

Rangement d'objets multiboîtes : modèles et algorithmes

Pierre Lemaire

▶ **To cite this version:**

Pierre Lemaire. Rangement d'objets multiboîtes : modèles et algorithmes. Autre [cs.OH]. Université Joseph-Fourier - Grenoble I, 2004. Français. NNT : . tel-00006893v2

HAL Id: tel-00006893

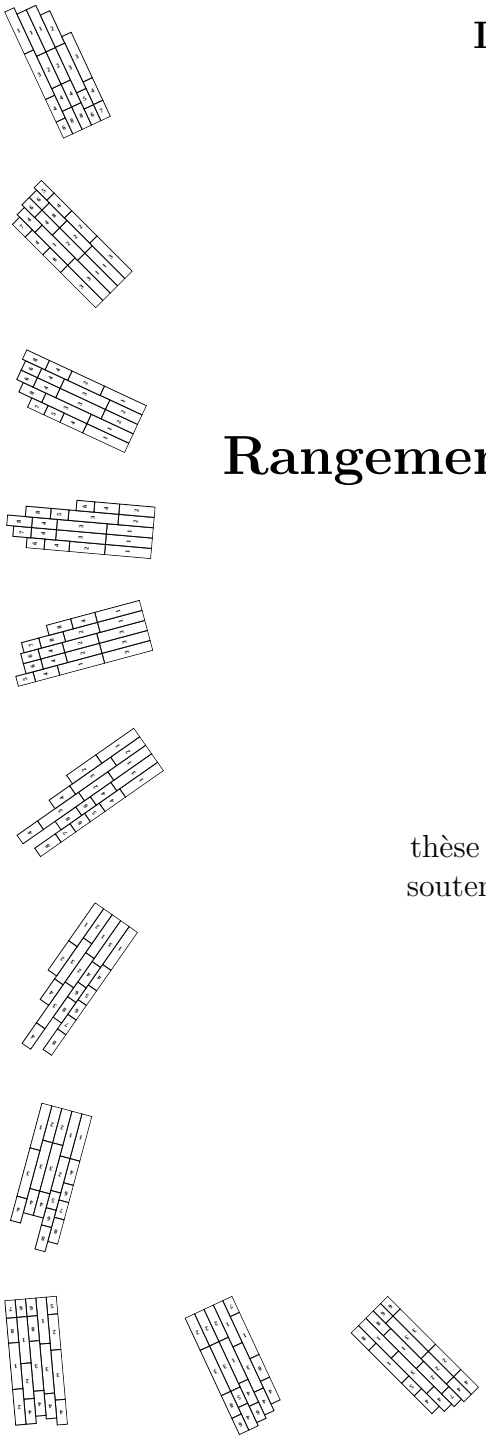
https://theses.hal.science/tel-00006893v2

Submitted on 23 Sep 2004

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

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

Université Joseph Fourier (Grenoble 1)
École doctorale « Mathématiques, Informatique, Sciences et Technologies de l'Information » (ED 0217)



Doctorat (spécialité : informatique)

Pierre Lemaire

Rangement d'objets multiboîtes

modèles et algorithmes

thèse dirigée par Gerd Finke et Nadia Brauner
soutenue le 6 septembre 2004 à Grenoble (France)

Jury composé de

Dominique de Werra	Président
Jacques Carlier	Rapporteur
Martine Labbé	Rapporteur
Christophe Rapine	Examinateur
Gerd Finke	Directeur
Nadia Brauner	co-Directrice

L'homme n'existe que pour être dépassé.
Qu'avez-vous fait pour le dépasser ?
— *Friedrich Nietzsche*

Remerciements

C'est pas pour dire, les copains, mais le patron,
quand même, quel type...
— *Marcel Gotlib*

Trois années ont été déroulées, pour autant de paires de chaussures épuisées sur des milliers de kilomètres de bitume, de terre et de cailloux gros ou petits. Mais, au delà de la sereine solitude du coureur de fond, ce sont avant tout de magnifiques personnalités que j'ai eu le privilège de découvrir, parfois dans les lieux les plus inattendus : perdu en forêt, accroché à une falaise, pendu à un cable, sur une scène musicale, voire dans une salle de cours ou même dans un laboratoire de recherche !

Tout est si intimiment lié, tous m'ont été si précieux pour rendre ces trois années si belles et si pleines, que je ne sais par qui commencer ma partition sans revenir sans cesse à toi, ma demoiselle Julie.

Il y a d'abord l'armature. M'man, P'pa : vous avez initié l'auteur, sinon le travail. Isabelle, François, Cécile, ça y est : j'ai été petit dernier, comme il se doit, mais à mon tour je ne suis plus étudiant.

Il y a ensuite le tempo. Cela fait plus de vingt, de dix, de cinq ans, peu importe pour vos amitiés intemporelles, que vous participez à ce qu'est l'auteur : Julien, Julien, Sandrine, Olivier et Fabien ; Benoît, Mathieu et Véronique.

Il y a enfin les accords. Tu es là, François, dont le pas aussi sûr que les lectures a accompagné nombre de mes errements, notamment entre les gares de Chambéry et de Grenoble ; Sophie et Ludo, avec vous j'ai quitté pas mal de chemins fort sympathiques ; Laurent, tu m'a permis de vaincre le « Stress ».

Quant à vous, Guillaume, Cécile, Aude, Olivier, Laurent, Etienne, Sophie, Jocelyn, leurs condisciples de mon laboratoire d'adoption, c'est avec plaisir que les circonstances m'ont permis de me joindre à vous — et à toi, Nicolas.

Et puisque *sans musique, la vie serait une erreur*, j'en ai écouté beaucoup et joué passionnement. Antoine, Cécile, Charles, Christophe, David, JS, Manu, Régis : sans vous, ces ressourçantes escapades sonores n'auraient pas eu les mêmes saveurs créatives et complices.

Montagne et musique ont ainsi rempli mes soirées, mes congés et ont été propices à ce que mes journées soient consacrées à tout mettre en boîte. Et puisqu'il n'est pas bon de se laisser enfermer, il eût été regrettable de ne pas visiter les bureaux voisins...

Dans la faune parfois étrange de la recherche, les herbivores sont thésards, avec Mehdi en première ligne pour supporter mes sautes d'humeur, Eric et Haris pour

m'accueillir, Mnacho, Eric, Laetitia, Nicolas, Frédéric, Prakash, Simon, Marie, Julien pour m'accompagner.

Dans le rôle des bienveillants « prédateurs », il y a les permanents : notamment Sylvain Gravier, Michel Burlet et Andras Sebö, aux discussions — qu'elles soient sarcastiques, pédagogiques ou techniques — intéressantes ; il y aussi Frédéric Maffray, Myriam Preissmann, Yann Kieffer. Et il y a surtout Maurice Queyranne dont la pertinence, la gentillesse et l'attention m'ont été d'une grande aide.

Mais la survie dans un tel environnement ne serait pas possible s'il n'y avait pas, pour en assurer la cohésion, l'efficace Béatrice, le disponible Robert, mais aussi Jacky, Henry, Amandine, Claudia, Christine, Josiane et Joséphine. Et puis, dans le rôle du berger incitant son troupeau à explorer tous les paturages dignes d'intérêt, campe Nicolas Balacheff, dont l'ouverture donne l'opportunité d'être un peu plus qu'un simple étudiant-chercheur de passage.

Ma vie de thésard passe aussi par Gérard Cognet et l'ensemble du CIES de Grenoble, par Laurent Mounier et toute son équipe avec qui j'ai eu l'immense plaisir d'enseigner. Mais cette expérience riche et essentielle d'enseignant ne me fait pas oublier mes études encore récentes et les professeurs qui m'ont donné le goût de l'informatique bien faite, en particulier Denis Naddef, Wojciech Bienia et Xavier Nicollin.

Il fallait toutefois que ce travail se termine. Jacques Carlier et Martine Labbé, vous avez accepté d'en être les rapporteurs avec une disponibilité touchante et un intérêt encourageant. Christophe Rapine, ce n'est bien sûr pas pour notre goût commun pour Nabokov et le Saint-Nectaire que je tenais à t'avoir dans mon jury. Dominique de Werra, ça a été un honneur des plus plaisants que de soutenir sous votre présidence alliant l'humour à la maîtrise du débat.

Vous tous, vous quatre, membres de mon jury, votre intérêt et votre gentillesse ont su désarmer la tension qui accompagne une fin de doctorat, l'ont rendue douce, et même un peu nostalgique.

En filigrane, derrière chaque mot, derrière chaque caractère de ce travail il y a Nadia Brauner et Gerd Finke, sans qui rien n'aurait été fait. Initiateurs, continuateurs.

Nadia, toujours souriante, sans tes encouragements et ton dynamisme, certains bas l'auraient été plus, certains hauts moins. C'est un plaisir de travailler avec toi, et je dédie chaque erreur corrigée à ta précision et ton pointillisme salutaires ; les autres fautes, je me les garde.

Gerd. Il y a trois ans, je pensai ne solliciter qu'un directeur de thèse... finalement si peu dirigiste : plutôt un guide, un conseiller très éclairé, si précieux. Vous savez être là, efficace et pertinent. Vous savez partager un savoir et une expérience riches et rassurants. Et vous savez aussi donner confiance et encouragements, sans concession. Il y a trois ans, je sollicitai non seulement un professeur, un Maître, mais surtout un grand Monsieur.

