats encore inférieurs à ceux du recuit simulé et de la méthode taboue.

Chapitre 2

Le problème de placement

2.1 Introduction

Ce chapitre, comme l'énonce son titre, se préoccupe de l'un des problèmes difficiles de l'optimisation combinatoire qui constitue le thème de notre thèse et qui a fait l'objet de plusieurs travaux.

Le problème de placement (*the bin packing problem*), sera présenté par ses différentes formulations. Nous parlerons également de sa $\mathcal{NP}$- complétude.

La dernière partie de ce chapitre sera consacrée aux différentes variantes de ce problème.

2.2 présentation du problème de placement

Le problème de placement standard est défini par la donnée d'une liste de n pièces $L = (i : 1 \leq i \leq n)$, chacune de taille $s(i) = a_i \leq 1$, $i = 1,\dots, n$ et une collection infinie de boites $\mathcal{B}_j$, de même capacité $c(\mathcal{B}_j) = 1$, pour tout j. On s'intéresse à placer toutes les pièces de cette liste dans un nombre minimum de boites.

Parmi la panoplie des déférentes situations rencontrées dans la pratique que le problème du placement modélise, on en trouve ceux de chargement de véhicules, de groupage, de découpage de matériaux, de conditionnement,…

Remarque 2.1 Il ne faut pas confondre bin packing et problème d'empilement : dans le premier, on cherche à prendre tous les objets en minimisant le nombre de conditionnements, dans le second on n'a qu'un conditionnement, qu'on cherche à remplir au maximum sachant qu'on ne peut pas tout prendre.

2.3 Formulations du problème de placement

Le problème de placement standard peut être formulé sous plusieurs versions. Nous en çitons quelques- unes.

2.3.1 Le problème de placement et la programmation mathématique

En utilisant la terminologie du problème du *sac à dos* « *knapsack problem* », le problème de placement se formule comme suit :

Etant donnés n pièces et n sacs à dos (ou boites), avec

a_j : le poids de la pièce j,
c : la capacité de chaque boite,

on affecte chaque pièce à une seule boite de sorte que le poids total des pièces de chaque boite n'excède pas c, en minimisant le nombre de boites utilisées. Formellement [40], on aura :

$$\text{Min } Z = \sum_{i=1}^{n} y_i ,$$

Sous contraintes,

$$\sum_{j=1}^{n} a_j x_{ij} \leq c y_i, \quad i \in \{1,2,\ldots, n\},$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad j \in \{1,2,\ldots, n\},$$

$$y_i \in \{0, 1\}, \quad i \in \{1,2,\ldots, n\},$$

$$x_{ij} \in \{0, 1\}, \quad i,j \in \{1,2,\ldots, n\},$$

où

$$y_i = \begin{cases} 1 & \text{si la boite i est utilisée;} \\ 0 & \text{sinon,} \end{cases}$$

$$x_{ij} = \begin{cases} 1 & \text{si la pièce j est assignée à la boite i;} \\ 0 & \text{sinon.} \end{cases}$$

Sans perte de généralité, on suppose que c est un entier positif et $a_j \leq c$, pour tout j.

2.3.2 Le problème de placement et la programmation linéaire

Soit $\mathcal{P}$(L) l'ensemble de tous les sous-ensembles de $L = \{\ i : 1 \le i \le n\ \}$ et soit A la matrice dont les éléments (a_{iS}), pour $i = 1, 2,\ldots, n$ et pour tout $S \in \mathcal{P}$(L), sont définis par

$$a_{iS} = \begin{cases} 1 & \text{si i} \in \text{S;} \\ 0 & \text{sinon.} \end{cases}$$

Le problème de placement se formule [5], comme suit :

$$\text{Min Z} = \sum_{S \in P(L)} y_s\,,$$

Sous contraintes :

$$\sum_{S \in P(L)} x_{iS} y_s = 1, \qquad i = 1,\ldots, n$$

$$y_S \in \{0, 1\}, \qquad \forall\ S \in \mathcal{P}(\text{L})$$

tels que,

$$y_S = \begin{cases} 1 & \text{si toutes les pièces de S sont placées dans une même boite,} \\ 0 & \text{sinon.} \end{cases}$$

2.3.3 Le problème de placement et les hypergraphes

Définition 2.1 Soit $X = \{x_1, x_2,\ldots, x_n\}$, un ensemble fini, et $\mathcal{E} = \{E_i\}_{i\in I}$ une famille de parties de X avec $I = \{1, 2, \ldots, n\}$. On dira que $\mathcal{E}$ constitue un *hypergraphe* sur X si l'on a :

(1) $E_i \neq \varnothing \quad (i\in I)$

(2) $\bigcup_{i\in I} E_i = X$

Le couple $\mathcal{H} = (X, \mathcal{E})$ s'appelle un *hypergraphe,* et son ordre est $|X| = n$. Les éléments $x_1, x_2,\ldots, x_n$ de X sont les sommets de l'hypergraphe, et les sous ensembles de $\mathcal{E}$, que l'on dénote $E_1, \ldots, E_m$ sont ses arêtes.

Sur une figure, on représente E_i par un trait plein entourant ses éléments si $|E_i| > 2$; si $|E_i| = 2$, par un trait continu joignant ses deux éléments comme l'arête d'un graphe ; ou, si $|E_i| = 1$, comme une boucle d'un graphe.

Dans un hypergraphe, deux arêtes E_i et E_j sont dites adjacentes si leur intersection est non vide. (Voir figure 1.1)

Un sous-ensemble $\mathcal{E}_0 \subseteq \mathcal{E}$ est appelé *couplage* de $\mathcal{H}$ (*matching*), si toute paire de ses arêtes n'a aucun sommet commun.

Un couplage est *parfait* si ses arêtes contiennent tous les sommets de X.

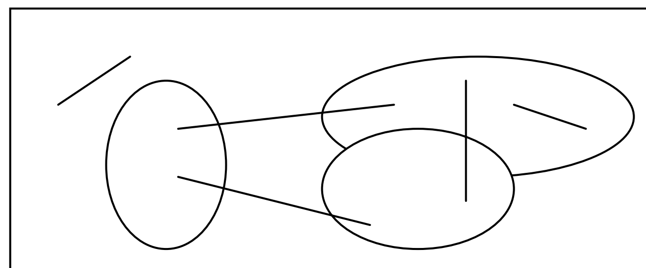


Figure 1.1

Le problème de placement peut donc prendre une autre facette de modélisation sous la forme de la recherche d'un *couplage parfait de cardinal minimum* dans l'hypergraphe $\mathcal{H} = (X, \mathcal{E})$ construit de manière que l'ensemble $X = \{x_1, x_2, \ldots, x_n\}$désigne l'ensemble des pièces $L = \{ i : 1 \leq i \leq n \}$ et une arête entoure un ensemble de sommets si la somme de leurs poids est inférieure à un (la capacité de chaque boite) [1].

Exemple. Soit une liste de pièces, $L = \{1, 2, 3, 4\}$ de tailles 1/2, 1/4, 1/4 , 3/4. L'hypergraphe suivant (voir figure 1.2) montre différentes possibilités de placements de ces pièces dans des boites de capacité égale à 1.

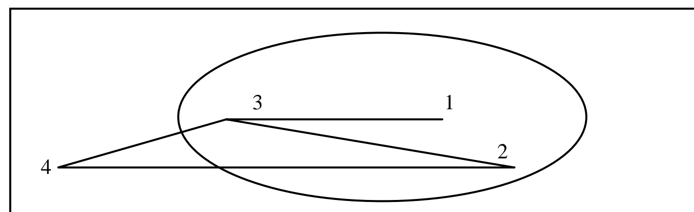


Figure 1.2

2.3.4 Le problème de placement et les graphes

Quelques fois on rencontre des conflits lors du placement d'une liste de pièces, ce qui veut dire que certaines pièces ne peuvent être mises dans une même boite pour des raisons bien précises.

On représente ce phénomène par un graphe $\mathcal{G} = (V, E)$, appelé *graphe conflit,* où les sommets de V représentent les pièces de la liste $L = \{ i : 1 \leq i \leq n \}$de cardinal n et les arêtes de E représentent les conflits entre les pièces. Autrement dit : les extrémités d'une arête ne sont que les pièces qui sont en conflit et ça sera impossible de les mettre dans une même boite. Ce graphe est muni d'une fonction w appelée *fonction poids* qui désigne la taille de chaque pièce de L.

Une solution *réalisable* est une partition des pièces en boites $B_1, \ldots, B_m$, telle que chaque boite B_i contient les pièces $B_{i1}, \ldots, B_{m_i}$ en satisfaisant :

$$\forall\, i. \qquad \sum_{j=1}^{m_i} w(B_{ij}) \leq 1$$

$$\forall i, j, k. \qquad \{B_{ij},\ B_{ik}\} \notin E.$$

Des outils de la théorie des graphes comme la recherche du nombre chromatique intervient ici pour résoudre ce type de problèmes du problème [29].

2.4 La complexité du problème de placement

Le problème de placement n'est pas seulement $\mathcal{NP}$-complet, mais davantage il est *fortement $\mathcal{NP}$-complet*.

Une manière de définir la notion de $\mathcal{NP}$-complétude forte, est d'utiliser un codage unaire pour les données numériques. Si le problème demeure $\mathcal{NP}$-complet, alors le problème est fortement $\mathcal{NP}$-complet, dans le cas contraire, il est faiblement $\mathcal{NP}$-complet. Formellement :

Définition 2.2 On dit qu'un problème est *pseudo-polynomial*, si et seulement s'il existe un algorithme, dit, également *pseudo-polynomial*, dont la complexité est bornée par un polynôme en la place prise en mémoire par les données, en acceptant d'utiliser un codage unaire pour les quantités numériques. Un problème $\mathcal{NP}$-complet qui n'est pas pseudo-polynomial est dit *fortement $\mathcal{NP}$-complet*.

Théorème 2.1. [14] *Le problème de placement est fortement $\mathcal{NP}$-complet.*

La démonstration de la $\mathcal{NP}$-complétude du problème de placement se fait en quatre étapes :

(a) montrer qu'il s'agit bien d'un problème appartenant à la classe $\mathcal{NP}$: A partir d'un algorithme non déterministe, on peut vérifier en un temps polynomial, que dans une solution composée de *m*

boites, le contenu de chacune d'entre elles est de taille n'excédant pas sa capacité.

(b) Choisir un problème connu qui soit $\mathcal{NP}$-complet : pour cela, on considère le problème de 3-partition qu'on définira ultérieurement.

(c) Construire une transformation f du problème de 3-partition en le problème de placement.

(d) Prouver que la transformation f est polynomiale.

Le problème de 3-partition est défini comme suit :

Données : Un ensemble fini $A = (1,2,...,3m)$ de $3m$ éléments, où chaque élément $i \in A$, est de taille $s(i) \in Z^+$ et une borne $B \in Z^+$, tels que : $\sum_{i \in A} s(i) = mB.$ et $\frac{B}{4} < s(i) < \frac{B}{2}, i = 1,...,3m$

Question : peut-on partitionner A en m sous-ensembles disjoints $S_1, S_2,..., S_m$ tels que $\sum_{i \in s_j} s(i) = B, j = 1,...,m$?

Théorème 2.2. [14] *Le problème de 3-parttion est fortement $\mathcal{NP}$- complet.*

Le problème de 3-partition est un cas particulier du problème de 3-couplage, dont la $\mathcal{NP}$-complétude découle de celle du problème de satisfaisabilité.

Une autre manière immédiate pour démontrer que le problème de placement est $\mathcal{NP}$-complet consiste à trouver un problème $\mathcal{NP}$-complet connu *Y en cas particulier* du problème de placement. Pour cela, il suffit d'observer que le problème de 3-partition est un cas particulier du problème de reconnaissance associé au problème de placement. Ce dernier se formule comme suit :

Données : Une liste $L = \{1, 2,\dots, n\}$ de n pièces, chaque pièce i est de taille s(i) et un nombre

entier positif k.

Question : Existe-t-il une partition de L en k sous-ensembles disjoints S_1, S_2, …, S_k ?

Pour les valeurs

- $n = 3m$,
- $\frac{B}{4} < s(i) < \frac{B}{2}$ et $\sum_{i \in A} s(i) = mB$.
- $k = m$.

On aura bien le problème de 3-partition.

2.5 Les différentes variantes du problème de placement

Vu que le problème de placement est $\mathcal{NP}$-complet, nous sommes condamnés à penser beaucoup plus et à être destinés vers la quête des heuristiques et des méthodes approchées au lieu des méthodes exactes. Heureusement, il existe beaucoup d'algorithmes approchés simples qui ont tendance à mieux se comporter pour la plupart des problèmes de placement. En outre, de nombreuses situations du problème possèdent des contraintes supplémentaires qui entravent extrêmement le déroulement des algorithmes appropriés à un cas spécifique du problème de placement dans leur résolution. Et ainsi, il y a des facteurs selon lesquels on distingue les variétés du problème de placement- qui ont une incidence sur le choix d'un algorithme :

- *Quelles sont les formes et les tailles des pièces ?* – Toute pièce est connue à partir se sa taille qui peut être une longueur, un poids, un volume…et sa forme géométrique plane ou tridimensionnelle, régulière ou irrégulière. Ces deux caractéristiques déterminent la nature du problème en terme de dimension.
- *Les boites sont-elles de taille uniforme ou variable ?* – Il sera mieux et même idéal de placer chaque pièce dans sa propre boite de

capacité identique à sa taille que d'utiliser uniquement des boites unitaires. En partant de cette remarque, on différencie deux types du problème de placement : celui de taille uniforme et celui de taille variable.

- *Y- a- t- il des conditions à respecter lors du placement des pièces ?* – A titre d'exemple : dans le chargement d'un camion, certaines caisses ne peuvent pas être mises en dessus d'autres qui soient plus lourdes qu'elles, par suite on doit prendre en considération cette contrainte. Le rajout de contraintes crée une sorte de restriction du problème de placement.

Comme il est impossible d'étaler toute la liste de ces variantes, on se contente de décrire celles qui suivent :

2.5.1. Problème de placement unidimensionnel

C'est sous cette variante du problème de placement que notre travail s'inscrit :
Le but est de placer une liste donnée de pièces, chacune étant connue par un seul paramètre qui peut être un poids, une longueur … en utilisant le plus petit nombre possible de boites. Plus de détails seront apportés dans le chapitre suivant.

2.5.2. Problème de placement de dimension deux (bidimensionnel)

Le problème de placement bidimensionnel dépend d'une manière intense des formes des pièces planes : elles peuvent être de forme régulière (rectangles, carrés) ou irrégulière, avec une largeur $w(i)$ et une hauteur $h(i)$.

Il s'agit de placer toutes ces pièces considérées planes, dans un nombre minimum de carrés unitaires de sorte que les pièces soient entièrement contenues dans les boites. Les cotés des pièces doivent être parallèles à celles des boites et deux pièces ne se chevauchent pas si elles sont de forme régulière (voir figure 2.1)

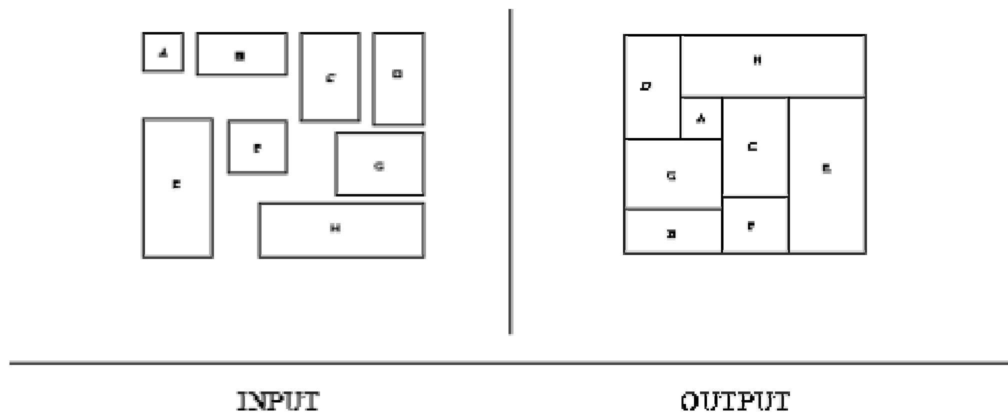


figure 2.1

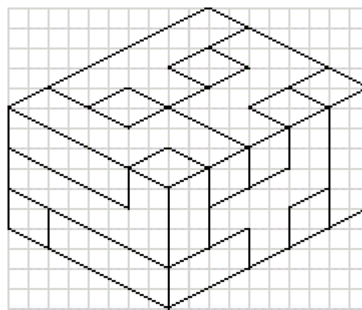
Sur le plan pratique, on rencontre souvent ce problème qui modélise une grande variété de problèmes de la production industrielle et de conditionnement ; par exemple, le découpage des bandes de tissus : étant donné une bande de tissu de longueur limitée et un ensemble de pièces à découper de forme (ir)régulière. Le but est de trouver le meilleur placement de l'ensemble de ces découpes afin de minimiser la longueur de la bande à utiliser. Pour résoudre ce type de problèmes, une nouvelle méthode heuristique qui s'appuie principalement sur une recherche locale a été proposée par M. Hifi et R. M'Hallah [21].

2.5.3 Problème de placement tridimensionnel

Ce problème consiste à placer une liste de pièces ayant des formes géométriques tridimensionnelles dans un nombre minimum de cubes unitaires. La rotation des pièces peut être permise dans certaines situations comme elle peut être interdite.

Le problème de placement des pentaminos est un bon exemple dont l'orientation des pièces n'est pas fixe.

Les pentaminos sont des pièces connexes formées de 5 cubes assemblés dans le plan. Il en existe 12 différents, notés F, I, L, P, N, T, U, V, W, X, Y et Z en raison de leur forme.
Il s'agit de placer ces pentaminos dans un volume de dimension donnée ; les cas les plus difficiles sont ceux où le volume est un bloc dont le produit hauteur x longueur x largeur est égal à 60 (5 cubes pour chacun des 12 pentaminos, soit 60 cubes à placer), (voir figure 2.2).
Le nom "pentaminos" a été créé par Solomon W.Golomb en 1953 dans une présentation qu'il fit au *Harvard Mathematics Club*, mais le premier problème sur les pentaminos fut publié en 1907 dans les *Canterbury Puzzles* par Henry Ernest Dudeny.



Le 3 x 4 x 5 : 3940 solutions.

figure 2.2

2.5.4. Placement de rectangles en maximisant les profits (rectangle-packing maximizing benefits)

Ce problème est défini par la donnée de n rectangles $R_i = (a_i, b_i)$ chacun de profit $p_i > 0$, $i = 1,\ldots, n$. On s'intéresse à placer un sous-ensemble de rectangles dans un rectangle $R = (a, b)$ de largeur unitaire ($a = 1$) en maximisant le profit total des rectangles placés. Les rectangles ne doivent pas déborder R avec une possibilité de leur rotation.

K. Jansen et H. Zhang avaient traité ce problème et pour avoir plus de détails, le lecteur pourra se reporter à l'article [27].

2.5.5. Problème de placement des lots de pièces (Batched Bin packing)

Dans ce problème de placement, les pièces sont disponibles sous forme de lots et le placement s'effectue progressivement sur un arrivage une fois qu'il serait présent. Un « *algorithme entassé* » consiste à placer un arrivage avant l'arrivée du prochain. Un arrivage peut contenir plusieurs pièces ; le cas particulier est d'avoir tout simplement une pièce par chaque arrivage et qui n'est que le problème de placement standard.

Des bornes inférieures du ratio asymptotique au pire des cas ont été obtenues pour le cas de deux lots par G. Gutin, T. Jensen et A. Yeo [19].

2.5.6. Problème de placement avec conflits

Lors du placement d'une liste de pièces, certaines ne peuvent être mises ensemble pour une raison ou une autre ce qui crée des « *conflits* ». Comme application, on trouve le problème de placement de clients autour des tables lors d'une réunion de compagnies. Dans l'ordre où chaque client doit faire un nombre maximum de connaissances, les membres d'une même compagnie ne doivent pas s'asseoir ensemble. La théorie des graphes a été utilisée pour la résolution de ce problème dans les travaux de B. McCloskey et A. J. Shankar [41].

2.6 Conclusion

Différentes formulations du problème de placement ont été données dans cette partie, dans le but de l'éclaircir à notre lecteur. Sa $\mathcal{NP}$ – complétude laisse le chemin grand ouvert vers la quête de meilleures approches en termes de performance, plusieurs recherches ont été établies au dessous et dans le chapitre suivant on en parlera explicitement.

Chapitre 3

Résolution du problème de placement

3.1 Introduction

Le but de ce chapitre est de présenter deux grandes classes de méthodes générales pour résoudre le problème de placement : les méthodes *exactes* et les méthodes *approchées.*

On oppose les méthodes approchées aux méthodes exactes qui trouvent toujours l'optimum si on leur en laisse le temps soit par *énumération complète* ou bien par les *méthodes arborescentes* ou même la *programmation dynamique.*

Nous évoquons parmi les méthodes exactes celles qui sont les plus fréquemment utilisées, comme celles de *séparation et évaluation* également dites *méthodes arborescentes* (*branch and bound methods*).

Nous présentons également différentes méthodes approchées pour résoudre notre problème pour les deux types : onlines (en ligne) et off-lines (hors ligne).

3.2 Les méthodes exactes

Les algorithmes exacts de résolution du problème de placement sont peu nombreux dans la littérature en raison de la $\mathcal{NP}$-complétude de ce problème ce qui a stimulé le recours aux méthodes approchées. Néanmoins, nous allons en décrire quelques -unes.

3.2.1 Les méthodes arborescentes

3.2.1.1 Principes généraux

Les méthodes arborescentes sont des méthodes exactes d'optimisation qui pratiquent une énumération intelligente (qui évitent l'énumération complète) de l'espace des solutions. Elles partagent l'espace des solutions en sous-ensembles de plus en plus petits, la plupart étant éliminés par des calculs de bornes avant d'être construits explicitement. D'où le nom synonyme de *méthodes d'énumération implicite*.

Appliquées aux problèmes $\mathcal{NP}$- complets, ces méthodes restent bien entendu exponentielles, mais leur complexité en moyenne (relative à la taille moyenne des instances du problème) est bien plus faible que pour une énumération complète. Elles peuvent donc pallier le manque d'algorithmes polynomiaux pour des problèmes de taille moyenne.

Pour des problèmes de grande taille, leur durée d'exécution devient prohibitive, et il faut se tourner vers des heuristiques.

Soit un problème d'optimisation combinatoire défini par un ensemble S de solutions et une fonction objectif f de S dans IR . On peut inventer pour un tel problème plusieurs méthodes arborescentes. Toutes auront cependant trois composantes communes :

- Une *règle de séparation* ou de partitionnement des solutions (*branching rule*) ;
- Une *fonction d'évaluation* ou de bornage des solutions (*evaluation function*) ;
- Une *stratégie d'exploitation* (*search strategy*).

3.2.1.2 Les méthodes arborescentes et le problème de placement

En 1971, Eilon et Christofides [8] avaient proposé un algorithme *énumératif en profondeur d'abord* (*depth first enumerative algorithm*) basé sur une règle de séparation appelée « *best fit decreasing* ».

A tout moment de la construction de l'arborescence, les sommets non encore séparés forment les *feuilles* de l'arbre. Au niveau de ces sommets, on suppose qu'on ait b boites utilisées, on définit $(\bar{c}_{i_1}, \bar{c}_{i_2}, ..., \bar{c}_{i_b})$, le vecteur des capacités résiduelles de ces boites prises dans l'ordre croissant, et $\bar{c}_{i_{b+1}} \equiv c_{b+1} = c$ la capacité de la boite suivante non encore utilisée.

La phase de séparation consiste à affecter la plus grande pièce non encore placée, à tour de rôle, aux boites $i_s, \ldots, i_b, i_{b+1}$, tel que : $s = \min\{h : 1 \le h \le b+1, \bar{c}_{i_h} + w_j \le c\}$, tel que w_j représente la taille de la pièce j.

Pour élaguer les sommets, la borne inférieure suivante (le contenu moyen par boite) est utilisée : $L_1 = \left\lceil \frac{\sum_{j=1}^{n} w_j}{c} \right\rceil$.

En 1978, Hung et Brown [25] avaient proposé un autre algorithme de séparation et d'évaluation pour le problème de placement de taille variable. Leur règle de séparation se base sur une caractérisation d'affectations équivalentes qui réduit le nombre de sommets de décision explorés. Ils ont néanmoins utilisé la même borne inférieure L_1 afin d'effectuer l'opération d'élagage.

En se basant sur les résultats calculatoires reportés dans [8] et [25], on attire l'attention du lecteur que le champ d'application des deux algorithmes ci- dessus se réduit uniquement aux petites instances du problème de placement.

Onze ans plus tard, en 1989, Martello et Toth [39], avaient suggéré un algorithme dit *MTP* (*Martelleo and Toth Procedure*), en s'appuyant sur la règle de séparation « *first fit decreasing* »
Initialement, les pièces sont triées par ordre décroissant de leurs tailles. L'algorithme indexe les boites dans l'ordre croissant de leur ouverture. A chaque sommet de décision, la plus grande pièce non encore placée est affectée, à tour de rôle, aux boites utilisées dans l'ordre croissant d'indices et à une nouvelle boite.

A chaque étape avancée ou encore dite « *forward* step », le calcul des bornes inférieures L_2 et L_3 [40] sert à élaguer les sommets et réduire le problème courant et dans le cas de l'impossibilité de l'élagage, on fait appel aux trois algorithmes approchés *FFD, BFD* et *WFD* (qu'on détaillera prochainement) dans l'espoir d'améliorer le meilleure solution déjà trouvée.

A chaque étape marche-arrière ou « *backtracking step* », on fait déplacer la pièce courante j_* de sa boite actuelle i_* vers la prochaine boite i_*+1, dans la mesure où cette dernière est débutée par la pièce j_*.

De plus, le critère suivant, appelé *critère de dominance* est utilisé pour ne pas explorer tous les sommets de décision :
Pour une pièce j_* affectée à une boite i_* de capacité résiduelle $\overline{c}_{i_*}$, si on a: $\overline{c}_{i_*} < a_{j_*} + a_n$, alors cette affectation domine toute autre affectation à la boite i_* des pièces $j > j_*$ et qui rend l'insertion d'au moins une autre pièce prohibitive.

<u>*Exemple illustratif*</u> :

Considérons l'instance du *MTP* définie par
$n = 10$; $(a_j) = (49, 41, 34, 33, 29, 26, 26, 22, 20, 19)$; $c = 100$.