Ce travail n'est, ainsi, pas seulement le mien, mais aussi un peu celui de tous ceux

qui en ont partagé les bons ou les rares mauvais moments. Un grand merci, donc, à l'Université Joseph Fourier et au Laboratoire Leibniz-IMAG de m'avoir accueilli en leur sein ; un immense merci à toutes les personnes que j'ai pu, avec plaisir, y côtoyer.

Table des matières

Remerciements **3**

Table des matières **7**

Pour commencer **11**

I Rangements d'objets multiboîtes **15**

1 Ranger des objets multiboîtes **17**
1.1 Les problèmes multiboîtes 17
1.1.1 Présentation 17
1.1.2 Cas connus et problèmes liés 20
1.2 Classification 23
1.3 Quelques exemples d'applications 24
1.3.1 Conception de réseaux de télécommunication 24
1.3.2 Analyse de documents 27
1.3.3 Sauvegardes de fichiers 27
1.3.4 Tests de médicaments 28
1.3.5 Répartition de l'utilisation de machines 28
1.3.6 Développement logiciel 28
1.3.7 Organisation d'une école d'été 28
1.4 Envoi 29

2 Modèles linéaires **31**
2.1 Modèles pour $B|size_j|H_{\max}$ 31
2.1.1 Un modèle en variables binaires 31
2.1.2 Un modèle en variables entières 32
2.2 Adaptations aux autres problèmes 34

3 Complexité, approximabilité **37**
3.1 Complexité des problèmes multiboîtes 37
3.2 Approximabilité des problèmes multiboîtes 40
3.3 Résolutions polynomiales pour $B|size_j, h_j = 1|\cdot$ 42
3.4 En résumé 44

4 Sur les liens entre rangements et ordonnancements **47**
4.1 Problèmes associés . . . 47
4.2 D'un ordonnancement à un rangement . . . 49
4.2.1 Garanties pour un abaissement de $Pm|size_j, p_j = 1|C_{\max}$ . . . 51
4.2.2 Quelques remarques . . . 52
4.3 D'un rangement à un ordonnancement . . . 54
4.4 Parti pris . . . 55

5 Le cas $Bm|any_j|\cdot$ **57**
5.1 Principes généraux ; le cas $Bm|any_j|H_{\max}$ . . . 57
5.2 Adaptation au cas $Bm|any_j|W$ . . . 60
5.3 Quelques conséquences . . . 61
5.4 Un FPTAS pour $Bm|any_j|H_{\max}$ . . . 61
5.5 Pour aller plus loin . . . 63

6 Le cas $B|allow_j|H_{\max}$ **65**
6.1 Formalisation de $B|allow_j|H_{\max}$ . . . 65
6.2 $B|allow_j, h_j = 1|H_{\max}$ et flots maximaux . . . 66
6.3 $B|allow_j, h_j = 1|H_{\max}$ et (g, f)-facteurs . . . 68
6.3.1 Un (g, f)-facteur particulier . . . 69
6.3.2 Résolution par (g, f)-facteur . . . 70
6.3.3 Conclusions sur l'approche par (g, f)-facteurs . . . 76
6.4 Pour mettre un peu de couleur . . . 77

II Résolution des problèmes $B|size_j|H_{\max}$ **79**

Rappels sur le problème $B|size_j|H_{\max}$ **81**

7 Bornes inférieures (et une pincée de bornes supérieures) **83**
7.1 Bornes de volume . . . 84
7.2 Raisonnement énergétique . . . 87
7.3 Amélioration systématique des bornes . . . 89
7.4 Synthèse . . . 92

8 Heuristiques à performances garanties **95**
8.1 Garanties pour une solution quelconque . . . 95
8.2 Meilleur placement . . . 96
8.2.1 La règle du meilleur placement . . . 96
8.2.2 Performances de la règle MP . . . 99
8.2.3 Meilleur placement décroissant . . . 106
8.2.4 Ordonnancements par liste et règle MP . . . 109
8.3 Placement dans les boîtes suivantes . . . 110
8.4 Placement optimiste . . . 113
8.5 Diviser pour régner . . . 115

8.5.1 L'algorithme et ses performances . . . 115
8.5.2 ADR et la « méthode par différences » . . . 118
8.6 Déplacements d'objets . . . 119
8.7 Quelques précisions et un récapitulatif . . . 121

9 Un schéma d'approximation en temps polynomial 125
9.1 Schéma d'approximation et conséquences . . . 125
9.2 Compléments . . . 127

10 Une approche évolutionniste 129
10.1 Les algorithmes génétiques . . . 129
10.2 Un algorithme hybride pour $B|size_j|H_{\max}$ . . . 131
10.2.1 Un algorithme hybride . . . 131
10.2.2 Altération d'un individu . . . 132
10.2.3 Gestion de la population . . . 132
10.3 Configuration de l'algorithme . . . 134
10.3.1 Réglages des paramètres . . . 134
10.3.2 Comportement de l'algorithme . . . 136
10.4 Extensions . . . 136

11 Comparaison expérimentale des algorithmes 139
11.1 Les données . . . 139
11.1.1 Méthodes de génération des données . . . 139
11.1.2 Les données utilisées . . . 140
11.2 Les bornes inférieures . . . 141
11.2.1 Bornes testées . . . 141
11.2.2 Résultats . . . 141
11.3 Les heuristiques . . . 144
11.3.1 Heuristiques testées . . . 144
11.3.2 Résultats . . . 145
11.3.3 Ordres aléatoires . . . 150
11.3.4 Remarques conclusives . . . 151
11.4 Apport de l'algorithme génétique . . . 154
11.5 Où il reste du travail . . . 159

Pour finir ? 161

Prologue à une suite 163

Perspectives 167
Les perspectives, en bref . . . 167
Questions ouvertes . . . 169

Annexes **171**

A Complexité et approximabilité (synthèse) **173**

B Compléments **175**
B.1 Performances du PTAS du chapitre 9 lorsque $size_j = 1$ 175
B.2 Temps d'exécution des algorithmes testés au chapitre 11 176

C Glossaire **181**

D Notations et abréviations **189**

E Outils et logiciels **191**

Liste des algorithmes **193**

Liste des figures **195**

Liste des tables **197**

Bibliographie et références **199**

Pour commencer

En tant qu'outil, les ordinateurs ne seront rien
de plus qu'une égratignure sur notre culture.
En tant que défi intellectuel, ils sont sans précédent
dans l'histoire culturelle de l'humanité.
— *Edsger Dijkstra*

Le chanteur l'annonçait : il s'agit de « les [mettre] tous dans des boîtes, petites boîtes toutes pareilles »[1]. C'est exactement ce qu'il m'a été proposé de faire, et j'y trouve mon plaisir depuis presque trois ans maintenant.

Certaines personnes, préjugeant sans doute qu'une thèse se doit d'avoir un sujet compliqué et que tout ce qui a trait aux mathématiques se doit d'être abstrait et abscons, ont cru que je me moquais lorsque j'annonçais que je rangeais dans des boîtes de simples objets... peut être pas si simples que ça, lorsque l'on pense à l'extraordinaire richesse et à la complexité de nos « enfantins » nombres entiers, comme l'œuvre de G. Ifrah [71, 72] en témoigne.

Il y a ainsi beaucoup de belles choses à réaliser, juste avec des segments à empiler ; notamment de très compliquées. C'est là l'un des charmes de la recherche opérationnelle, et de la combinatoire en général.

Gerd Finke, mon directeur de thèse, m'a proposé le concept de « rangement d'objets multiboîtes ». Avec Nadia Brauner, ma co-directrice, ils ont encadré attentivement et efficacement mes travaux, tout en sachant me laisser l'initiative des directions dans lesquelles je voulais mener mes recherches. Le champ ouvert est banalement immense ; ce travail ne fait qu'apporter les premières pierres.

Sont-elles vraiment les premières ? pas tout à fait, bien sûr. En fait le concept d'objets multiboîtes est très lié aux tâches multiprocesseurs, étudiées en ordonnancement, et de nombreuses autres variantes des problèmes de rangement ont déjà été explorées. Le sujet était ainsi déjà très documenté, essentiellement sur ses frontières et à-côtés, avec énormément de méthodes et d'approches disponibles, n'attendant qu'à être adaptées sur ce nouveau système de contraintes.

* * *

Rentrons, légèrement, dans le sujet. Que sont donc ces « objets multiboîtes » ? Pour un problème de rangement classique, les objets sont des segments qu'il faut réussir à empiler dans de plus grands segments, tous identiques : les boîtes. Dans le

[1] *Little Boxes* de Malvina Reynolds (adaptation française de Graeme Allwright).

cas d'objets multiboîtes, un objet est composé de plusieurs segments, tous identiques, qu'il faut chacun mettre dans une boîte différente (le nombre de segments est appelé la largeur d'un objet).

Nous nous confrontons donc à un problème d'allocation de ressources, soit à la première partie d'un problème d'ordonnancement[2], et de nombreuses techniques issues de ce domaine seront utilisées.

Pour entrer un peu plus dans les détails, et avant d'entamer le récit de mes travaux proprement dit, voici un avant-goût de ce qui attend le lecteur.

* * *

La première partie de ce travail traite du concept de rangement multiboîte de manière générale. Aussi commence-t-elle par une description détaillée (tout d'abord informelle pour bien saisir l'idée, puis formelle pour éviter tout malentendu) du concept d'« objet multiboîte ». Cette présentation s'accompagne d'un tour d'horizon des problèmes liés et des sous-cas déjà abondamment traités dans la littérature. On trouvera aussi une classification des différents modèles, selon le type des objets et les objectifs considérés, et plusieurs exemples d'applications. La lecture de cet ensemble, qui constitue le chapitre 1, permettra au lecteur d'être familier avec toutes les variations utilisées par la suite du concept d'objet multiboîte.