Une solution réalisable est définie par un vecteur (b_j), tel que :
bj = la boite à laquelle la pièce j est assignée ($j = 1,\ldots, n$) ; la figure 3.1 représente l'arbre produite par l'algorithme *MTP*. Initialement, les bornes inférieures calculées sont de valeur 3 et l'algorithme *FFD* donne la première solution réalisable (b_j) = (1, 1, 2, 2, 2, 3, 3, 3, 3, 4), $z = 4$ correspondante aux sommets de décision 1-10. Aucun deuxième descendant n'est généré par les sommets 5-9, car il va se produire une solution de valeur 4 ou plus. Les sommets 11 et 12 sont taillés par la borne L_2. Le premier descendant du sommet 2 initie la boite 2, donc aucun autre descendant de ce sommet n'existe. Le premier descendant du sommet 13 est dominé par le sommet 2, car dans les deux cas aucune autre pièce ne peut être assignée à la boite 1 ; pour la même raison le sommet 2 domine le premier descendant du sommet 15. Le sommet 14 est élagué par la borne inférieure L_3. Au niveau du sommet 16, la procédure *MTRP* (*Martelleo and Toth Reduction Procedure*),dite aussi L_3, est appliquée.

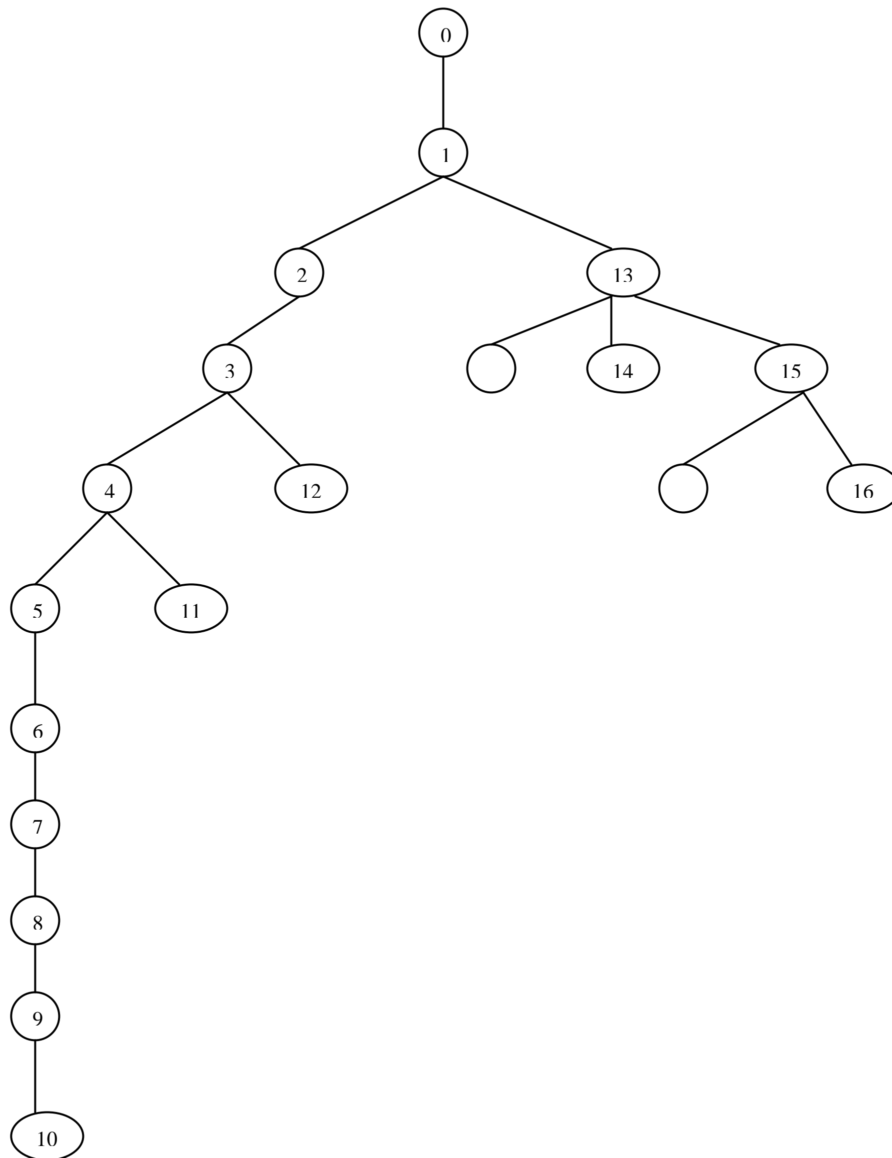


figure 3.1

3.3 Les méthodes approchées

Les algorithmes approchés ont été développés dans l'espoir d'apporter une réponse à l'impossibilité de la résolution d'une grande variété de problèmes d'optimisation. Si la solution optimale est inatteignable, alors il serait raisonnable de sacrifier l'optimalité et de se contenter d'une *bonne* solution réalisable qui puisse être obtenue efficacement. Les deux

critères que tout chercheur tente d'améliorer sont le temps de calcul de la solution et sa proximité de la solution optimale. Un algorithme approché est supposé toujours efficace.

La métrique généralement utilisée pour l'évaluation d'un algorithme approché, $\mathcal{A}$, est le *ratio de performance asymptotique du pire cas*, R_A^∞.

Définition 3.1 Un algorithme polynomial $\mathcal{A}$, est dit *δ-approximation* d'un problème (P) si pour toute instance I de (P), de solution optimale de valeur $OPT(I)$,

$$\frac{A(I)}{OPT(I)} \leq \delta .$$

Pour un problème de minimisation, $\delta \geq 1$. Le plus petit δ est le *ratio d'approximation* (ou *performance*) noté R_A de l'algorithme $\mathcal{A}$.

δ possède plusieurs appellations : la borne au pire cas, performance au pire cas, facteur d'approximation, ratio de performance, ratio d'erreur, borne de performance,…[22]

Définition 3.2 *Le ratio de performance absolu,* R_A, d'un algorithme approché $\mathcal{A}$ est,

$$R_A = inf\,\{\mathrm{r} \geq 1 \,/\, R_A(I) \leq r \text{ pour toute instance } I\}.$$

Le *ratio de performance asymptotique* R_A^∞ de l'algorithme $\mathcal{A}$ est,

$$R_A^\infty = inf\,\{r \geq 1 \,/\, \exists\, n \in Z^+,\ R_A(I) \leq r \text{ pour toute instance } I \text{ tel que } OPT\,(I) \geq n\}.$$

Dans le cas d'un problème de placement d'une liste de pièces $L = \{i : 1 \leq i \leq n\}$, la même métrique citée ci-dessus est utilisée pour évaluer un algorithme approché $\mathcal{A}$. En outre, si les pièces de la liste L sont toutes de taille inférieure à α, $\alpha \in IR^+$, alors :

$$R_A^\infty(\alpha) = inf\,\{r \geq 1 \,/\, \exists\, n \in Z^+,\ R_A(L) \leq r,\ \forall\, L \text{ tel que } OPT\,(L) \geq n \text{ et } s\,(i) \leq \alpha,\ \forall\, i \in L\ \}$$

Notons que pour $\alpha = 1$, nous avons $R_A^\infty(\alpha) = R_A^\infty$.

Le plus souvent, dans un problème de placement, toutes les pièces sont connues préalablement. Un algorithme qui place les pièces sous ces critères est dit *algorithme off-line.* Mais, il arrive parfois que les pièces ne soient connues qu'une seule à la fois et dés le moment d'arrivée d'une pièce, elle doit être immédiatement placée avant que la pièce suivante n'arrive, ce qui explique le phénomène de manque d'espace pour stocker des pièces avant leur placement. Dans une situation pareille, on utilise des algorithmes appelés *algorithmes online*.

3.3.1 Algorithmes Online (en ligne)

Le premier algorithme de type glouton, proposé pour résoudre le problème du placement et qui est considéré le plus simple est *Next Fit* (*NF*). Il a été décrit par Johnson en 1973 [30], comme suit :
Il place la pièce en question dans la dernière boite ouverte qui puisse la contenir, sinon il ferme cette boite définitivement et il ouvre une nouvelle boite dans laquelle il place notre pièce. Johnson avait montré que $R_{NF}^{\infty} = 2$ et que cet algorithme est de complexité $O(n)$.

First Fit (*FF*) est l'algorithme le plus utilisé. Il procède comme suit :
Pour placer une pièce, *First Fit* examine les boites ouvertes dans l'ordre croissant de leur ouverture et choisit la première boite qui puisse recevoir la pièce. Si aucune de ces boites n'est en mesure de recevoir la pièce en question, une nouvelle boite est ouverte. En 1976, Garey, Graham, Johnson et Yao [13] ont montré que $R_{FF}^{\infty} = 1.7$ et que *FF* se déroule en $O(n \log n)$.

Un autre algorithme ayant le même ratio de performance que *First Fit*, appelé *Best Fit* (*BF*) a été proposé. Il examine toutes les boites ouvertes et place la pièce en question dans la boite la plus remplie qui puisse la contenir. Cet algorithme se déroule, aussi, en $O(n \log n)$.

Contrairement à l'algorithme précédent, l'algorithme *Worst Fit* (*WF*) place la pièce dans la boite la moins remplie et ouvre une nouvelle boite si aucune boite ouverte ne peut la recevoir. Cet algorithme a le même ratio de performance que *Next Fit* et est de complexité $O(n \log n)$.

Il serait intéressent de contempler l'algorithme *Almost Worst Fit* (*AWF*) qui procède comme *WF*, seulement le choix de la boite se fait sur celle qui a le second plus petit niveau de remplissage. Ce qui est plus étonnant, c'est que *AWF* est meilleur que *WF* malgré la petite différence qui les distingue. En effet, selon Johnson [30], [32], $R_{AWF}^{\infty} = 1.7$

Plus généralement, il existe deux propriétés qui caractérisent les algorithmes online.

- la première dite la classe *Any Fit* (*AF*) et qui consiste à :

n'ouvrir une nouvelle boite pour placer la pièce courante
que si cette dernière ne peut être contenue dans aucune boite déjà utilisée.

Cette classe contient tous les algorithmes que nous venons de décrire.

- la seconde appelée la classe *Almost Any Fit* (*AAF*),

les algorithmes de cette classe *n'optent jamais pour la boite la moins remplie à moins qu'il n' existe plus d'une telle boite ou bien cette boite est la seule à être capable de recevoir la pièce en question.*

Les algorithmes *First Fit, Best Fit et Almost Worst Fit* appartiennent à cette classe, de plus la classe *AAF* est contenue dans la classe *AF*. Le théorème suivant caractérise ces deux classes :

Théorème 3.1. [30], [31]. Pour tout α, $0 < \alpha \leq 1$,

- Si $\mathcal{A}$ est un algorithme *Any Fit* alors, $R_{FF}^{\infty}(\alpha) \leq R_{A}^{\infty}(\alpha) \leq R_{NF}^{\infty}(\alpha)$.

- Si $\mathcal{A}$ est un algorithme *Almost Any Fit* alors, $R_{A}^{\infty}(\alpha) = R_{FF}^{\infty}(\alpha)$.

Remarquons qu'à chaque instant du déroulement de l'algorithme *Next Fit* on a une seule boite ouverte. Ce qui veut dire qu'il appartient à la classe des algorithmes *à espace borné* caractérisé par un nombre limité et constant de boites ouvertes à tout instant. Ces algorithmes sont définis par une règle de placement et une règle de fermeture des boites ouvertes lorsque le nombre de ces dernières dépasse la borne fixe. Parmi les algorithmes de cette classe, on en trouve le fameux algorithme *Harmonic*$_K$ (H_K) de Lee et Lee [37]

Lee et Lee subdivisent la taille des pièces en K intervalles I_j, $j = 1, \ldots, K$ par :

$$I_j = \begin{cases} \left]\frac{1}{j+1}, \frac{1}{j}\right] & \text{si } 1 \leq j \leq K-1 \\ \left]0, \frac{1}{k}\right] & \text{si } j = K \end{cases}$$

Les pièces du même intervalle seront destinées vers la même catégorie de boites. De cette façon, on a K boites ouvertes à tout instant et lorsqu'une boite ne peut recevoir la pièce en question, on la ferme et on ouvre une nouvelle boite qui sera destinée au même type de pièces que celle qui vient d'être fermée.

Pour K suffisamment grand, Lee et Lee [37] ont montré que $\lim_{K \to \infty} R^{\infty}_{H_K} = 1.69103...$

Plus que ça, ils ont montré qu'aucun algorithme online à espace borné ne peut améliorer le ratio de performance de l'algorithme *Harmonic*$_K$.

Théorème 3.2. [37] Si $\mathcal{A}$ est un algorithme online à espace borné alors $R^{\infty}_{A} \geq 1{,}69103...$

En se basant sur les deux théorèmes précédents on constate que pour franchir la barrière 1.7, il faudrait trouver un algorithme ne faisant partie ni de la classe *Any Fit* ni de celle d'espace borné.

Le premier algorithme qui avait réussi à faire mieux que *First Fit* est *Refined First Fit* (*RFF*) qui est dû à Yao [47]. Il se base sur la subdivision de la taille des pièces en quatre intervalles $\left]0,\frac{1}{3}\right]$, $\left]\frac{1}{3},\frac{2}{5}\right]$, $\left]\frac{2}{5},\frac{1}{2}\right]$, et $\left]\frac{1}{2},1\right]$. Ainsi, quatre types de boites sont définis selon les mêmes intervalles. Cet algorithme procède comme suit :

A tout instant, pour chaque type, une seule boite est ouverte pour recevoir les pièces d'un intervalle donné excepté $\left]\frac{1}{3},\frac{2}{5}\right]$. Pour ce type, une pièce sur six subit un traitement spécial. D'abord, on regarde si cette pièce peut être placée dans l'une des boites déjà ouvertes pour les pièces de l'intervalle $\left]\frac{1}{2},1\right]$. Si cela est le cas, on la met dans cette boite. Sinon, on lui ouvre une nouvelle boite dans l'espoir qu'il y aurait une pièce de taille supérieure à $\frac{1}{2}$ qui puisse être placée avec elle.

Yao avait montré que $R_{RFF}^{\infty}=\frac{5}{3}=1.666...$

En 1985, Lee et Lee [37] avaient développé un nouvel algorithme dénommé *Refined Harmonic*$_K$ (*RH*$_K$), un hybride des algorithmes *Harmonic* et *Refined First Fit*. Il utilise la subdivision de *Harmonic*$_{20}$, et remplace les deux intervalles $\left]\frac{1}{3},\frac{1}{2}\right]$ et $\left]\frac{1}{2},1\right]$ par $\left]\frac{1}{3},y\right]$, $\left]y,\frac{1}{2}\right]$, $\left]\frac{1}{2},1-y\right]$ et $\left]1-y,1\right]$, où $y=\frac{37}{96}$. Il procède comme l'algorithme *Harmonic*$_K$ à l'exception des pièces de l'intervalle $\left]\frac{1}{3},y\right]$. Une de ces pièces sur sept est placée soit dans une boite déjà ouverte pour l'intervalle $\left]\frac{1}{2},1-y\right]$, soit dans une nouvelle boite en espérant lui rajouter une pièce du même intervalle. Lee et Lee ont démontré que $R_{RH_{20}}^{\infty}\leq\frac{373}{228}=1.63596...$

Après, il y a eu plusieurs algorithmes qui ont été proposés par exemple : *Modified Harmonic* (*MH*$_K$) de Ramanan, Brown, Lee, et Lee [42]. En 1991, Richey a présenté son algorithme *Harmonic* + 1 [44]. Mais le meilleur est dû à Steven Seiden publié en 2002 qui s'appelle

Harmonic + + ayant un ratio de performance égal à 1.58889 et il a montré que le ratio de *Harmonic* +1 n'est pas compris entre 1.5874 et 1.588720.

Pour savoir à quel point un algorithme online peut être amélioré en terme de performance, des bornes inférieures sur la meilleure performance possible ont été établies.
La première borne était celle de Yao [47] qui avait montré qu'aucun algorithme online $\mathcal{A}$ pour le problème de placement standard ne peut avoir $R_A^\infty < 1.5$ et la dernière était celle de Van Vliet [44], [45] donnée par le théorème :

Théorème 3. 3. [44], [45] Pour tout algorithme online $\mathcal{A}$, $R_A^\infty \geq 1.540$.

La seule alternative qui reste à présent pour franchir la borne 1.540 est d'abandonner carrément le fait qu'un algorithme soit online.

La première idée qui venait à l'esprit consiste à permettre un certain nombre de déplacements des pièces déjà placées à chaque instant qu'une nouvelle pièce arrive, une chose qui était non tolérée auparavant. Il s'agit d'une nouvelle sorte d'algorithmes, appelée *algorithmes semi-online.* Le meilleur de ces algorithmes est celui de Ivcovic et LIyod [26] ayant un ratio de performance inférieur ou égal à $\frac{5}{4}$.

La seconde idée est une relaxation complète de la contrainte online où toutes les pièces sont connues. Sous ces conditions, on est dans le cas *off-line* et donc les algorithmes auxquels on fait appel sont *off-line.*

3.3.2 Algorithmes Off-line

Examinons cette instance : Soit *L* une liste de pièces formée de 6*N* de chacune des tailles $\frac{1}{7}+\varepsilon, \frac{1}{3}+\varepsilon,$ et $\frac{1}{2}+\varepsilon$ données dans cet ordre. L'algorithme *First Fit* utilise 10*N* boites, alors qu'un algorithme optimal ne nécessite que 6*N* boites comme le montre la figure 3.1

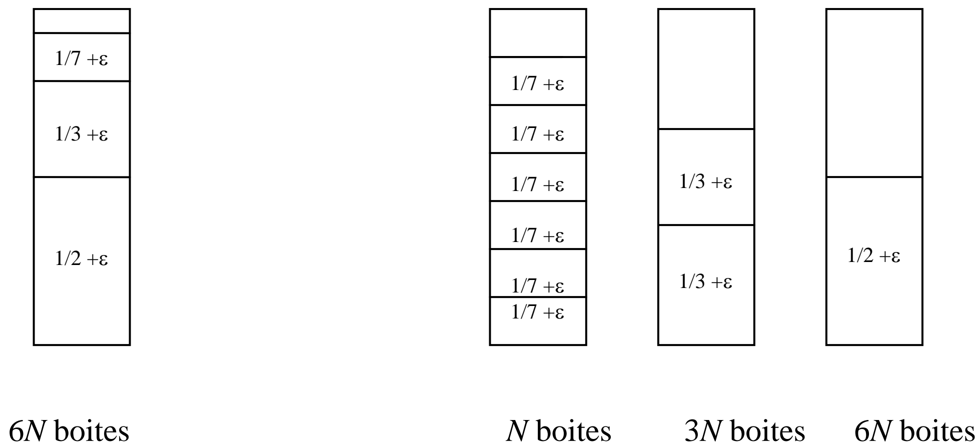


Figure 3.1 : Placement optimal et placement par *FF*

Dans ce cas, le ratio de performance de *First Fit* vaut $\frac{10}{6}=1.666...$ et est presque égal au ratio de performance de *First Fit* dans le pire des cas. Il est clair à travers cette instance, que l'ordre croissant a détérioré la performance de *First Fit*. Par contre le tri décroissant des tailles de ces pièces va l'améliorer. Les plus célèbres des algorithmes qui ont inspiré l'idée du tri décroissant des pièces puis appliquer les règles de placements *First Fit* et *Best Fit* sont respectivement *First Fit Decreasing* (*FFD)* et *Best Fit Decreasing* (*BFD*).

Théorème 3.4. [30] $R_{FFD}^{\infty}=\mathrm{R}_{\mathrm{BFD}}^{\infty}=\frac{11}{9}=1.222...$

Garey et Johnson [15] avaient proposé un algorithme dénommé *Modified First Fit Decreasing* (*MFFD*) qui procède de la même façon que *First Fit Decreasing* à l'exception des pièces dont

la taille est dans l'intervalle $\left]\frac{1}{6},\frac{1}{3}\right]$. Pour ces pièces, l'algorithme examine, d'abord, les boites contenant une seule pièce de taille supérieure à $\frac{1}{2}$ dans l'ordre croissant de leur niveau de remplissage. Ensuite, si les deux plus petites pièces de $\left]\frac{1}{6},\frac{1}{3}\right]$ peuvent être mises dans la première boite examinée, on les met dedans. Sinon, on saute ce traitement spécial et on applique *First Fit*.

Cet algorithme est de complexité $O(n\log n)$ et possède un ratio de performance $R_{MFFD}^{\infty} = \frac{71}{60} = 1.1833...$

Outre les méthodes approchées, on trouve de nombreuses heuristiques et méta-heuristiques qui ont été établies pour résoudre le problème de placement : Falkenauer [9] a décrit un algorithme génétique «*a hybrid grouping genetic algorithm* »(*HGGA*). Très récemment, Fleszar et Hindi [11] ont proposé de nouvelles heuristiques dont la plus efficace est basée sur la méta-heuristique *VNS* [20] et utilise de nouvelles bornes inférieures proposées par Fekete et Schepers [10].

3.4 Conclusion

Dans ce chapitre, nous avons présenté sous forme d'historique, les différentes approches de résolution du problème de placement standard qui existaient déjà dans lé littérature, même si c'était des fois d'une manière concise et dans ce cas, le lecteur peut se rapporter à nos articles de référence et ainsi il pourra nous suivre dans le prochain chapitre, sans aucune ambiguïté, dans lequel nous entamerons le premier thème de notre recherche.

Chapitre 4

Le problème de placement sous la contrainte LIB : version online

4.1 Introduction

Dans ce chapitre, nous nous intéressons au problème de placement sous la contrainte LIB : *Largest Items at the Bottom* qui a été introduit par Manyem [38].

En premier lieu, nous présentons les travaux de Manyem concernant le comportement des deux algorithmes approchés *Next Fit* (*NF*) et *First Fit* (*FF*) pour ce problème.

En dernier lieu, nous évoquons une des perspectives visées par Manyem [38] : *le problème de placement de taille variable sous la contrainte Largest Items at the Bottom*, en étudiant le comportement de l'algorithme *Next Fit using Largest bins only* (*NFL*) et enfin, nous proposons un nouvel algorithme online qu'on dénomme *Improved Best Fit* .

4.2 Présentation du problème

Dans la version *online* du problème de placement standard (de taille uniforme ou variable) *PPS* en abrégé, les pièces d'une liste *L* arrivent une par une. Lorsqu'une pièce *i* de taille a_i arrive, elle doit être immédiatement assignée à une boite (et cette affectation ne peut jamais être changée ultérieurement), et la pièce suivante *i*+1 de taille a_{i+1} n'est connue qu'après placement de la pièce *i*.
Si on impose la condition suivante :
Dans une boite quelconque, pour toute paire de pièces *i* et *j*, si la taille de *j* est supérieure à celle de *i*, alors *j* devrait être placée dans cette boite au dessous de *i*. En d'autres termes, les grandes pièces doivent être placées au fond de la boite, au dessous des petites pièces, on aura ainsi la nouvelle version online du problème de placement sous la contrainte *largest items at the bottom,* en abrégé *PPLIB*.

Nous retrouvons ce problème lors du transport des articles industriels : au cours de chargement d'un camion, si le grand article est placé au dessous du petit article on facilite cette tâche. En termes de poids, une meilleure stabilité sera atteinte par le camion si les articles lourds sont mis au dessous de ceux de poids léger.

4.3 $\mathcal{NP}$-complétude du *PPLIB*

Le problème de placement avec la contrainte *LIB* (*PPLIB*) est $\mathcal{NP}$. Etant donné qu'une proposition d'une solution (affectation des pièces d'une liste *L* aux boites) du problème de reconnaissance est VRAI, on peut vérifier en un temps polynomial que cette affectation utilise au plus *q* boites, qu'aucune boite n'est débordée, et qu'aucune boite ne contient une petite pièce placée au dessous d'une autre qui soit plus grande qu'elle.

Le *PPLIB* est *NP*- complet puisqu'il est équivalent au *PPS* dans le sens où toute solution du premier est une solution du second et si on a une solution du *PPS*, on sait trouver une solution pour le *PPLIB* en effectuant un réarrangement des pièces dans toute boite appartenant à la solution du *PPS.*

4.4 Le problème de placement de taille uniforme sous la contrainte LIB : Version online

On dispose d'un nombre infini de boites de taille unitaire, et d'une liste L de n pièces, $L = \{ i : 1 \leq i \leq n \}$, la taille de chaque pièce i est $a_i \in (0, 1]$.

Chaque pièce doit être placée dans une boite (au dessus des pièces précédemment placées dans cette boite) dès qu'elle arrive. Ce placement ne peut être changé ultérieurement , ce qui exprime le cas online.

Le but est de trouver le nombre de boites minimum pour placer toutes les pièces de la liste L, telles que dans toute boite utilisée, les grandes pièces doivent être placées au dessous des petites pièces (la contrainte *LIB*) et que aucune d'elles ne soit débordée.

La propriété online peut être formulée comme suit :

$$[\text{ La pièce } i \text{ est au dessous de la pièce } j] \Rightarrow [\, i < j \,] . \qquad (1)$$

Similairement, la contrainte *LIB* peut être formulée comme suit :

$$[\text{ La pièce } i \text{ est au dessous de la pièce } j\ ] \Rightarrow [a_i \geq a_j] . \qquad (2)$$

4.4.1 Approches de résolution

On présente une analyse de deux fameux algorithmes online *Next Fit* et *First Fit* adaptés du *PPS* au *PPLIB* de taille uniforme :

4.4.1.1 Next Fit

A chaque itération, l'algorithme *Next Fit* maintient pour la boite courante B_0 deux paramètres *TopSize* et *TotalSize*, représentant la taille de la dernière pièce placée dans B_0 et la somme totale des tailles de pièces contenues dans B_0, respectivement.

Initialement, la boite courante B_0 est vide, $TopSize\ B_0 = 1$ et $TotalSize\ B_0 = 0$. Une boite B_0 est dite *non saturée,* si $TotalSize\ B_0 < 1$.

Pour une pièce i qui arrive, si
$a_i \leq 1 - TotalSize\ B_0$ et $TopSize\ B_0 \geq a_i$, alors
La pièce i est placée dans B_0, $TopSize\ B_0 = a_i$ et $TotalSize\ B_0$ est incrémenté de a_i.

Sinon, B_0 est fermée définitivement, une nouvelle boite est ouverte et sera désormais la boite courante B_0 pour recevoir la pièce i avec $TopSize\ B_0 = a_i$ et $TotalSize\ B_0 = a_i$.

La complexité de *NF* est linéaire en n, le nombre de pièces de la liste L.

Manyem avait montré que le ratio de performance asymptotique au pire des cas de cet algorithme est polynomial (O(n)). Pour le faire, il a considéré la liste suivante de pièces
$L = \{ i : 1 \leq i \leq n\}$, $n \in$ N et les pièces sont de tailles a_i comme suit :

- $a_i = \varepsilon$, si $i = 1 \pmod 4$,
- $a_i = 2\varepsilon$, si $i = 2 \pmod 4$,
- $a_i = 3\varepsilon$, si $i = 3 \pmod 4$, et

• $a_i = 4\varepsilon$, si $i = 0 \pmod 4$.

Avec $0 < \varepsilon \leq 1$ et on suppose que $n\varepsilon \leq 1$.

Comme les pièces arrivent une par une en séquence, l'algorithme *NF* devrait les placer dans des boites en commençant par la boite B_1 comme suit :

B_1 : {1} B_2 : {2} B_3 : {3} B_4 : {4, 5}

B_5 : {6} B_6 : {7} B_7 : {8, 9} (*)

B_8 : {10} B_9 : {11} B_{10} : {12, 13}

……

En supposant que *n* est un multiple de quatre, le nombre de boites dont l'algorithme *NF* a besoin est $\frac{3n}{4} + 1 = \theta(n)$.