Quelques modèles linéaires en nombres entiers sont proposés (chapitre 2). On ne s'arrêtera pas longtemps dessus, cette approche n'ayant pas été développée dans mes travaux, et l'on passera rapidement au chapitre 3 qui pose les bases théoriques de la difficulté des problèmes multiboîtes. Dans ce chapitre, la $\mathcal{NP}$-complétude des principaux modèles est démontrée, et le caractère critique (sur le plan théorique) du nombre de boîtes est révélé : pour la plupart des modèles, l'aspect arbitraire ou fixé de ce paramètre permet de passer de $\mathcal{NP}$-complet au sens fort, à simplement $\mathcal{NP}$-complet mais soluble en temps pseudo-polynomial. On trouvera également dans ce chapitre la résolution polynomiale pour plusieurs objectifs du cas d'objets de largeur fixe et de hauteur unitaire.

Dans les chapitres 1 et 3, les premiers jalons illustrant la proximité des rangements multiboîtes et des ordonnancements de tâches multiprocesseurs ont été posés. Le chapitre 4 est entièrement dédié à la clarification de ce lien. On y montre que l'apport de l'un pour l'autre doit avant tout être méthodologique, et que c'est dans l'adaptation des techniques de résolution que chacun a le plus à bénéficier de l'autre.

Le chapitre 5 applique immédiatement ce principe, puisque des résultats similaires basés sur des algorithmes semblables existent déjà pour l'ordonnancement. Il s'occupe du cas d'objets de largeur variable. Des algorithmes pseudo-polynomiaux (pour un nombre de boîtes fixé) sont proposés lorsque l'objectif est de maximiser le nombre pondéré d'objets rangés, ou bien la hauteur des boîtes pour tous les ranger. Dans ce deuxième cas, l'approche est étendue et un FPTAS est déduit.

[2]Pour reprendre les mots de J. Błażewicz dans un de ses cours : « l'ordonnancement est l'affectation dans le temps de ressources, afin de réaliser un ensemble de tâches ». On garde le côté affectation, sans s'occuper, néanmoins, de la notion de temps.

Au chapitre 6, le cas d'objets de largeur constante, mais avec des incompatibilités entre objets et boîtes, est traité. Pour des objets de hauteur unitaire et l'objectif de minimiser la hauteur des boîtes, le problème est résolu polynomialement avec une approche de type flot. Ces premiers résultats sont ensuite développés et améliorés en tirant parti des propriétés spécifiques du problème, traité comme un cas particuliers de (g, f)-facteur d'un graphe biparti.

Ces deux derniers chapitres closent ainsi la première partie en résolvant excatement, au moins théoriquement, deux cas particuliers. La deuxième partie est, elle, entièrement dédiée à la résolution d'un autre cas particulier : le cas d'objets de largeur fixe, avec pour objectif de minimiser la taille des boîtes. Ce problème étend, notamment le célèbre Ordonnancement de tâches sur machines parallèles ($P||C_{\max}$). La première partie a suffi à poser le cadre théorique de ce problème : il est $\mathcal{NP}$-difficile au sens fort dans le cas général ; si le nombre de boîtes est fixé, il est alors soluble en temps pseudo-polynomial et admet un FPTAS. Il s'agit maintenant de le résoudre en pratique.

Étant donné la complexité du problème, un compromis doit être trouvé entre qualité des solutions et temps de calcul. Entre résoudre optimalement aussi vite que possible et résoudre rapidement aussi bien que possible, je me suis concentré sur la deuxième approche.

La deuxième partie débute avec la présentation de bornes (inférieures) pour ce problème et de leurs performances (chapitre 7). En plus de la borne évidente basée sur le volume total des objets, on trouvera une borne inspirée du « raisonnement énergétique » pour l'ordonnancement avec contrainte de ressources, et une méthodologie tirée du Bin-Packing permettant d'améliorer de manière systématique des bornes inférieures.

Le chapitre 8 est le cœur de la deuxième partie ; y sont proposées plusieurs heuristiques à performances garanties. Parmi elles, on notera essentiellement la règle du « meilleur placement », qui est une extension de la règle LPT pour $P||C_{\max}$. Les résultats classiques pour cette dernière sont ainsi étendus, notamment le fait que, si les objets sont triés par hauteurs décroissantes, la méthode est une 4/3-approximation.

À partir de cette règle de meilleur placement, le chapitre 9 présente un PTAS. Théoriquement moins bon que le FPTAS présenté dans la première partie, il ouvre toutefois d'autres perspectives, notamment dans la conjonction des deux approches, pour le développement d'algorithmes encore plus performants.

Les heuristiques proposées au chapitre 8, bien que de très bonne qualité, laissent encore place à quelques gains : c'est pourquoi, dans le chapitre 10 un algorithme génétique est proposé. Celui-ci est en fait hybride, et est basé sur les meilleures heuristiques.

Pour évaluer réellement la qualité des bornes, des heuristiques, et de l'apport de l'algorithme génétique, le chapitre 11 termine la deuxième partie par une analyse empirique de toutes ces méthodes.

La traditionnelle conclusion générale achèvera ce document, conjointement avec la liste des questions ouvertes éparpillées au fil des pages.

* * *

Avant de terminer cette introduction, je veux attirer l'attention sur quelques points importants pour une lecture plus agréable et plus facile ; en premier lieu le fait que l'ensemble des notations utilisées est répertorié dans l'annexe D.

Beaucoup de concepts usuels de la recherche opérationnelle ne sont pas définis dans le texte (par exemple, une « α-approximation »). Ils sont en revanche rassemblés dans l'annexe C (conjointement aux nombreux néologismes du jargon mathématique). Deux raisons m'ont amené à ce choix. Tout d'abord, cela permet une lecture beaucoup plus fluide et continue pour quiconque est familier avec les concepts ; quant au novice, il me semble pertinent qu'il acquière au préalable les bases indispensables à une réelle compréhension, ou qu'il se contente d'un survol. De plus, la plupart des termes sont utilisés en différentes places, notamment pour annoncer ce qui va suivre et ainsi expliciter la ligne directrice de ces travaux. Il est alors peu opportun de définir au chapitre 1 ce qu'est un FPTAS, uniquement parce qu'on va en proposer un au chapitre 5.

Pour la plupart des chapitres, j'indique dans une note de bas de page (non signalée) les parutions et présentations éventuelles du contenu de ce chapitre, ainsi que les collaborations.

Un dernier détail : l'orthographe de l'adjectif et néologisme « multiboîte ». J'ai décidé, en suivant l'exemple de « multiprocesseur » donné par le *Petit Robert* [112], qu'il serait variable et s'accorderait en nombre. Ainsi parlerai-je d'un rangement multiboîte et d'objets multiboîtes.

Première partie

Rangements d'objets multiboîtes

L'existence est une suite de notes de bas de page pour un chef-d'œuvre vaste, obscur et inachevé.
— *Vladimir Nabokov*

Ranger des objets multiboîtes

Toute science crée une nouvelle ignorance.
Tout apport nouveau crée un nouveau néant.
— *Henri Michaux*

Ce chapitre sert d'introduction aux concepts d'objets et de rangements multiboîtes : ceux-ci sont définis et placés dans leur contexte.

Une première section permet de se familiariser avec ces notions et de les appréhender de manière informelle, ceci pour en acquérir une idée intuitive : on y trouve une présentation de la problématique suivie d'un rappel des principaux travaux déjà existants sur des problèmes proches et des cas particuliers.

La deuxième section est plus formelle : y sont définis précisément les principaux modèles d'objets et les différents objectifs considérés dans ce document.

Afin de mieux concevoir et de motiver davantage le concept d'objets multiboîtes, des exemples d'applications sont décrits dans la troisième section.

1.1 Les problèmes multiboîtes

Le concept d'« objet multiboîte » a été proposé par Gerd Finke [55, 56]. L'idée sous-jacente est d'étendre les problèmes de rangement de la manière dont les problèmes d'ordonnancement de tâches multiprocesseurs étendent les problèmes classiques d'ordonnancement sur machines parallèles : un objet peut avoir besoin de plusieurs boîtes pour être rangé.

Cette section débute par une description plus précise des problèmes de rangement d'objets multiboîtes, avec une emphase particulière sur le lien avec les problèmes d'ordonnancements de tâches multiprocesseurs. Une seconde sous-section est ensuite dédiée aux problèmes liés et aux cas particuliers traités dans la littérature.

1.1.1 Présentation

Ranger des objets dans des boîtes est un défi courant des applications industrielles, comme de la vie de tous les jours. Le cas classique est de disposer de beaucoup de boîtes identiques et de vouloir ne se servir que d'un minimum d'entre elles

Ce chapitre est essentiellement une reprise, étendue, des parties correspondantes de [96, 98] ; il a été présenté en diverses occasions [90, 97, 96].

pour tout ranger. Avec un tel objectif, il s'agit du problème bien connu du Bin-Packing.

Un autre problème extrêmement fréquent et classique est l'Ordonnancement de tâches sur machines parallèles ($P||C_{\max}$). Dans ce cas, différentes tâches doivent être exécutées, plusieurs machines sont à disposition et l'objectif est de finir le travail le plus tôt possible.