Par contre, il existe une solution optimale qui ne requiert que quatre boites :

B_1 : {1, 5, 9, 13,…},

B_2 : {2, 6, 10, 14,…}, (**)

B_3 : {3, 7, 11, 15,…},

B_4 : {4, 8, 12, 16,…}.

Chaque boite B_i $(1 \leq i \leq 4)$ contient $\frac{n}{4}$ pièces, chacune de taille $i\times\varepsilon$. Il s'ensuit :

$$R_{NF}(L) = \frac{NF(L)}{OPT(L)} = \frac{3n+4}{16} = O\ (n).$$

Par conséquent, on a le lemme suivant:

***Lemme 1**. Il existe une instance pour le PPLIB de taille uniforme tel que le ratio de performance asymptotique de l'algorithme NF est d'ordre O(n)*

4.4.1.2 First Fit

L'algorithme *First Fit* (*FF*) appliqué au problème de placement de taille uniforme sous la contrainte *LIB* procède comme suit :

A l'arrivée d'une pièce *i*, on suppose qu'on ait déjà utilisé *m* boites, B_1, B_2, …, B_m . Chaque boite B_j , $1 \leq j \leq m$, possède deux paramètres, *TopSize* et *TotalSize*, représentant la taille de la dernière pièce dans B_j et la somme totale des tailles des pièces contenues dans B_j, respectivement.

L'algorithme *FF* scanne les boites dans cet ordre : de B_1 à B_m. Pour chaque boite B_j, il vérifie si :
i) $a_i \leq$ *TopSize* B_j, et
ii) $a_i \leq 1-$ *TotalSize* B_j.

Il place la pièce *i* dans la première boite B_j qui satisfait les deux conditions précédentes et effectue les mêmes changements que l'algorithme *NF* pour les paramètres *TopSize* B_j et *TotalSize* B_j.
Dans le cas où aucune parmi les *m* boites ne vérifie les conditions *i)* et *ii)*, l'algorithme *FF* ouvre alors, une nouvelle boite B_{m+1}dans laquelle la pièce *i* est mise.

La complexité de cet algorithme est $O(n^2)$.

Pour analyser le comportement de l'algorithme *FF*, Manyem avait utilisé une technique basée sur la distinction de l'ordre d'arrivée des pièces et ceci en trois éventualités :

1- Les tailles sont d'ordre *MND* (croissant).
2- Les tailles sont d'ordre *MNI* (décroissant).
3- L'ordre des tailles n'est ni *MND* ni *MNI.*

♦ Pour le premier cas, Manyem avait constaté que le ratio de performance de l'algorithme *FF* relatif à cette classe d'instances, $R_{FF}(I)$, est égal à un.

♦ Pour le second cas, il a démontré le lemme suivant :

Lemme 2. *Le ratio de performance asymptotique au pire des cas pour la classe d'instances MNI du problème de placement sous la contrainte LIB de taille uniforme est au plus deux.*

♦ Pour le troisième cas, des simulations ont été mises en œuvre sur des listes de pièces dont le nombre *n* prend les valeurs 10, 15, 20, 25, 30 et 35 en effectuant 1000 et 5000 essais. La méthode *branch and bound* a été utilisée pour calculer la solution optimale. Et à partir des comparaisons entre les résultats obtenus, Manyem a noté que la plupart du temps le ratio de performance est inférieur à deux, bien qu'il se peut qu'il existe un ratio supérieur à deux dans certains cas particuliers. Et ainsi, il a énoncé cette conjecture :

Conjecture de Manyem (2002). *Le ratio de performance asymptotique au pire des cas pour FF pour le PPLIB de taille uniforme est au plus deux.*

4.5 Le problème de placement de taille variable sous la Contrainte *LIB*

Dans ce paragraphe, nous traitons la version *taille variable* du problème de placement sous contrainte *LIB* (*PPLIB*) qui représente une occurrence très fréquente sur le plan pratique.

4.5.1 Définition du problème

Nous considérons une liste L de n pièces $L = \{ i : 1 \leq i \leq n \}$, chacune de taille a_i entre 0 et 1, pour tout i dans $\{1,2,\dots,n\}$ et k types de boites $B_1, B_2,\dots, B_k$ de capacités $C(B_1), C(B_2),\dots, C(B_k)$, respectivement, toutes inférieures ou égales à 1. Nous supposons sans perte de généralité que : $C(B_1) > C(B_2) > \dots > C(B_k)$.

Soient :

$\mathcal{A}(L) = \sum_{i=1}^{p} C(B_A^i)$, la capacité totale des boites utilisées par l'algorithme $\mathcal{A}$ pour le placement des pièces de L sous la contrainte *LIB*,

$S(L) = \sum_{i=1}^{n} a_i$, la somme des tailles des pièces de L.

$B(A,L) = (B_A^1, B_A^2, \dots, B_A^p)$, la liste des boites utilisées par l'algorithme $\mathcal{A}$ pour placer les pièces de L sous la contrainte *LIB*,

$B(OPT,L) = (B_{OPT}^1, B_{OPT}^2, \dots, B_{OPT}^m)$, la liste des boites utilisées par l'algorithme optimal *OPT* pour placer les pièces de L sous la contrainte *LIB*,

$OPT(L) = \sum_{i=1}^{m} C(B_{OPT}^i)$, la capacité totale utilisée par un algorithme optimal *OPT* pour le placement de L sous la contrainte *LIB*.

Il s'agit de trouver la plus petite capacité globale des boites utilisées pour placer L sous la contrainte *LIB*

Dans ce cas, le ratio de performance d'un algorithme $\mathcal{A}$ relatif à L, $R_A(L)$, est défini par le rapport de la capacité utilisée par l'algorithme $\mathcal{A}$ sur la capacité optimale :

$$R_A(L) = \frac{\sum_{i=1}^{p} C(B_A^i)}{\sum_{i=1}^{m} C(B_{OPT}^i)} = \frac{A(L)}{OPT(L)}\,.$$

Et le ratio asymptotique dans le pire des cas, R_A^{∞}, prend la même définition précédente.

4.5.2 Approches de résolution

Nous étudions le comportement de l'algorithme *Next Fit using largest bins only* (*NFL*) [12] et ceci à partir d'une simple instance construite par nous-mêmes.
Ensuite, nous présentons un algorithme de résolution du *PPLIB* de taille variable que nous baptisons *Improved Best Fit.*

4.5.2.1 Next Fit using Largest bins only (NFL)

En n'utilisant que les grandes boites (de tailles 1), l'algorithme *NFL* procède de façon analogue à celle de l'algorithme *NF*, c'est-à-dire :

A chaque itération, l'algorithme *NFL* maintient pour la boite courante B_0 deux paramètres *TopSize* et *TotalSize*, représentant la taille de la dernière pièce placée dans B_0 et la somme totale des tailles de pièces contenues dans B_0, respectivement.

Initialement, la boite courante B_0 est vide, *TopSize* $B_0 = 1$ et *TotalSize* $B_0 = 0$. Une boite B_0 est dite *non saturée,* si *TotalSize* $B_0 < 1$.

Pour une pièce i qui arrive, si
$a_i \leq 1 -$ *TotalSize* B_0 et *TopSize* $B_0 \geq a_i$, alors
La pièce i est placée dans B_0, *TopSize* $B_0 = a_i$ et *TotalSize* B_0 est incrémenté par a_i.
Sinon, B_0 est fermée définitivement, une nouvelle boite est ouverte et sera désormais la boite courante B_0 pour recevoir la pièce i avec *TopSize* $B_0 = a_i$ et *TotalSize* $B_0 = a_i$.

La complexité de *NFL* est linéaire en *n*, le nombre de pièces de la liste *L*.

Considérons la liste suivante de pièces *L*, comprenant *2n* pièces de taille $\frac{1}{2n}$ et *2n* pièces de taille $\frac{1}{4n}$ et dont l'ordre d'arrivée est le suivant :

$$L = (\frac{1}{2n}, \frac{1}{4n}, \ldots\ldots., \frac{1}{2n}, \frac{1}{4n}) .$$

On dispose de deux types de boites dont le premier est de capacité 1 et le second est de capacité $\frac{1}{2}$.

La capacité totale des boites utilisées par l'algorithme *NFL* est $2n$ (voir figure 4.1).

2n boites

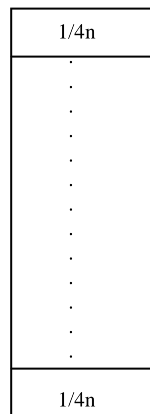
Figure 4.1 : placement de l'algorithme *NFL* de la liste *L*

Et ainsi, on a :

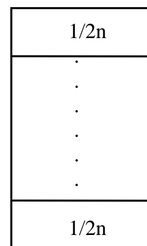
$$NFL(L) = \sum_i C(B^i_{NFL}) = 2n \times 1 = 2n.$$

Un algorithme optimal *OPT* utilise pour placer les pièces de *L*,

- Une boite B_1 de capacité 1 et une boite B_2 de capacité $\frac{1}{2}$ (voir figure 4.2)



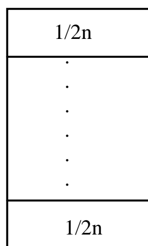
B_1 contenant $2n$ pièces



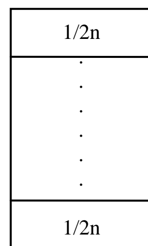
B_2 contenant $2n$ pièces

Figure 4.2 : Placement de la liste L par un algorithme optimal.

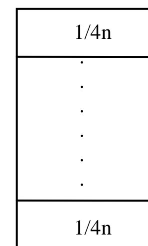
- Soit trois boites B_1, B_2 et B_3 chacune de taille $\frac{1}{2}$ (voir figure 4.3)



B_1 contenant n pièces



B_2 contenant n pièces



B_2 contenant $2n$ pièces

Figure 4.3 : Placement de la liste L par un algorithme optimal.

Et on a bien :

$$OPT(L) = \sum_i C(B^i_{OPT}) = \frac{1}{2} + 1 = \frac{3}{2}.$$

Il s'ensuit que le ratio de performance de *NFL* relatif à l'instance *L*, est

$$R_{NFL}(L)=\frac{4n}{3}.$$

D'où la proposition suivante :

Proposition 4.1. Il existe une instance pour Le problème de placement de taille variable sous la contrainte Largest Items at the Bottom, version online tel que ratio de performance asymptotique de l'algorithme *NFL* est d'ordre *O(n).*

4.5.2.2 Improved Best Fit (*IBF*)

En reprenant l'instance précédente, $L=(\frac{1}{2n},\frac{1}{4n},...,\frac{1}{2n},\frac{1}{4n})$ avec la donnée de deux types de boites B_1 de taille 1 et B_2 de taille $\frac{1}{2}$, nous avons trouvé, en appliquant l'algorithme *NFL,* que $R_{NFL}(L)=\frac{4n}{3}$.

Si on pense à ré- appliquer l'algorithme *NF* mais en n'utilisant que les petites boites B_2 , la taille totale des boites utilisée sera réduite en moitié, c'est-à-dire : $\sum_i C(B^i_{NFS}) = n$.

Les pièces présentes dans cette liste sont de taille petite, alors il est raisonnable d'utiliser les petites boites au lieu des grandes, en dépit du fait que le ratio de performance relatif à cette instance reste polynomial, afin de minimiser l'espace occupé dans ces boites.

Mais d'une manière générale, une idée simple qui vient à l'esprit consiste à affecter la pièce en question à la boite la plus remplie (la règle de placement *Best Fit*), mais bien entendu, en tenant compte de sa taille ainsi des capacités des boites disponibles. A cet effet, nous inspirons cet algorithme que nous l'appelons *Improved Best Fit* (*IBF*) appliqué au problème de placement de taille variable sous la contrainte *LIB* :

Algorithme Improved Best Fit

Etant donnés deux types de boites : $\overline{B_1}$ de capacité 1 et $\overline{B_2}$ de capacité $\frac{1}{2}$. Les pièces arrivent l'une après l'autre.

1- Pour une pièce *i* qui arrive,

Si $a_i > \frac{1}{2}$, alors l'affecter à une nouvelle boite $\overline{B_1}$ sinon aller à 2.

2- Affecter la pièce *i* à la boite $\overline{B_2}$ la plus remplie qui vérifie :

$a_i \le \frac{1}{2}$ - *TotalSize* B_2 et *Topsize* $B_2 \ge a_i$,sinon aller à 3.

3- Affecter la pièce *i* à la boite $\overline{B_1}$ la plus remplie qui vérifie :

$a_i \le 1$- *TotalSize* B_1 et *TopSize* $B_1 \ge a_i$,sinon aller à 4.

4- Affecter la pièce *i* à une nouvelle boite $\overline{B_2}$.

La complexité de l'algorithme *IBF* est $O(n^2)$ puisque chaque pièce *i* subit au maximum 2(*i* - 1) comparaisons et une seule affectation.

4.5.2.3. Ratio de performance de l'algorithme *IBF*

Nous suivons la même technique que Manyem afin de tester la performance de notre algorithme en traitant seulement 2 parmi les 3 cas possibles de l'ordre d'arrivée des pièces :

♦ **L'ordre de tailles est croissant**

Le *IBF* procède comme suit : si la pièce courante *i* est plus grande que celle qui la précède

i -1, alors elle est placée dans sa propre (nouvelle) boite $\overline{B_1}$ ou $\overline{B_2}$ selon sa taille. Autrement, si $a_i = a_{i+1}$, alors la mettre en dessus de i -1 dans cette boite à condition que cette dernière puisse la recevoir. Cet algorithme utilise autant de boites y compris leurs capacités qu'un algorithme optimal car il n'existe pas une autre manière pour placer toutes les pièces de cette liste en utilisant un nombre plus petit de boites. D'où le résultat suivant :

***Lemme* 3** Le ratio asymptotique du pire des cas de l'algorithme *IBF* pour le *PPLIB* de taille variable, dans le cas où les pièces arrivent dans un ordre croissant est un.

- **L'ordre de tailles est décroissant**

Sous cette hypothèse, on peut aisément démontrer que l'algorithme *IBF* garantit une borne de 2 .

Théorème 4.2 $IBF(L) < 2\,OPT(L) + \frac{1}{2}$, pour toute liste L de pièces d'ordre décroissant de tailles .

Preuve. Soit m_1 le nombre de boites de taille 1 utilisées par l'algorithme *IBF* et m_2 le nombre de boites de taille $\frac{1}{2}$ utilisées par *IBF*. Et le nombre total est m.

On désigne par $US(B_i)$, l'espace occupé dans la boite B_i *(Used Space)*.

Pour $1 \le i \le m_1$, $US(B_i) > \frac{1}{2}$ (chaque boite contient une seule pièce de taille supérieure à $\frac{1}{2}$).

Ce qui implique : $\sum_{i=1}^{i=m1} US(B_i) > \frac{1}{2} m_1$.

.Si $m_2 \ge 2$, pour $m_1 + 1 \le i < m_2$, $US(B_i) + US(B_{i+1}) > \frac{1}{2}$ car s'il existe $i_0 \in [\, m_1 + 1, m_2 [$ pour lequel $US(B_{i0}) + US(B_{i0+1}) \le 1/2$, et on aurait dû placer le contenu de la boite B_{i0+1} dans la boite B_{i0} dés le moment où toute pièce de B_{i0+1} est de taille inférieure ou Egale à celles des pièces de la boite B_{i0} (l'ordre est décroissant).

On aura par la suite et ceci par sommation : $\sum_{i=m1+1}^{m} US(B_i) > \frac{m - m_1}{4}$ - 1/4 et

$$IBF\,(L) = m_1 + \frac{1}{2} m_2 = m_1 + \frac{m-m_1}{2} - 1/2 + 1/2 < 2 \sum_{i=1}^{i=m_1} US(B_i) + 2 \sum_{i=m_1+1}^{i=m} US(B_i) + 1/2 =$$

$$2 \sum_{i=1}^{i=m} US(B_i) + 1/2 = 2 \sum_{i=1}^{i=l} US(B^*_i) + 1/2 \leq 2 \sum_{i=1}^{i=m_1} C(B^*_i) + 1/2 = = 2\ OPT\,(L) + 1/2.$$

. Si $m_2 = 1$, on aura :

$$IBF\,(L) = m_1 + \frac{1}{2} < 2 \sum_{i=1}^{i=m_1} US(B_i) + 1/2 < 2 \sum_{i=1}^{i=m} US(B_i) + 1/2 =$$

$$2 \sum_{i=1}^{i=l} US(B^*_i) + 1/2 \leq 2 \sum_{i=1}^{i=m_1} C(B^*_i) + 1/2 = 2\ OPT\,(L) + 1/2.$$

4.6 Conclusion et perspectives

Lors de l'étude des heuristiques *NF* et *NFL* de résolution du problème de placement sous la contrainte *LIB*, version online, l'ordre décroissant des tailles des pièces arrivées constitue un véritable facteur qui déclenche la détérioration du ratio de performance relatif aux instances prises dans cet ordre. Dans ce cas, ces deux algorithmes online se comportent très mal dés le moment où leurs performances sont d'ordre polynomial. Mais mis à part cet ordre, il est très curieux de remarquer l'amélioration apportée par un tri qui lui diffère.

Noua avons présenté un algorithme nommé *Improved Best Fit* et étudié son comportement dans deux cas. Concernant le cas où l'ordre des pièces est aléatoire, il serait intéressant de faire une étude simulatrice.

Un deuxième volet de perspective sera de généraliser cet algorithme au cas où on se permet d'utiliser plusieurs types de boites, voire, la variation de leurs tailles.

Chapitre 5

Le problème de placement avec graphe de conflits (*PPGC*) ayant un seul conflit par boite.

5.1 Introduction

Une nouvelle extension du problème de placement appelée *problème de placement avec graphe de conflits* (*PPGC*) a été introduite par K. Jansen et S. Öhring [29]. Ensuite B. McCloskey et AJ. Shankar [41] se sont intéressés à un cas particulier de ce problème, appelé *problème de placement avec graphe de cliques- conflits* (*PPGCC*).

Nous commençons par décrire le problème (*PPGC*) et exposer les différentes approches de résolution élaborées par K. Jansen et S. Öhring [29], ensuite nous présentons nos résultats à partir de l'étude d'une généralisation du problème posé par B. McCloskey et AJ. Shankar dans lequel on tolère l'existence d'un seul conflit par boite.

Après avoir appliqué les algorithmes usuels *First Fit, Next Fit* et *First Fit Decreasing* à ce problème, nous proposons un algorithme approché basé sur la recherche d'un couplage maximal et la coloration minimale d'un graphe de ratio de performance égal à 3 .

5.2 Présentation du problème (*PPGC*)

Etant données une liste composée de n pièces $L = \{ i : 1 \le i \le n \}$, chacune de taille $a_i \in (0,1]$ et une collection infinie de boites β toutes de même capacité égale à 1.

Certaines pièces de L ne peuvent pas être placées dans une même boite ce qui constitue des *conflits*. Ces conflits sont représentés par les arêtes d'un graphe $G = (V, E)$ dont l'ensemble V désigne les pièces de la liste L. On définit une fonction poids $w : V \rightarrow (0, 1]$, qui représente la taille de chaque pièce de L. Une solution réalisable est une partition des pièces en boites $B_1, \ldots, B_m$, telle que chaque boite B_i contient les pièces $B_{i,1}, \ldots, B_{i,mi}$ vérifiant :

$$\forall i. \qquad \sum_{j=1}^{mi} w(B_{i,j}) \le 1$$

$$\forall i, j, k. \qquad \{B_{i,j}, B_{i,k}\} \notin E$$

Le problème consiste à trouver un nombre minimum de boites pour le placement de toutes les pièces sans avoir aucun conflit par boite et sans excéder la capacité d'aucune de ces boites.

5.3 $\mathcal{NP}$-complétude du *PPGC*

Le problème de placement avec graphe de conflits (*PPGC*) est $\mathcal{NP}$. Etant donnée qu'une proposition d'une solution (affectation des pièces d'une liste L aux boites) du problème de reconnaissance est VRAI. On peut vérifier en un temps polynomial que cette affectation utilise au plus q boites, qu'aucune boite ne soit débordée, et qu'il n'y ait conflit dans aucune boite utilisée.

Une transformation polynomiale du problème de placement standard suffit pour montrer que le *PPGC* est $\mathcal{NP}$–complet. Soit V un ensemble de pièces relatif au problème de placement, la réduction crée un graphe de conflits vide, $G = (V,\emptyset)$, puisque le *PPGC* sans conflits est équivalent au bin packing

.

5.4 Approches de résolution du (*PPGC*)

Nous présentons dans cette section une analyse des approches faites par K. Jansen et S. Öhring [29] sur le problème de placement avec graphe de conflits.

5.4.1 Méthodes d'approximation classiques du problème de placement

Le premier algorithme est défini comme suit :

On place les pièces $i \in V$ prises dans l'ordre d'une liste donnée $L = \{ i : 1 \le i \le n \}$ et ceci en les partitionnant en un ensemble d'indépendants U_1, U_2, …, de sorte que chaque indépendant ne contient aucune une paire de pièces en conflit.

Algorithme 1 (*NF*) : Pour placer une pièce *i*, prendre le plus grand indice *k* pour lequel $U_k \neq \emptyset$. Si $U_k \cup \{i\}$ reste indépendant et $w(i) + \sum_{j \in U_k} w(j) \le 1$ alors placer *i* dans U_k ; sinon placer *i* dans la prochaine boite crée U_{k+1}.

Algorithme 1 (*FF*) : Pour placer une pièce *i*, chercher le plus petit indice *k* tel que $U_k \cup \{i\}$ soit indépendant et que $w(i) + \sum_{j \in U_k} w(j) \le 1$ et placer *i* dans U_k.

Algorithme 1 (*FFD*) : Ranger la liste $L = \{i : 1 \le i \le n\}$ selon l'ordre décroissant des poids et appliquer l'algorithme 1(*FF*) sur la liste résultante.

Pour abréger les algorithmes décrits ci-dessus, on utilise *Alg* 1(*NF*), *Alg* 1(*FF*) et *Alg* 1(*FFD*).

Théorème 5.1. [29] Il existe une instance ($G = (V, E), w$) où $w(i) = \frac{1}{k}$ pour tout $i \in V$ avec ($k = \frac{|V|}{2}$) et une liste $L = \{i : 1 \leq i \leq n\}$ telles que :

$$Alg\ 1(NF)\ (L) = Alg\ 1(FF)\ (L) = Alg\ 1(FFD)\ (L) = O(n).\ OPT(L).$$

5.4.2 Méthode de coloration

Définition 5.1. On appelle *nombre chromatique* d'un graphe G, le plus petit nombre de couleurs nécessaires pour colorier les sommets, de sorte que deux sommets adjacents distincts ne soient pas de même couleur. On le désigne par $\chi(G)$.

K. Jansen et S. Öhring [29] avaient proposé une méthode de résolution du *PPGC* ayant un ratio de performance asymptotique égal à 2,7 pour les graphes qui peuvent être colorés en un temps polynomial.

Dans la première étape, l'algorithme calcule une coloration minimale du graphe de conflits et dans la seconde, il applique une heuristique du problème de placement à chaque ensemble de pièces ayant une même couleur.

Algorithme 2 :

Etape 1 : Déterminer une partition minimale en ensembles indépendants $U_1, \ldots, U_{\chi(G)}$ à partir du graphe de conflits G.

Etape 2 : Appliquer une heuristique du problème de placement parmi (*NF*, *FF*, *FFD*) sur chaque ensemble indépendant U_i, $1 \leq i \leq \chi(G)$.

On note *Alg* 2(*NF*), *Alg* 2(*FF*) et *Alg* 2(*FFD*) les algorithmes composés correspondant à chacune des trois heuristiques. On a le théorème suivant :

Théorème 5.2. [29] Le ratio de performance asymptotique au pire des cas de l'algorithme $Alg\,2(NF)$ pour le *PPGC*, $R^{\infty}_{Alg\,2(NF)}$, est égal à 3 .

Ce ratio s'améliore pour les graphes bipartis et devient 2 comme nous le montre le théorème :

Théorème 5.3. [29] Si $G = (V, E)$ est un graphe biparti, alors $R^{\infty}_{Alg\,2(NF)} = 2$.

L'algorithme $Alg\,2(FF)$ est meilleur que $Alg\,2(NF)$.

Théorème 5.4. [29] Le ratio de performance asymptotique au pire des cas de l'algorithme $Alg\,2(FF)$ pour le *PPGC*, $R^{\infty}_{Alg\,2(FF)}$, est égal à 2,7 .

Pour démontrer ce résultat, K. Jansen et S. Öhring ont utilisé un résultat d'une certaine fonction poids W [13] appliqué sur chaque classe de couleur, ils ont abouti à avoir le ratio de performance asymptotique au pire des cas de $Alg\,2(FFD)$ variant entre 2.691 et 2.7.

5.4.3 Méthode de pré-coloration

L'ensemble des pièces de grande taille est noté $\overline{V} = \{v \in V / \, w(v) > \frac{1}{2}\}$. On va analyser une méthode de résolution du *BPGC* lorsque les éléments de $\overline{V}$ sont colorés différemment. On note $\chi_{\overline{V}}(G)$ le nombre minimum de couleurs dans une coloration de G avec cette propriété. Ce problème est connu dans la littérature par *le problème d'extension de pré-coloration* (*precoloring extension problem*), $1 - PrExt$.

Définition 5.2. Etant donné un graphe $G = (V, E)$ et k sommets distincts $v_1, \ldots, v_k$, le problème $1 - PrExt$ consiste à trouver une coloration minimale f de G telle que $f(v_i) = i$ pour $1 \le i \le k$.

Ce problème est résolu polynomialement pour les graphes d'intervalles, les forêts, les graphes scindés, les complémentaires des graphes bipartis, les cographes, les *K*-arbres partiels et les complémentaires des graphes de Meyniel [2, 3, 23, 24, 28, 36]. Pour avoir les définitions de ces graphes voir [4, 18].

Algorithme 3 :

Etape 1 : Soient les pièces de V ayant les poids $w(v) > \frac{1}{2}$.

Etape 2 : Déterminer une partition minimale en ensembles indépendants $U_1, \ldots, U_{\chi_{\overline{V}}(G)}$ du graphe de conflits G telle que $\left|U_i \cap \overline{V}\right| \leq 1$ pour tout $1 \leq i \leq \chi_{\overline{V}}(G)$.

Etape 3 : Appliquer une heuristique du problème de placement parmi (*NF*, *FF*, *FFD*) sur chaque ensemble indépendant U_i, $1 \leq i \leq \chi_{\overline{V}}(G)$.

On note, comme précédemment *Alg* 3(*NF*), *Alg* 3(*FF*) et *Alg* 3(*FFD*) les algorithmes composés correspondant à chacune des trois heuristiques. On applique ces algorithmes sur les classes de graphes pour lesquelles le problème $1 - PrExt$ est résolu polynomialement et on a :

Théorème 5.5. [29] Le ratio de performance asymptotique au pire des cas de l'algorithme *Alg* 3(*FF*) pour le *PPGC*, $R^{\infty}_{Alg3(FF)}$, est égal à 2.5 .