Ces deux problèmes sont en fait très liés. Fondamentalement, ils correspondent au même problème de décision (est-ce qu'on peut faire tenir tous les objets dans m boîtes de hauteur H ; est-ce qu'on peut effectuer toutes les tâches sur m machines en un temps C ?) et ne diffèrent que par l'objectif à minimiser. Aussi passe-t-on facilement de l'un à l'autre. Par exemple, une simple rotation du diagramme de Gantt représentant un ordonnancement permet de voir celui-ci comme un rangement (voir figure 1.1).

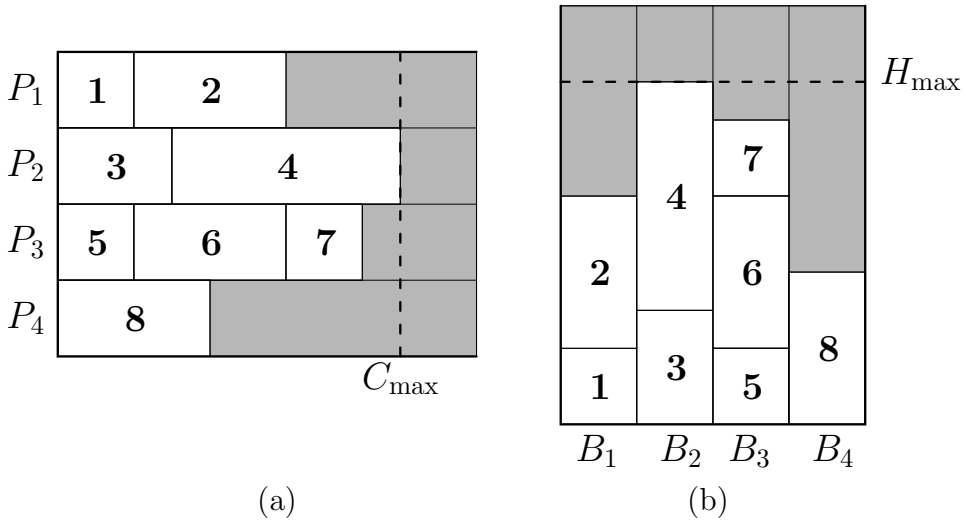


Figure 1.1 – Passage d'un ordonnancement à un rangement.
(a) un ordonnancement ; (b) le rangement correspondant.

Depuis une vingtaine d'années, les problèmes d'ordonnancement sur machines parallèles ont été étendus : une tâche peut avoir besoin de plusieurs machines, simultanément, pour son exécution. Cela correspond à de nombreux problèmes réels, notamment en informatique avec l'utilisation de structures parallèles.

Les objets multiboîtes sont aux objets habituels ce que les tâches multiprocesseurs sont aux tâches classiques. Ainsi, un objet multiboîte est constitué de plusieurs bouts identiques, chacun devant être emballé dans une boîte différente. Le parallèle entre ordonnancements et rangements existe toujours, avec cependant une différence importante : dans un ordonnancement de tâches multiprocesseurs, il peut y avoir du temps mort *entre* deux tâches (phénomène impossible[1] pour des tâches monoprocesseurs s'il n'y a pas de précédences), dû au fait qu'une tâche nécessite plusieurs

[1]Pour être exact : tout temps mort peut être supprimé sans modifier la réalisabilité de la solution, ni en dégrader la valeur.

machines *en même temps*. Lorsque, comme précédemment, on tourne le diagramme pour voir cet ordonnancement comme un rangement, ce temps mort n'a pas de sens : il n'y a pas de notion de temps dans un rangement. Il faut donc laisser tomber les objets pour boucher les trous. La figure 1.2 illustre ce phénomène.

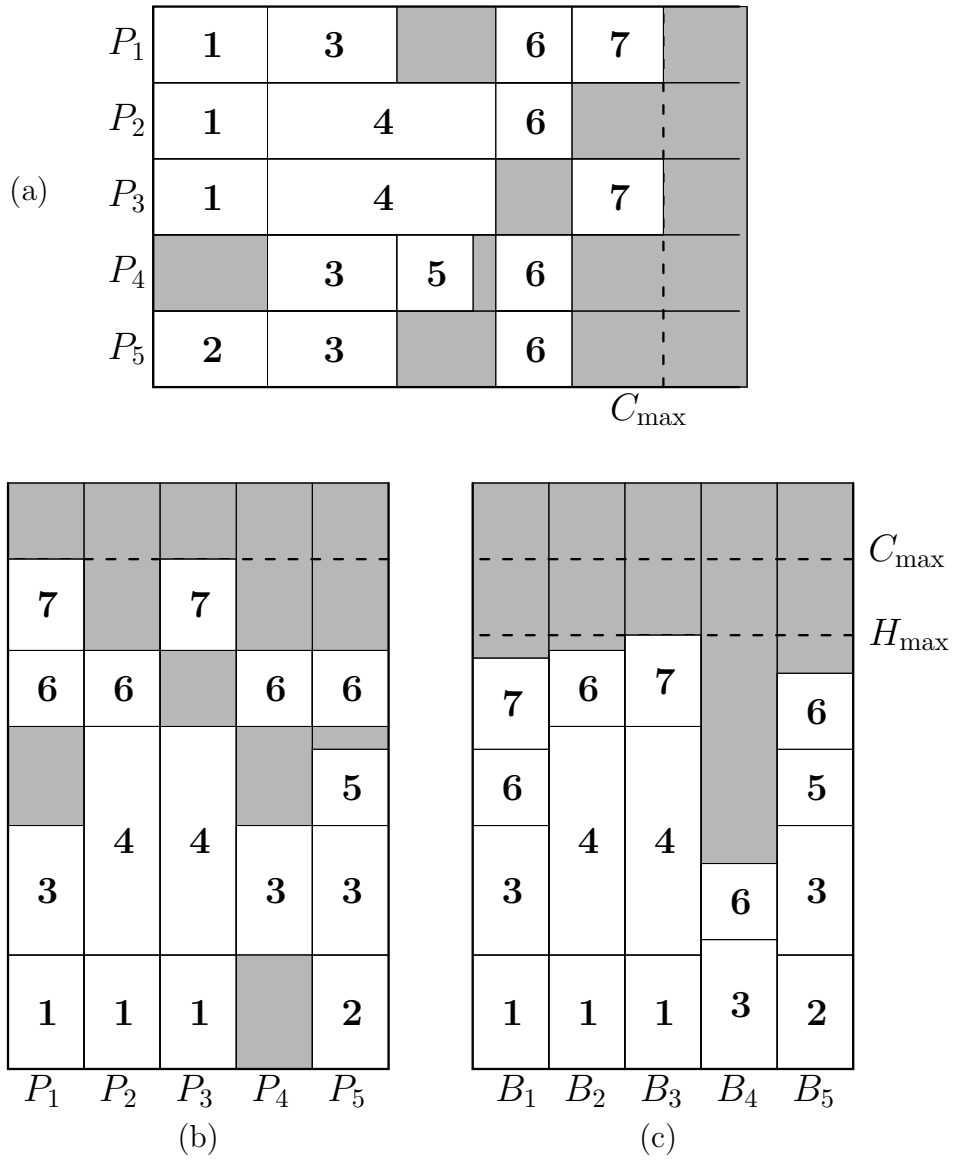


Figure 1.2 – Passage d'un ordonnancement de tâches multiprocesseurs à un rangement d'objets multiboîtes.
(a) un ordonnancement ; (b) le même, tourné ; (c) le rangement correspondant.

Cette transformation met en évidence une autre façon de voir les problèmes de rangement d'objets multiboîtes : non pas seulement comme des extensions des problèmes de rangement, mais aussi comme des relaxations des problèmes d'ordonnancement de tâches multiprocesseurs pour lesquelles la notion de temps est mise de côté. Il n'y a ainsi aucune notion d'ordre au sein d'une boîte, et la contrainte de

simultanéité d'exécution des tâches ne se retrouve pas pour des objets. Ce rapport entre rangements et ordonnancements sera développé au chapitre 4.

Par abus de langage, j'utiliserai souvent les termes de « rangement multiboîte » ou de « problème multiboîte » en place de « rangement d'objets multiboîtes » ou de « problème de rangement d'objets multiboîtes ». De telles ellipses n'ajoutent aucune confusion, étant donné que seuls les objets sont réellement multiboîtes, et autorisent une plus grande fluidité d'expression.

Une instance d'un rangement multiboîte consiste en la donnée de n objets et du nombre de boîtes m.

Pour un objet multiboîte O_j, $size_j$ est sa largeur (le nombre de boîtes dans lesquelles il est, ou doit être, rangé : chaque partie d'un objet doit être rangé dans une boîte différente) et h_j sa hauteur (la place qu'il occupe dans chaque boîte). La hauteur et la largeur respectivement du plus grand et du plus gros objet sont représentées par $h_{\max}$ et $size_{\max}$.

Pour une boîte B_i, H_i est sa hauteur (la hauteur d'une boîte est la somme des hauteurs des objets qui y sont rangés). $H_{\max}$ est la hauteur de la plus haute boîte, et Δ est la plus grande différence de hauteur entre deux boîtes. Toutes les boîtes sont identiques et l'ordre des objets au sein d'une boîte est sans importance. Pour avoir des notations simples, je confonds souvent une boîte avec l'ensemble des objets qu'elle contient ; ainsi écrivé-je $O_j \in B_i$ pour dire qu'un des bouts de l'objet O_j est rangé dans la boîte B_i.