Corollaire 5.1. [29] L'algorithme *Alg* 3(*FF*) est polynomial et possède un ratio de performance asymptotique au pire des cas, $R^{\infty}_{Alg3(FF)}$, égal à 2.5 pour les classes de graphes suivantes :

les graphes d'intervalle, les forets, les graphes scindés, les compléments des graphes bipartis, les cographes, les *K*-arbres partiels et les complémentaires des graphes de Meyniel.

Parmi les thèmes de réflexion posés par K. Jansen et S. Öhring, on trouve celui de la recherche d'algorithmes approchés pour des classes spéciales de graphes. Ce qui a stimulé B. McCloskey et AJ. Shankar à prendre cette voie en traitant les graphes où les conflits constituent des cliques et ont établi de nombreux résultats et ont posé le problème de placement avec graphe de conflits dans lequel on permet l'existence d'un nombre constant de conflits par boite. Nous définissons le *problème de placement avec graphe de conflits ayant un seul conflit par boite*, *PPGC*.

5.5 Présentation du problème (*PPGC*) en tolérant un seul conflit par boite

Etant données une liste composée de n pièces $L = \{i : 1 \leq i \leq n\}$, chacune de taille $a_i \in (0,1]$ pour tout i et une collection infinie de boites β toutes de même capacité égale à 1. Certaines pièces de L ne peuvent pas être regroupées et placées dans une même boite. Ces conflits sont représentés par des arêtes d'un graphe $G = (V, E)$ dont l'ensemble V désigne les pièces de la liste L.

Il s'agit de trouver une partition minimale des pièces en boites sans dépasser la capacité d'aucune d'elles et en tolérant l'existence d'un seul conflit par boite.

5.6 $\mathcal{NP}$-complétude du *PPGC* avec un seul conflit

Le problème de placement avec graphe de conflits (*PPGC*) est $\mathcal{NP}$. Etant donnée qu'une proposition d'une solution (affectation des pièces d'une liste L aux boites) du problème de reconnaissance est VRAI. On peut vérifier en un temps polynomial que cette affectation utilise au plus q boites, qu'aucune boite n'est de capacité dépassée, et qu'il y a un seul conflit par boite.

Une transformation polynomiale du problème de *PPGC* suffit pour montrer que le *PPGC* avec un seul conflit est $\mathcal{NP}$–complet. Soit $G = (V, E)$ le graphe de conflits relatif au problème de placement dans lequel on permet l'existence d'un seul conflit par boite et désignons par $S \subset E$ l'ensemble des couples de ces pièces tolérées, la réduction crée un graphe de conflits $G = (V,E)$ avec $S = \emptyset$, dés que *PPGC* sans l'existence d'aucun conflit par boite est équivalent au *PPGC* qui est un problème $\mathcal{NP}$–complet.

5.7 Approches de résolution du *PPGC* avec un seul conflit

Nous allons étudier ce nouveau problème, en commençant par analyser le comportement des trois algorithmes usuels du problème de placement standard, ensuite nous proposons une nouvelle méthode approchée qui garantit un ratio de performance asymptotique égal à 3.

5.7.1 Méthodes classiques du problème de placement

Nous effectuons une petite modification de l'algorithme précédent, *Algorithme* 1 afin de l'adapter à notre nouveau problème et qu'on le notera *Algorithme* 1' :
On place les pièces $i \in V$ prises dans l'ordre d'une liste donnée $L = \{i : 1 \le i \le n\}$ dans des boites indexées $B_1, B_2, \ldots,$

Algorithme 1' (*NF*) : Pour placer une pièce i, prendre le plus grand indice k pour lequel $B_k \neq \emptyset$. Si $B_k \cup \{i\}$ contient au plus un conflit et $w(i) + \sum_{j \in B_k} w(j) \le 1$ alors placer i dans B_k ; sinon placer i dans la prochaine boite vide B_{k+1}.

Algorithme 1' (*FF*) : Pour placer une pièce i, chercher le plus petit indice k tel que $B_k \cup \{i\}$ contient au plus un conflit et que $w(i) + \sum_{j \in B_k} w(j) \le 1$ et placer i dans B_k.

Algorithme 1' (*FFD*) : Ranger la liste $L = \{i : 1 \leq i \leq n\}$ selon l'ordre décroissant des poids et appliquer l'algorithme 1'(*FF*) sur la liste résultante.

Pour abréger les algorithmes décrits ci-dessus, on utilise *Alg* 1'(*NF*), *Alg* 1'(*FF*) et *Alg* 1'(*FFD*). Malheureusement, ces algorithmes se comportent mal comme l'annonce cette proposition :

Proposition 5.1. Il existe une instance $(G = (V, E), w)$ et une liste $L = \{i : 1 \leq i \leq n\}$ telles que :

$$Alg\ 1'(NF)\ (L) = Alg\ 1'(FF)\ (L) = Alg\ 1'(FFD)\ (L) = O(n).\ OPT(L).$$

Preuve .Considérons la liste de pièces suivantes $L = \{a_1, \ldots, a_{3k}\} \cup \{b_1, \ldots, b_{3k}\}$ de cardinalité $n = 6k$, tels que $s(a_i) = s(b_i) = \frac{1}{3k}$, $\forall\ i \in \{1, \ldots, 3k\}$. Soit $G = (V, E)$ un graphe de conflits, avec

$$E = \{a_i b_j \,/\, 1 \leq i = j \leq 3k\}.$$

En prenant la liste L dans cet ordre : $L = (a_1, b_1, a_2, b_2, \ldots, a_{3k}, b_{3k})$ et en appliquant l'algorithme *NF*, on aura bien cette partition :

$\{a_1, b_1, a_2\}$, $\{b_2, a_3, b_3\}$, . . ., $\{b_{3k-1}, a_{3k}, b_{3k}\}$ en $3k$ boites et chacune d'elle contient un et un seul conflit.

En revanche, un algorithme exact utilise uniquement 2 boites selon la partition :
$\{a_1, a_2, \ldots, a_{3k-2}, a_{3k}, b_{3k}\}$, $\{b_1, b_2, \ldots, b_{3k-1}, a_{3k-1})$ avec deux conflits, un dans chacune d'elles ou bien $\{a_1, a_2, \ldots, a_{3k}\}$, $\{b_1, b_2, \ldots, b_{3k}\}$.

D'où : $Alg\ 1'(NF)\ (L) = 3k = \frac{n}{2} = \frac{n}{4}.2$. Ceci reste vrai pour *Alg* 1'(*FF*) et *Alg* 1'(*FFD*).

5.7.2 Méthode de couplage maximal et précoloration

Nous suggérons une approche du problème de placement avec graphe de conflits en tolérant un conflit par boite basée sur la recherche de couplage de cardinal maximum et ceci pour déterminer une borne inférieure au nombre de boites utilisées qui va être dans la suite minimisé. Ensuite nous procédons à une précoloration minimale après avoir contracté le graphe de conflits initial $G = (V, E)$. Nous appliquons enfin, une des heuristiques classiques du problème de placement standard.

Motivation

Données :

- Une liste $L = \{1, 2,\ldots, n\}$ de projets, chacun est de taille $s(i) \in (0, 1]$ qui représente l'enveloppe budgétaire nécessaire.

- Une collection de petites compagnies récemment crées chacune d'elles ayant un capital $C = 1$.

- Un graphe de conflits associé à la liste L, $G = (V,E)$, tels que V désigne la liste L et deux arêtes joignant deux projets si ces derniers sont difficiles à être réalisés simultanément par une compagnie parmi L ce qui nécessite une intervention externe en matière d'expérience (compagnie étrangère). Sachant qu'une collaboration se fait avec une seule petite compagnie et couvre exactement deux projets.

Objectif :

- On cherche à minimiser le nombre de compagnies qui investissent sur l'ensemble de tous les projets L après avoir déterminé une borne inférieure concernant le nombre de collaborations (pour permettre au plus grand nombre possible des petites compagnies de bénéficier de l'expérience externe)..

Algorithme 3’

Soient L une liste de pièces et $G = (V, E)$ le graphe de conflits associé au problème de placement de L.

Etape 1 : Trouver un couplage maximum C_{max}, dans le graphe de conflit $G = (V, E)$ de cardinal $q=|C_{max}|$.

Etape 2 : Construire le graphe $G' = (V', E')$, contracté du graphe $G = (V, E)$ obtenu en contractant chaque paire d’extrémités d’une arête de C_{max} en un seul sommet dans G'.

Etape 3 : Dans le graphe G', affecter une nouvelle couleur à chaque sommet résultant d’une opération de contraction. Désignons par $\overline{V}'$ l’ensemble de sommets colorés.

Etape 4 : Déterminer une partition minimale en ensembles $V_1, \ldots, V_{\chi_{\overline{V}'}(G')}$ du graphe de conflits G' tel que $|V_i \cap \overline{V}'| \leq 1$ pour tout $1 \leq i \leq \chi_{\overline{V}'}(G')$.

Etape 5 : Appliquer une heuristique du problème de placement parmi (*NF*, *FF*, *FFD*) sur chaque ensemble indépendant V_i, $1 \leq i \leq \chi_{\overline{V}'}(G')$.

Soient *Alg* 3(*NF*), *Alg* 3(*FF*) et *Alg* 3(*FFD*) les algorithmes composés correspondant à chacune des trois heuristiques. On applique ces algorithmes sur les classes de graphes dont le problème 1 – *PrExt* est résolu polynomialement .

Théorème 5.7. Le ratio de performance asymptotique au pire des cas de l’algorithme *Alg*’ 3(*NF*) pour le *PPGC* avec un seul conflit par boite, $R^{\infty}_{Alg'3(NF)}$, est égal à 3 .

Preuve. Soit $q=|C_{max}|$ après avoir appliqué la première étape de l'algorithme.

Considérons la partition en sous ensembles du graphe *G*', $V_1, \ldots, V_{\chi_{\overline{V'}}(G')}$, où $\chi_{\overline{V'}}(G')$ représente le nombre minimum de couleurs dans une coloration de *G*'précédée par la coloration des sommets de $\overline{V}'$.

Notons n_i, le nombre de boites généré par l'algorithme *NF* sur V_i, $\forall\, i = 1, \ldots \chi_{\overline{V'}}(G')$

Comme $\chi_{\overline{V'}}(G') \geq q=|C_{max}|$, chaque ensemble V_i possède exactement une boite contenant un conflit, $\forall\, i = 1, \ldots, q$,

Notons : $B_1^{V_1}, B_1^{V_2}, \ldots, B_1^{V_{q1}}$. ces boites.

En appliquant l'algorithme *NF* sur chaque ensemble V_i, la somme du contenu de chaque paire de boites consécutives est supérieure à 1.

$$\forall\, i \in \{1, \ldots, q\}, \qquad \sum_{j=1}^{n_i} C(B_j^{V_i}) > \frac{n_i - 1}{2} \qquad (1)$$

$$\Rightarrow \qquad \sum_{i=1}^{q} \sum_{j=1}^{n_i} C(B_j^{V_i}) > \frac{\sum_{i=1}^{q} n_i - q}{2} \qquad (2)$$

De même pour $i \in \{q+1, \ldots, \chi_{\overline{V'}}(G')\}$, on a :

$$\sum_{i=q+1}^{\chi(G')_{V'}} \sum_{j=1}^{n_i} C(B_j^{V_i}) > \frac{\sum_{i=q+1}^{\chi(G)} n_i - (\chi_{V'}(G') - q)}{2} \qquad (3)$$

- Si $\chi_{\overline{V'}}(G') = q$ et $\forall\, i = 1, \ldots, q$, $n_i = 1$, notre algorithme fournit une solution optimale.

- Sinon, en additionnant (2) et (3) on aura :

$$\sum_{i=1}^{\chi(G')_{V'}} \sum_{j=1}^{n_i} C(B_j^{V_{i_i}}) = \sum_{i=1}^{n} s(i) > \frac{\sum_{i=1}^{\chi_{V'}(G')} n_i - \chi_{V'}(G')}{2}$$

$\Rightarrow$ $$OPT(L) > \frac{\sum_{i=1}^{\chi_{V'}(G')} n_i - \chi_{V'}(G')}{2}$$, car $OPT(L) \geq \sum_{i=1}^{n} s(i)$.

$\Rightarrow$ $$OPT(L) > \frac{Alg'3(NF(L)) - \chi_{V'}(G')}{2}$$, car $\sum_{i=1}^{\chi(G')} n_i = Alg'3(NF(L))$.

$\Rightarrow$ $Alg'\,3(NF(L)) \leq 3\ OPT(L)$. , car $\chi_{\overline{V'}}(G') \leq OPT(L)$.

Il nous reste à montrer que cette borne est atteinte asymptotiquement :

Soient l un entier naturel pair et une liste de pièces, $L = C_1 \cup C_2 \cup U_1 \cup U_2$, tels que :

$C_1 = \{c_{1,1}, \ldots, c_{1,\frac{l}{2}}\}$, $C_2 = \{c_{2,1}, \ldots, c_{2,\frac{l}{2}}\}$, $U_1 = \{u_{1,1}, \ldots, u_{1,l}\}$ et $U_2 = \{u_{2,1}, \ldots, u_{2,l}\}$.

Leurs tailles sont données par : $s(c) = \frac{\delta}{2}$, $c \in C_1, C_2$; $s(u_{1,j}) = 3\delta$ et $s(u_{2,j}) = \frac{1}{2} - \delta$ avec

$\delta = \frac{1}{3l}$.