Plusieurs objectifs sont envisageables, notamment minimiser la taille des boîtes ou leur nombre ; maximiser le nombre d'objets rangés. Ceci m'amène à parler plus en détails des problèmes liés et des cas connus.

1.1.2 Cas connus et problèmes liés

Voici venu le temps d'un tour d'horizon des problèmes liés aux rangements multiboîtes. Pour commencer, les nombreux cas particuliers traités dans la littérature sont abordés, classés par objectif. Ensuite viennent en complément quelques autres problèmes très proches des problèmes de rangements multiboîtes.

Minimiser la taille des boîtes

Lorsqu'il s'agit de minimiser la hauteur des boîtes, le problème est généralement considéré sous l'angle de l'ordonnancement : cela correspond à la minimisation du temps global d'exécution (critère $C_{\max}$).

Ainsi, le rangement d'objets de largeur unitaire est le problème d'Ordonnancement sur machines parallèles ($P||C_{\max}$). Ce problème est $\mathcal{NP}$-difficile au sens fort [60]. Graham a proposé l'algorithme LPT[2] qui consiste à placer les tâches par ordre décroissant de leurs durées, dès qu'un processeur est disponible ; il a prouvé une garantie de 4/3 pour cet algorithme [64]. Une autre approche très efficace, avec

[2]LPT : *Longuest Processing Time first* (les plus longues d'abord).

des performances légèrement supérieures, est due à Coffman, Garey et Johnson : *Multifit* [36], basée sur les algorithmes d'approximation pour le Bin-Packing (voir plus bas). Leurs idées ont été reprises et développées par Hochbaum et Shmoys, qui en ont déduit un PTAS [68]. Pour un nombre de machines fixé, Sahni a proposé un FPTAS [113]. Il existe de nombreuses variantes et extensions de ce problème (machines dédiées, précédences, préemptions, *etc.*) qui dépassent le cadre des rangements multiboîtes. Pour plus de détails, le lecteur est renvoyé à [32] ainsi qu'à [16].

Lorsqu'on ne considère que deux boîtes, le problème de rangement est équivalent au très classique Bi-Partition. Ce problème est $\mathcal{NP}$-complet, mais il se résout en pratique bien et en temps pseudo-polynomial [60]. Il admet un FPTAS [10].

N'ayant *a priori* que peu à voir avec les problèmes d'ordonnancement, le problème de l'équilibrage de la charge d'un anneau (connu dans la littérature sous le nom de Ring Loading Problem) est cependant aussi un cas de rangement d'objets multiboîtes, mais à « boîtes dédiées », qui apparaît dans la conception de réseaux de télécommunications de type SONET bidirectionnel. Les objets n'ont pas de largeur fixe, mais peuvent être rangés dans l'un ou l'autre de deux ensembles complémentaires de boîtes, dépendant de l'objet considéré. Ce problème est présenté plus avant dans les applications des problèmes multiboîtes (section 1.3.1).

Minimiser le nombre de boîtes

Un deuxième critère d'optimisation habituel est de minimiser le nombre de boîtes utilisées pour ranger tous les objets. Le problème est alors vu sous l'angle du Bin-Packing (appellation traditionnelle lorsque les objets sont de largeur unitaire).

Ce problème a été très étudié, et le lecteur intéressé se reportera à [37]. Je rappelle ici les principaux résultats. Le Bin-Packing est $\mathcal{NP}$-difficile au sens fort et il n'est pas approximable à moins de 3/2 (sauf si $\mathcal{P} = \mathcal{NP}$) [60]. Il existe toutefois d'excellents algorithmes pour le résoudre, notamment FF[3] et sa garantie de 17/10 [75]. Pour des garanties asymptotiques (et excellentes en pratique) FFD[4] a un rapport de 11/9 et MFFD[5] de 71/60 [61]. En fait des PTAS asymptotiques existent [54] et l'on peut même avoir des algorithmes polynomiaux asymptotiquement optimaux [76]. À noter également de récentes avancées pour le calcul de bonnes bornes inférieures pour ce problème [53], sur lesquelles je reviendrai au chapitre 7.

Le Bin-Packing est décliné en différentes autres versions. L'une d'entre elle est le Bin-Packing ouvert, dans lequel un objet par boîte peut déborder. Ce problème est $\mathcal{NP}$-difficile mais admet un PTAS (asymptotique) [100] ; le cas où un ordre est imposé sur les objets, et que l'objet qui dépasse une boîte doit être celui de plus grand indice a également été étudié [123]. Certains auteurs ont aussi traité le Bin-Packing Dual, pour lequel il s'agit de maximiser le nombre de boîtes, chacune

[3]FF : *First-Fit* (la première qui convient) consiste à ranger un objet dans la première boîte qui peut le contenir.

[4]FFD : *First-Fit Decreasing* (FF, selon l'ordre décroissant) est une variation de FF où les objets sont considérés par ordre de tailless décroissantes.

[5]MFFD : *Modified First-Fit Decreasing* (modification de FFD) est une version améliorée de FFD qui traite de manière particulière les objets les plus problématiques.

devant avoir un niveau minimum de remplissage. Plusieurs algorithmes polynomiaux avec des garanties de 1/2 à 3/4 ont été proposés [9], ainsi que des bornes supérieures et inférieures conduisant à un algorithme de type « brancher et borner » efficace et exact [82].

Maximiser le nombre d'objets rangés

Le troisième objectif naturel est de vouloir maximiser le nombre (pondéré) d'objets rangés dans un nombre donné de boîtes de taille donnée.

La version pondérée la plus simple de ce problème est le problème du Sac-à-Dos, qui correspond à un problème multiboîte pour des boîtes de hauteur unitaire et des objets de largeur constante (ou bien à une unique boîte). Ce problème apparaît fréquemment comme sous-problème d'autres applications et il est intensivement traité dans la littérature. Signalons seulement ici qu'il est $\mathcal{NP}$-difficile, soluble en temps pseudo-polynomial [60] et qu'il admet des FPTAS [85, 67]. Pour plus de détails, le lecteur est renvoyé à [78].

Le cas non pondéré avec des objets de largeur unitaire et des boîtes de hauteur quelconque est connu sous le nom de Bin-Packing de cardinalité maximum. Pour ce cas, des bornes supérieures et des algorithmes d'approximation ont été proposés [38, 83].

Problèmes liés

D'autres problèmes sont relativement proches des problèmes de rangement d'objets multiboîtes, sans en être des cas particuliers.

Ont déjà été évoqués les problèmes d'Ordonnancement de tâches multiprocesseurs ($P|size_j|C_{\max}$, ou parfois CuSP[6]). Ces problèmes sont $\mathcal{NP}$-difficiles au sens fort dès qu'il y a au moins 5 machines ; ils sont $\mathcal{NP}$-difficiles mais solubles en temps pseudo-polynomial pour 2 ou 3 machines (pour 4 machines, la complexité exacte est inconnue) [44]. Pour des tâches de durée unitaire, le problème reste $\mathcal{NP}$-difficile au sens fort ; il se résout toutefois linéairement si les largeurs des tâches sont bornées par une constante[7] [13]. À noter, pour ces problèmes ainsi que le problème très proche du RCPSP, de récentes avancées pour le calcul de bonnes bornes inférieures [12, 27, 28]. Le concept de tâches multiprocesseurs se décline en de nombreuses variantes ; pour plus de précisions, le lecteur est renvoyé à [43, 15, 14].

Un autre type d'ordonnancements est très proche des problèmes multiboîtes — et d'une certaine manière les généralise — : les ordonnancements avec conflits, où certaines tâches ne peuvent partager le même processeur. Ces problèmes sont généralement $\mathcal{NP}$-difficiles (en particuliers pour trois machines, même pour des

[6] CuSP : *Cumulative Scheduling Problems* (problèmes d'ordonnancements cumulatifs) pour lesquels les tâches requièrent plus d'une unité de la ressource par unité de temps.

[7] Ce « linéairement » mérite quelques précisions. Si les tâches ont une largeur dans $\{1, K\}$, la résolution est effectivement efficace ; pour des tâches ayant une largeur dans $\{1, 2, \ldots, K\}$ la résolution reste linéaire mais passe en fait par la résolution d'un gros système linéaire et s'avère très coûteuse.

durées unitaires [11] ou bien un graphe d'exclusion biparti [18]). Pour plus de détails, notamment quelques algorithmes d'approximation ou la résolution de cas particuliers, se référer à [18, 73, 11]. Cette notion de conflits a aussi été étudiée pour le Bin-Packing [74].

Enfin, il existe les rangements multidimensionnels. Pour ces problèmes, boîtes et objets ont plusieurs dimensions. La différence fondamentale avec des objets multiboîtes formés de plusieurs bouts est que chaque objet n'est qu'un unique bloc. Des approximations et des bornes inférieures ont été données pour la minimisation de la taille des boîtes [31] ou de leur nombre [26, 50, 31, 20].

Pour tous ces modèles théoriques, il faut souvent rajouter des contraintes spécifiques lorsqu'on veut résoudre une application industrielle réelle, et ainsi adapter les méthodes pour résoudre ces systèmes complexes. [21] illustre cela.

1.2 Classification

Comme nous venons de le voir, les rangements d'objets multiboîtes sont très diversifiés et regroupent de nombreux problèmes, selon les particularités des objets et l'objectif considérés. Aussi, dans cette section, une classification est proposée, afin de mettre un peu d'ordre.

En raison des similitudes, la notation pour les problèmes de rangement d'objets multiboîtes est une notation à trois champs semblable à celle existant pour les problèmes d'ordonnancement de tâches multiprocesseurs [15, 43].

Rappelons que la notation à trois champs, proposée par Graham *et al.* [65] et reprise par Błażewicz *et al.* [17], s'écrit sous la forme $\alpha|\beta|\gamma$, où α spécifie l'environnement de travail, c'est-à-dire les processeurs (pour nous, ce sera donc les boîtes) ; β décrit l'ensemble des tâches (pour nous, l'ensemble des objets), notamment les contraintes auxquelles elles sont soumises ; enfin, le dernier champ, γ, indique le critère à optimiser.

Dans ce travail, je n'ai considéré que des rangements dans des boîtes identiques, ce qui est noté B. Les seules contraintes possibles sur un tel environnement sont le nombre et la hauteur des boîtes. Ainsi, Bm indique que le nombre de boîtes est fixé à m, et B^H précise que les boîtes ont une hauteur déterminée H.

En ce qui concerne l'ensemble des objets, je reprends les principales contraintes données dans le cas de tâches multiprocesseurs [15, 43]. Ainsi les objets peuvent avoir des hauteurs fixées (p.ex. $h_j = 1$) ou bornées (p.ex. $h_j \leq k$) ; de même pour les largeurs (p.ex. $size_j = k$ ou $size_j \leq k$). Plus particulières sont les contraintes sur les boîtes qui peuvent ou doivent contenir un objet ; pour préciser cela :

- $size_j$ signifie que chaque objet O_j a besoin d'un nombre fixé $size_j$ de boîtes différentes et qu'il occupe une hauteur h_j dans chacune d'elles.
- any_j indique que chaque objet O_j peut être mis dans un nombre quelconque $1 \leq size_j \leq m$ de boîtes, et qu'il occupe une hauteur dépendant de ce nombre, $h_j(size_j)$, dans chacune d'elles (ce modèle généralise le cas $size_j$).

- fix_j révèle que chaque objet O_j doit être mis dans un ensemble déterminé fix_j de boîtes[8].
- set_j signale que chaque objet O_j peut être rangé dans différents ensembles de boîtes, et qu'il occupe une hauteur h_j dans chacune d'elles (ce modèle généralise le cas fix_j).
- $allow_j$ précise que chaque objet O_j a besoin d'un nombre fixé $size_j$ de boîtes, mais qu'il ne peut être rangé que dans les boîtes autorisées, définies par $allow_j$, et qu'il occupe une hauteur h_j dans chacune d'elles (ce modèle est un cas particulier de set_j).

Les quatre premiers systèmes de contraintes décrits ci-dessus sont les principaux systèmes définis dans le cadre de tâches multiprocesseurs, et ce sont les seuls que je considère dans ce document. Il en existe toutefois d'autres [43], dont certains gardent tout leur sens dans le cas de rangements multiboîtes. Quant au modèle $allow_j$, il n'a pas, à ma connaissance, été considéré tel quel pour les problèmes d'ordonnancement (bien qu'il se rapproche des modèles avec processeurs dédiés). Il sera traité spécifiquement au chapitre 6.

Nous en arrivons maintenant au troisième champ : l'objectif. Dans la mesure où l'ordre des objets au sein d'une boîte n'a pas d'importance, certains critères d'ordonnancement n'ont aucun sens pour les rangements multiboîtes. Pour ce travail, je n'ai retenu que trois critères principaux :

- $H_{\max}$: minimiser la hauteur maximale de m boîtes.
- m : minimiser le nombre de boîtes de hauteur H.
- N (ou W) : maximiser le nombre (ou nombre pondéré) d'objets rangés dans m boîtes de hauteur H.

Lorsqu'aucun objectif particulier n'est considéré, et que seule la réalisabilité du problème entre en compte (problème de décision), l'absence d'objectif est indiquée par $\cdot$.

1.3 Quelques exemples d'applications[9]

Les problèmes de rangement d'objets multiboîtes sont des modèles assez généraux qui incluent de nombreuses applications potentielles. En voici quelques unes.

1.3.1 Conception de réseaux de télécommunication

Un problème de rangement d'objets multiboîtes survient lors de la conception de réseaux de télécommunication de type SONET : équilibrer des signaux le long des anneaux formant le réseau. Ce problème est connu dans la littérature sous le nom

[8] Ce cas n'a pas grand sens pour les problèmes de rangements multiboîtes, puisqu'alors il n'y aucun choix sur le placement des objets. Je l'ai laissé en raison de son emploi au chapitre 4.

[9] Les applications des sections 1.3.2 et 1.3.3 sont dues à G. Finke et N. Brauner [55] ; l'idée de l'application de la section 1.3.7 m'a été soufflée par N. Brauner.

de Ring Loading Problem. Commençons par préciser ce que sont les réseaux SONET, puis par donner quelques résultats de la littérature[10].

La technologie SONET/SDH[11] [118, 122, 119] est un standard de conception de réseaux qui permet une bonne résistance aux pannes sans avoir de sur-coût prohibitif. Elle consiste en l'élaboration d'anneaux auto-cicatrisants (ou *Self-Healing Ring*) : les nœuds du réseau sont regroupés en cycles et sont connectés selon ces cycles en utilisant au moins deux fibres séparées orientées dans des sens opposés. En chaque nœud se trouve un ADM[12] correctement dimensionné qui permet les communications et qui oriente automatiquement le signal sur une fibre ou l'autre. Tous les ADM d'un anneau ont la même aptitude de traitement, qui est donc une caractéristique de l'anneau (sa capacité), dimensionnée en fonction des demandes à satisfaire et du protocole utilisé.

Le protocole le plus simple, qui est le standard européen défini par l'ETSI[13] [48, 47], est d'utiliser des anneaux unidirectionnels, pour lesquels un sens de parcours est dédié au fonctionnement normal, l'autre étant réservé aux cas de défaillance. Comme l'illustre la Figure 1.3, les ADM sont capables de réorienter le signal sur la fibre de sécurité lorsqu'une rupture de connexion survient, ce qui permet de continuer d'assurer le trafic même lors de la défaillance d'une connexion par anneau. De plus on constate qu'avec un tel protocole une demande transite sur tout l'anneau (sur la figure 1.3, par exemple, la demande entre A et C utilise deux arcs dans un sens, les quatre autres arcs dans l'autre) : ainsi la capacité de traitement d'un ADM (sur une fibre) doit être au moins égale à la somme des demandes entre les nœuds de l'anneau, pour le cas où les demandes sont toutes simultanées.

L'élaboration de tels réseaux, et principalement leur découpe en un nombre minimum d'anneaux de capacité donnée, a été largement étudiée.

Goldschmidt, Hochbaum, Levin et Olinick [62], et indépendamment, Brauner, Crama, Finke, Lemaire et Wynants [23, 22], ont montré que la simple partition du graphe des demandes en anneaux de capacité fixée était $\mathcal{NP}$-difficile. Les premiers ont proposé un algorithme d'approximation avec une garantie de $(\sqrt{\frac{k}{2}} + \sqrt{\frac{2}{k}})$, où k est la capacité d'un anneau (leur algorithme est exact pour un graphe de demandes arborescent avec une capacité $k = 3$ [87]). Les seconds ont réalisé plusieurs heuristiques gloutonnes ainsi qu'une méthode tabou pour ce problème. Brauner et Lemaire [25] ont aussi étudié une approche par recouvrement d'ensemble, basée sur l'algorithme glouton de Chvátal [34] pour les problèmes de couverture.

Pour d'autres versions du problème, plusieurs heuristiques gloutonnes ont été

[10]Ce qui suit n'est pas, à l'exception des deux derniers paragraphes, directement lié aux problèmes multiboîtes ; cela permet toutefois de mieux comprendre ce que sont les réseaux SONET. Cela correspond aussi à une partie de mon travail, effectuée en DEA.

[11]SONET : *Synchronous Optical NETwork* (réseau optique synchrone) est la terminologie américaine, SDH : *Synchronous Digital Hierarchy* (hiérarchie des lignes numériques synchrones) européenne.

[12]ADM : *Add/Drop Multiplexer* (multiplexeur par insertion et extraction).

[13]ETSI : *European Telecommunications Standards Institute* (institut européen des standards de télécommunications).

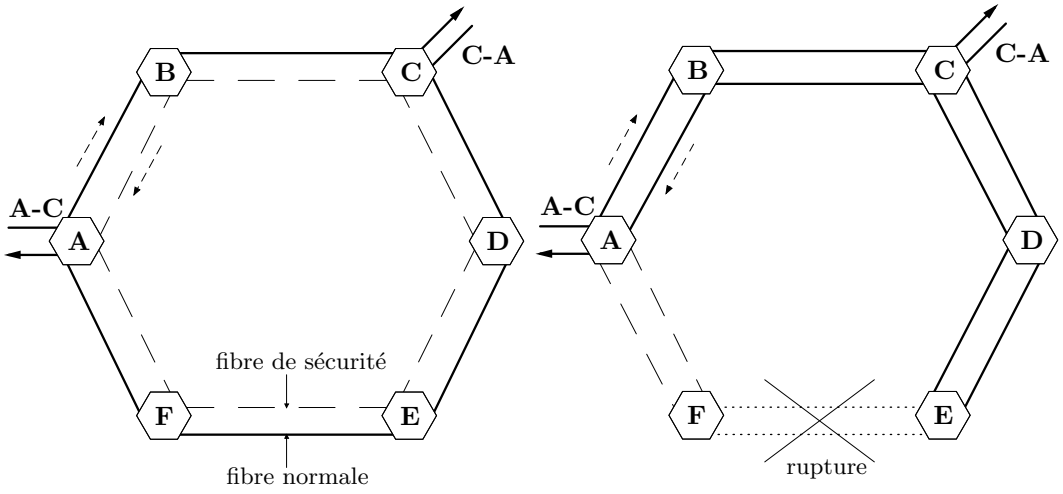


Figure 1.3 – Un anneau unidirectionnel (réseau SDH).
(a) fonctionnement normal ; (b) cas d'une panne.