Soit l'ensemble des arêtes du graphe de conflits défini par :

$E = \{(c, c') / c, c' \in C_1, c \neq c'\} \cup \{(c_i, c_j) / c_i \in C_1, c_j \in C_2 \text{ pour } i = j\}$

- En appliquant l'algorithme $Alg'\,3(NF)$ sur cette instance, on aura :

- $C_{max} = \{(c_i, c_j) / c_i \in C_1, c_j \in C_2 \text{ pour } i = j\}$ de cardinal $\frac{l}{2}$.

- Une partition minimale du graphe *G'* est :

$$\{c_{1,1}, c_{2,1}\},\ldots,\{c_{1,\frac{l}{2}}, c_{\frac{l}{2}}, u_{2,1}, u_{1,1},\ldots,u_{2,l}, u_{1,l}\} \Rightarrow \chi_{\overline{V'}}(G') = \frac{l}{2}.$$

- L'application de l'algorithme *NF* sur chaque ensemble de la partition précédente, crée la partition en boites :

$$\{c_{1,1}, c_{2,1}\},\ldots,\{c_{1,\frac{l}{2}}, c_{2,\frac{l}{2}}, u_{2,1}, u_{1,1}\}\{u_{2,2}, u_{1,2}\},\ldots,\{u_{2,l}, u_{1,l}\} \text{ de cardial } \frac{l}{2}+(l-1).$$

- Une solution optimale est composée de la partition :

$$\{u_{2,1}, u_{2,2}, c_{1,1}, c_{2,1}\},\ldots,\{u_{2,l-1}, u_{2,l}, c_{1,\frac{l}{2}}, c_{2,\frac{l}{2}}\}\{u_{1,1},\ldots,u_{1,l}\} \text{ de cardinal } \frac{l}{2}+1.$$

Il s'en suit alors, $\lim_{l\to\infty} \dfrac{\frac{l}{2}+(l-1)}{\frac{l}{2}+1} = 3$.

5.8 Conclusion et perspectives

Dans ce chapitre, nous nous sommes fixés pour objectif la résolution du problème de placement avec graphe de conflits en autorisant l'existence d'un seul conflit par boite, ce qui constitue un cas particulier de la perspective proposée par B. McCloskey et AJ. Shankar [41].

L'algorithme que nous avons développé est basé sur des idées simples inspirées de la théorie des graphes et hybridées avec des heuristiques du problème du Bin packing. Ceci ne permet pas de clore la porte de la recherche d'autres heuristiques de meilleur ratio de performance.

Il serait très intéressant de faire une étude du *PPGC* en tolérant *k* conflits par boite et de traiter des cas où le graphe de conflit fait partie de certaines classes particulières de graphes.

Conclusion générale

Que faut-il retenir de la Théorie de la Complexité ? Essentiellement qu'il existe des problèmes d'optimisation combinatoire qu'il ne faut pas s'acharner à résoudre optimalement.

En effet, si notre problème est $\mathcal{NP}$-complet, il vaut mieux abandonner l'espoir de trouver un algorithme polynomial. Si on en trouve un, ce qui est peu probable, on deviendrait célèbre (à défaut de devenir riche) : on aurait infirmé ainsi la conjecture $\mathcal{P} \neq \mathcal{NP}$!

Le thème de recherche auquel nous sommes intéressés se trouve effectivement parmi les problèmes les plus difficiles de l'optimisation combinatoire puisqu'il fait partie de la classe des problèmes dit $\mathcal{NP}$–complets.

Il s'agit du *problème de placement unidimensionnel* qui s'énonce d'une manière très simple: Il s'agit de placer n objets d'une liste L chacun de taille a_i, $i = 1, \ldots, n$ dans un nombre minimum de boites, toutes de capacité égale à 1.

Le problème de placement puise son rôle dans le champs d'application en modélisant de véritables problèmes rencontrés dans l'industrie ce qui lui en fait un centre d'attraction de plusieurs chercheurs depuis longtemps.

Dans cette thèse, nous avons étalé les différentes facettes relatives à ce problème en commençant par cerner le cadre théorique dans lequel il est inscrit.

L'essentiel de la thèse s'articulait principalement autour des deux derniers chapitres dans lesquels nous avons présenté notre contribution personnelle.

Au quatrième chapitre, nous avons traité une variante du problème de placement due à Manyem et dénommée « *problème de placement sous la contrainte LIB* » : *Largest Items at the Bottom* . Cette contrainte est rencontrée sur le plan pratique lors du chargement de véhicules par des articles de différentes tailles, tout pour garantir une meilleure stabilité au cours de leur transport, il y a lieu de placer toujours les grands articles au dessous des petits.

Le *problème de placement de taille variable sous la contrainte Largest Items at the Bottom* a fait l'objet de notre étude. Nous avons analysé les résultats des comportements de l'algorithme *Next Fit using Largest bins only* et de l'algorithme *Improved Best Fit* que nous avons proposé.

Dans le dernier chapitre, nous sommes intéressés à une extension du problème de placement dite « *problème de placement avec graphe de conflits* » (*PPGC*). Après avoir rappelé les travaux et résultats établis concernant ce problème, nous avons traité le cas où on permet un seul conflit par boite.

Nous avons réussi à développer un algorithme approché qui se base sur la recherche d'un couplage maximum, puis sur une coloration minimale du graphe obtenu à partir du graphe initial en effectuant un certain nombre de contractions de sommets et enfin sur l'application d'une règle de placement sur chaque classe de sommets de même couleur.

Comme perspectives, nous proposons d'étudier les problèmes suivants :

1- Le problème de placement de taille variable sous la contrainte Longest Items at the Bottom dans lequel on utilise plusieurs types de boites voire la variation de leurs tailles.
2- Déterminer le ratio de performance au pire cas de l'algorithme *Improved Best Fit* dans le cas où l'ordre d'arrivée des pièces est aléatoire.
3- Le problème de placement avec graphe de conflits ayant k conflits par boite, $k > 1$.
4- Le problème de placement avec graphe de conflits dans lequel le graphe fait partie d'une classe particulière de graphes.

Bibliographie

[1] S. Benabderrahmane ; Problème de placement. Approches de résolution : Thèse de Magister, 2001.

[2] M. Biro, M. Hujter and Z. Tuza ; Precoloring extension. I. Interval graphs, Disc. Math., 100 ; pp. 267-279 ; 1991.

[3] H. L. Bodlaender , K. Jansen and G. J. Woeginger ; Scheduling with incompatible jobs, Workshop Graph Theoretical Concepts in Computer Science, LNCS 657 ; pp. 37-49 ; 1992.

[4] A. Brandstadt ; Special Graph Classes , a survey. Report SM-DU-199, Universitat-Gesamthochschule Duisburg ; 1991.

[5] L. M. A. Chan, D. S. Levi and J. Bramel ; Worst case analysis, linear programming and the bin packing problem ; Mathematical Programming 83 ; pp. 213-227 ; 1998.

[6] S. A. Cook ; The complexity of theorem-proving procedures ; Proc.3rd Ann. ACM Symp. On Theory of Computing ; Association for Computing Machinery ; New York; pp. 151-158 ; 1971.

[7] J.Edmonds ; Paths, trees and flowers ; Cand. J. Math. 17 ; pp.449-467 ; 1965.

[8] S. Eilon and N. Christofides ; The loading problem. Management Science 17, pp. 259-267 ; 1971.

[9] E. Falkenaur ; A hybrid grouping genetic algorithm for bin packing ; Journal of Heuristics 2 ; pp. 5-30 ; 1996.

[10] S. P. Fekete and J. Schepers ; New classes of fast lower bounds for bin packing problems . Mathematical Programming 91 , 1 ; pp. 11-31 ; 2001.

[11] K. Fleszar and K. Hindi ; New heuristics for one-dimensional bin packing; Computers and Operations Research 29 , 7 ; pp. 821-839 ; 2002.

[12] D. K. Friesen and M. A. Langston ; Variable sized bin packing ; SIAM J. Compute., 15 pp. 222-230 ; 1986.

[13] M. R. Garey, R. L. Graham, D. S. Johnson and A. C. Yao ; Ressource constrained scheduling as generalized bin packing ; J. Comb. Th. Ser. A, 21 ; pp. 257-298 ; 1976.

[14] M. R. Garey and D.S.Johnson ; Computers and intractability : A Guide to the theory of NP-Completeness. W.H.Freeman and CO., San Francisco, 1979.

[15] M. R. Garey and D. S. Johnson ; A $\frac{71}{60}$ theorm for bin packing ; J. of Complexity 1 ; pp. 65-106 ; 1985.

[16] F. Glover, Tabu search. Part I, ORSA Journal on Computing, vol 1, pp. 190-206, 1989.

[17] D. E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison Wesley, 1989.

[18] M. C. Golumbic ; Algorithmic Graph Theory and Perfect Graphs, Academic Press, London ; 1980.

[19] G. Gutin, T. Jensen and A. Yeo ; Batched Bin Packing , www.cs.rhul.ac.uk/home/gutin/paperstsp/bbpp5.pdf.

[20] P. Hansen and N. Mladenovic ; An introduction to variable neighbourhood search ; in Metaheuristics : Advances and Trends in Local Search Procedures for Optimization (S. Voss, S. Martello, I. H. Osman and C. Roucairol, eds), pp. 433-458, Kluwer ; 1999.

[21] M. Hifi and R. M'Hallah ; A best-local position procedure-based heuristic for the two-dimensional layout problem ; http://mse.univ-paris1.fr/Cahiers2001/2001070B.htm.

[22] D. S. Hochbaum ; Various notions of appromixation : Good, Better, Best, and more ; in D. S. Hochbaum (Ed), Approximationalgorithms for NP-Hard problems, PWS publishing ; Boston ; pp. 346-395 ; 1997.

[23] M. Hujter and Z. Tuza ; Precoloring extension. II. Graph classes related to bipartite graphs, Acta Math. Univ. Comenianae , 61 ; 1993.

[24] M. Hujter and Z. Tuza ; Precoloring extension. III. Classes of perfect graphs, unpublished manuscript ; 1993.

[25] M. S. Hung and J. R. Brown ; An algorithm for a class of loading problems. Naval Research Logistics Quarterly 24, pp. 571-579 ; 1978.

[26] Z. Ivkovic and E. LIyold; Fully dynamic algorithms for bin packing : being myopic helps ; In proccedings of First European Symposium on algorithms, N° 726 in Lecture Notes in Computer Science ; pp. 224-235 ; New York ; 1993.

[27] K. Jansen and H. Zhang ; On rectangle packing: maximizing benefits, http://portal.acm.org/citation.cfm.?id=982822

[28] K. Jansen and P. Scheffler ; Generalized coloring for tree-like graphs, Workshop Graph Theoritical Concepts in Computer Science, LNCS 657 ; pp. 50-59 ; 1992.

[29] K. Jansen and S. R. Öhring ; Approximation algorithms for time constrained schaduling. In Workshop on Parallel Algorithms for Irregularly Structured Problems, pp. 143-157 , 1995.

[30] D. S. Johnson ; Near-optimal Bin Packing algorithms. PhD thesis, Massachusetts Institute of Thecnology, Departement of Mathematics, Cambridge ; 1973.

[31] D. S. Johnson ; Fast algorithms for bin packing ; Journal of Computer and System Sciences 8 ; pp. 272-314 ; 1974.

[32] D. S. Johnson ; The NP-Completeness column : An ongoing guide. J. Algorithms, 3 ; pp. 288-300 ; 1982.

[33] R. M. Karp ; Reducibility among Combinatorial problems ; in R.E.Miller and J.W.Thatcher (eds.) ; complexity of Computer Computations ; Plenum Press ; New York ; pp. 85-103 ; 1972.

[34] D.G. Kirkptrick, Gellat, Vecchi ; Optimization by simulated annealing, Science, 220, pp 671-680, 1983.

[35] J. Kratochvil ; Precoloring extension with fixed color bound, Acta Math. Univ. Comenianae, 62 ; pp. 139-153 ; 1993.

[36] E. L. Lawler ; Compinatorial Optimization : Networks and Matroids, Holt, Rinehart and Winston, New York 1976.

[37] C. C. Lee and D. T. Lee ; A simple online packing algorithm ; J. ACM, 32, pp. 562-572 ; 1985.

[38] P.Manyem ; Bin packing and covering with longets items at the bottom : online version; ANZIAM J, 43. 2001/2002. http://amzianj.austms.org.au/V43/E044.

[40] S. Martello and P. Toth ; An exact algorithm for the bin packing problem. Presented at EURO X, Beograd ; 1989.

[41] S. Martello and P. Toth ; Knapsack problems, Algorithms and Computer Implementations. John Wiley and Sons Ltd, England, 1990.

[42] B. McCloskey et A. J. Shankar ; Approaches to Bin Packing with Clique-Graph Conflicts , March 2004 ; www.cs.berkeley.edu/~billm/paper-cdmix.pdf.

[43] P. Ramanan , D. J. Brown. C. C. Lee and D. T. Lee. On-line bin packing in linear time, J. Algorithms. 10 ; pp. 305-326 ; 1989.

[44] M. B. Richey ; Improved bounds for harmonic-based bin packing algorithms. Disc. Appl. Math., 34 ; pp. 203-227 ; 1991.

[45] A. Van Vliet ; Lower and Upper bounds for on-line Bin Packing and Scheduling Heuristic. PhD thesis, Erasmus University, Rotterdam, Netherlands ; 1995.

[46] A. Van Vliet ; On the asymptotic worst case behaviour of harmonic fit. J. Algorithms, . 20 ; pp. 113-136 ; 1996.

[47] D. J. A. Welsh ; Matroid Theory ; Academic Press, London, New York, San Francisco; 1975.

[48] A. C. Yao ; New algorithms for bin packing. J. Assoc. Comput. Mach., 27 ; pp. 207-227 ; 1980.