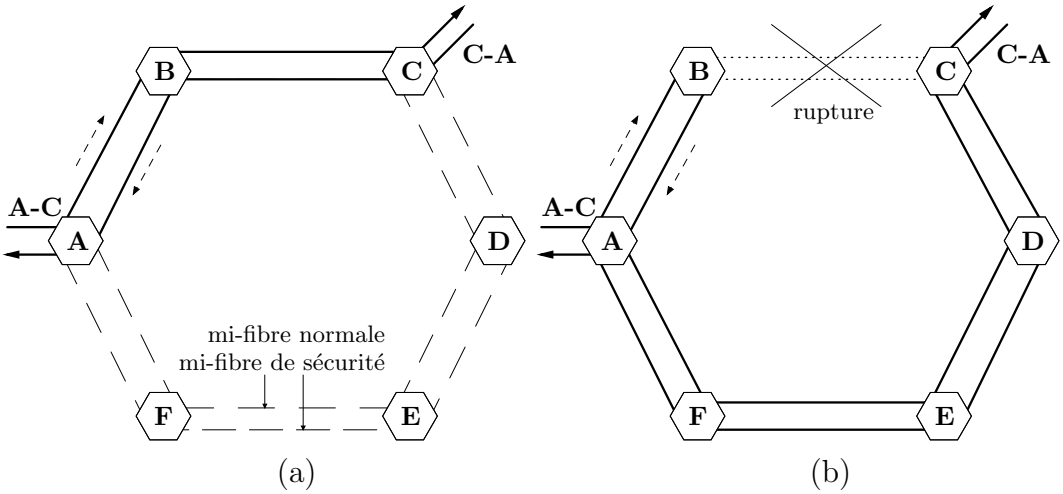


Figure 1.4 – Un anneau bidirectionnel (réseau SONET).
(a) fonctionnement normal ; (b) cas d'une panne.

proposées [63], ainsi que diverses méthodes basées sur la recherche tabou [84, 6, 59] ou la programmation en nombres entiers [86, 120].

Un autre protocole consiste en l'utilisation d'anneaux bidirectionnels, pour lesquels les signaux peuvent aller dans un sens comme dans l'autre, la moitié de la capacité de chaque fibre servant de sécurité. Ce standard permet en général un meilleur rendement (*i.e.* un plus grand débit pour un même dimensionnement) mais s'ajoute alors un problème de répartition des signaux sur chaque fibre pour tirer véritablement profit de cette propriété. Ainsi, un anneau ayant été construit, il faut équilibrer les demandes afin d'en minimiser la somme en un même point. Il faut donc attribuer à chaque demande l'une des deux routes qui lui sont possibles le long de l'anneau (voir figure 1.4).

Ce problème, connu dans la littérature sous le nom de Ring Loading Problem, est un cas particulier de rangement d'objets multiboîtes. Les m arêtes de l'anneau sont les m boîtes et les n demandes sont les n objets (la hauteur d'un objet étant la valeur de la demande). Chaque objet, *i.e.* chaque demande d_{ij}, peut être affecté soit à l'ensemble des arêtes de i à j dans le sens direct, soit à l'ensemble complémentaire. Le problème de répartition de la charge le long d'un anneau est donc un cas particulier du modèle $B|set_j|H_{\max}$.

Schrijver, Seymour et Winkler [114] ont prouvé que ce problème est $\mathcal{NP}$-difficile dans le cas général ; ils ont également construit un algorithme d'approximation garantissant de ne pas dépasser la valeur optimale de plus d'un terme additif (3/2 fois la plus grande demande). Leur approche a été développée par Khanna [79] qui en a tiré un schéma d'approximation polynomial. Lorsque les demandes peuvent être coupées et envoyées de part et d'autre de l'anneau, le problème devient facile [108, 114].

1.3.2 Analyse de documents

Un groupe d'experts examine des documents. Chaque document doit être examiné par plusieurs experts, en fonction de son importance, et un temps d'expertise est attribué à chaque document. L'objectif est de terminer l'expertise le plus vite possible.

En assimilant les experts aux boîtes et les documents aux objets, on obtient typiquement un problème $B|size_j|H_{\max}$, si tous les experts sont compétents pour tous les documents ; dans l'autre alternative, c'est un problème $B|allow_j|H_{\max}$ ou $B|set_j|H_{\max}$.

1.3.3 Sauvegardes de fichiers

On désire sauvegarder un ensemble de fichiers (les objets) sur des disques (les boîtes). Pour des raisons de sécurité, chaque fichier doit être sauvegardé plusieurs fois, selon son importance. Avec pour objectif de minimiser le nombre de disques nécessaires, c'est un problème $B|size_j|m$.

1.3.4 Tests de médicaments

En guise d'exemple pour introduire la notion de couplage, Lovász et Plummer [102] considèrent une entreprise pharmaceutique qui veut tester n antibiotiques sur n volontaires, en tenant compte des cas d'allergies. Tester un médicament une seule fois n'est cependant pas suffisant, et il serait plus réaliste de donner n médicaments à m cobayes, chaque médicament devant être donné un certain nombre de fois (à des cobayes différents). Puisque le but normal de telles expériences est de guérir les gens plus que de les rendre malades, aucun volontaire ne devrait recevoir trop de produits.

Il s'agit alors d'un problème $Bm|allow_j|H_{\max}$, où les boîtes sont les cobayes et les objets les médicaments. La hauteur d'un objet serait une mesure de sa dangerosité potentielle, et le vecteur $allow_j$ représenterait les incompatibilités allergiques.

1.3.5 Répartition de l'utilisation de machines

Dans [115] est décrit un problème de répartition de ressources, traité comme un problème $Bm|allow_j, h_j = 1|\cdot$. Il s'agit de faire un emploi du temps de l'affectation d'un terminal informatique. Chaque utilisateur (un objet O_j) a une demande ($size_j$ heures) et des disponibilités (représentées par $allow_j$). Les boîtes sont les heures ouvrables du terminal, et il s'agit de déterminer si un emploi du temps satisfaisant toutes les demandes est possible.

1.3.6 Développement logiciel

Pour un logiciel, plusieurs bibliothèques de programmes sont à écrire. Chacune peut être codée par un nombre quelconque de programmeurs et le temps pour la réaliser dépend, bien entendu, de ce nombre. De plus, même au sein d'une bibliothèque, les routines sont suffisamment indépendantes pour qu'il n'y ait pas besoin de synchroniser les développeurs.

Plusieurs objectifs peuvent être envisagés. Si l'on dispose de m programmeurs et que l'on veut finir le plus tôt possible, c'est un problème $Bm|any_j|H_{\max}$. Si, au contraire, une date limite est imposée et que l'on veut minimiser le nombre d'employés nécessaires pour la respecter, c'est un problème $B|any_j|m$.

1.3.7 Organisation d'une école d'été

Pour une école d'été, plusieurs cours doivent être sélectionnés et programmés durant une semaine. Chaque cours nécessite $size_j$ fois un créneau de h_j heures, et deux leçons d'un même cours ne doivent pas avoir lieu le même jour. Le but est de mettre en place autant de cours que possible.

Dans ce problème, les cours sont les objets et les jours de la semaine les boîtes (m jours de H heures). C'est alors un problème $B|size_j|N$, ou $B|size_j|W$ si certains cours ont priorité sur d'autres.

1.4 Envoi

Le concept de rangement d'objets multiboîtes vient d'être posé formellement et plusieurs exemples d'application montrent la pertinence pratique de ce modèle. Les chapitres suivants sont dédiés à l'analyse de ce concept ; il s'agira notamment de préciser la complexité des problèmes et de proposer des méthodes de résolution.

Modèles linéaires

Ce chapitre n'est pas une étude exhaustive des possibilités de résolution des problèmes multiboîtes par la programmation linéaire : une telle étude n'a pas été une priorité de mes travaux. Aussi le lecteur ne trouvera ici que des pistes de recherche, dont plusieurs mériteraient d'être explorées plus avant.

Afin de faire ressortir les idées essentielles la première section ne traite que du modèle le plus simple, $B|size_j|H_{\max}$. Une seconde section présente brièvement quelques adaptations possibles pour les autres problèmes, selon l'objectif et le type d'objets considérés.

2.1 Modèles pour $B|size_j|H_{\max}$

2.1.1 Un modèle en variables binaires

La manière la plus directe de modéliser un problème de rangement multiboîte par le biais de la programmation linéaire est d'utiliser des variables booléennes : x_{ij}, vraie si et seulement si l'objet O_j est rangé dans la boîte B_i. Pour les problèmes $B|size_j|H_{\max}$, le modèle est alors immédiat :

$$(B|size_j|H_{\max}) \quad \begin{cases} \min & H_{\max} & & \\ s.c. & \bullet \sum_{j=1}^{n} h_j\, x_{ij} \leq H_{\max} & i = 1, 2, \ldots, m \\ & \bullet \sum_{i=1}^{m} x_{ij} = size_j & j = 1, 2, \ldots, n \\ & \bullet\ x_{ij} \in \{0, 1\} & i = 1, 2, \ldots, m, j = 1, 2, \ldots, n \end{cases}$$

La première contrainte contrôle qu'aucune boîte ne dépasse la hauteur $H_{\max}$; la deuxième implique que chaque objet est complètement rangé ; la troisième assure que les bouts d'un même objet sont dans des boîtes distinctes.

À l'optimalité, la fonction-objectif a bien pour valeur la hauteur de la plus haute boîte. De plus, il n'est pas nécessaire de préciser d'autres contraintes sur la variable $H_{\max}$: à l'optimalité, elle est nécessairement entière et positive.

Avantages et inconvénients

Un intérêt évident de ce modèle est sa grande simplicité, qui permet de bien appréhender le problème. À remarquer cependant : si m n'est pas fixé, la taille du modèle ($nm + 1$ variables, $mn + m + n$ contraintes) n'est que pseudo-polynomiale en la taille de l'instance ($\mathcal{O}(n(\log m + \log h_{\max}))$).

Tel quel, ce modèle n'est pas adapté à une résolution réelle : il possède beaucoup trop de symétries (p.ex. on peut permuter les boîtes, ou des bouts d'objets identiques). Ces symétries peuvent toutefois être réduites (voir le modèle suivant) et même exploitées, ainsi que le fait que le polyèdre est en 0-1.

Relaxation continue

La résolution fractionnaire de ce modèle est immédiate : ainsi, mettre $size_j/m$ de chaque objet dans chaque boîte est une solution optimale pour la relaxation continue, de valeur $\sum_j size_j h_j/m$. Cela donne une borne inférieure, si l'on prend en compte l'entièreté :

Proposition 2.1. *La valeur optimale d'un problème $B|size_j|H_{\max}$ vérifie :*

$$H^*_{\max} \geq \left\lceil \frac{\sum_{O_j \in \mathbf{O}} size_j\, h_j}{m} \right\rceil,$$

où $\mathbf{O}$ *est l'ensemble des objets.*

Il serait toutefois intéressant d'explorer plus avant la résolution de la relaxation continue : en effet un algorithme de type Simplexe ne donnerait pas la solution triviale précédente, mais une solution (dite « solution de base ») n'ayant au plus que $m + n$ variables (parmi nm) fractionnaires. Des techniques d'arrondi seraient alors envisageables.

2.1.2 Un modèle en variables entières

Voici maintenant un deuxième modèle, non plus en 0-1 mais en variables entières. L'idée derrière ce modèle est de briser autant que possible les symétries du modèle précédent.

Briser les symétries dues aux objets identiques est aisé : on ne considère plus que des objets O_j distincts, caractérisés par une hauteur h_j et une largeur $size_j$; une variable supplémentaire n_j indique le nombre d'objets de type O_j. Ainsi, les variables de décision x_{ij} sont maintenant des variables entières dont la valeur est le nombre d'objets de type O_j mis dans la boîte B_i :

$$(B|size_j|H_{\max}) \quad \begin{cases} \min & H_{\max} \\ s.c. & \bullet \sum_{j=1}^{n} h_j\, x_{ij} \leq H_{\max} \quad i = 1, 2, \ldots, m \\ & \bullet \sum_{i=1}^{m} x_{ij} = n_j\, size_j \quad j = 1, 2, \ldots, n \\ & \bullet\ x_{ij} \leq n_j \qquad\qquad i = 1, 2, \ldots, m, j = 1, 2, \ldots, n \\ & \bullet\ x_{ij} \in \mathbb{N} \qquad\qquad i = 1, 2, \ldots, m, j = 1, 2, \ldots, n \end{cases}$$

La troisième contrainte assure que deux bouts d'un même objet ne seront pas mis dans la même boîte.

Élimination des symétries

Ce modèle brise les symétries dues aux objets ; pour celles dues aux boîtes, plusieurs coupes sont possibles. Les solutions peuvent, tout d'abord, être « normalisées ». On impose alors que les boîtes soient rangées par ordre croissant des tailles :

$$\sum_{j=1}^{n} h_j\, x_{i,j} \leq \sum_{j=1}^{n} h_j\, x_{i+1,j},\ 1 \leq i < n.$$

Ensuite, puisque dans une solution optimale la différence de hauteur entre deux boîtes est au plus $h_{\max}$, la contrainte suivante peut être ajoutée :

$$\sum_{j=1}^{n} h_j\, x_{m,j} - \sum_{j=1}^{n} h_j\, x_{1,j} \leq h_{\max}.$$

La contrainte précédente peut être généralisée : si un objet O_j est mis dans B_{i_1} mais pas dans B_{i_2} et que la différence de hauteur entre ces deux boîtes $\Delta_{i_1 i_2} \geq h_j$, alors O_j peut être déplacé de B_{i_1} dans B_{i_2} sans détériorer la solution. Ainsi la transformation suivante est valide :

1. définir les Y variables $y^j_{i_1 i_2}, i_1 < i_2$, représentant la différence de nombre de copies de O_j entre B_{i_1} et B_{i_2} ;
2. ajouter les contraintes : $\Delta_{i_1 i_2} \leq h_j\, y^j_{i_1 i_2}$, *i.e.* : $\sum_{l=1}^{n} h_l (x_{i_2 l} - x_{i_1 l}) \leq h_j\, y^j_{i_1 i_2}$;
3. ajouter les contraintes $x_{i_1 j} - x_{i_2 j} \leq y^j_{i_1 i_2}$, afin de calculer $y^j_{i_1 i_2}$;
4. la valeur des $y^j_{i_1 i_2}$ n'est correcte que si les contraintes de l'item précédent sont serrées ; pour cela, transformer la fonction-objectif en $G = H_{\max} + \varepsilon \sum y^j_{i_1 i_2}$. Pour ε « suffisamment petit » (p.ex. $\varepsilon = (h_{max} Y)^{-1}$), $\lfloor G \rfloor$ est la valeur optimale du problème initial.

Comportement expérimental

D'un point de vue expérimental, l'intérêt des contraintes précédentes n'a été étudié que sur quelques petites instances et aucune conclusion définitive ne ressort. Signalons toutefois les quelques tendances suivantes.

Tout d'abord, sur le modèle en 0-1, un algorithme standard[1] trouvait une solution optimale généralement rapidement (typiquement quelques secondes), mais passait alors énormément de temps à confirmer cette optimalité (typiquement plusieurs minutes). Le simple fait d'indiquer que la variable $H_{\max}$ est entière atténue un peu cette attitude ; supprimer les symétries dues aux objets grâce au modèle de cette section assure un gain encore plus conséquent.

Pour ce qui est des contraintes additionnelles : forcer les boîtes à être rangées dans l'ordre des tailles améliore bien les performances ; par contre l'intérêt d'imposer une différence de hauteur entre les boîtes inférieure à $h_{\max}$ n'est pas évident. Quant à la généralisation de cette contrainte par l'introduction des variables y_{ikj}, elle a permis, sur les instances testées, de réduire notablement le nombre de nœuds dans l'arbre de recherche, *i.e.* le nombre de programmes linéaires résolus, mais au prix d'un surcoût de calcul pour la résolution de ces systèmes globalement supérieur au gain.

Dans tous les cas, une étude plus précise et exhaustive est cependant indispensable avant de tirer des conclusions définitives.

Relaxation continue

La relaxation continue donne la même valeur pour ce second modèle que pour le premier (une solution optimale est donnée par $x_{ij} = n_j\, size_j/m$). Toutefois, la proportion de contraintes, par rapport aux variables, ayant augmenté, même un algorithme de type Simplex ne donnerait, *a priori*, que peu de variables entières ; aussi, malgré ses avantages, il n'est sans doute pas préférable au premier modèle pour appliquer des techniques d'arrondi de solutions fractionnaires.

2.2 Adaptations aux autres problèmes

Les modèles linéaires pour le problème $B|size_j|H_{\max}$ sont simples mais déjà peu propices, en l'état, pour une résolution directe. L'adaptation aux autres objectifs et/ou modèles renforce malheureusement l'aspect négatif : elle passe généralement par l'ajout de variables entières. Je propose ici certains ajustements à partir du modèle en variables en 0-1 pour $B|size_j|H_{\max}$.

Pour prendre en compte l'objectif W (ou N), il suffit de rajouter une variable booléenne par objet : r_j, valant 1 si et seulement si l'objet O_j est rangé. Ceci est pris en compte dans la deuxième contrainte, validant que l'objet est soit complètement rangé, soit pas du tout :

[1] J'ai utilisé CPLEX interfacé par MPL.

$$(B|size_j|W) \quad \begin{cases} \max & \displaystyle\sum_{j=1}^{n} r_j\, w_j & \\ s.c. & \bullet \displaystyle\sum_{j=1}^{n} h_j\, x_{ij} \leq H & i = 1, 2, \ldots, m \\ & \bullet \displaystyle\sum_{i=1}^{m} x_{ij} = r_j\, size_j & j = 1, 2, \ldots, n \\ & \bullet\ x_{ij} \in \{0, 1\} & i = 1, 2, \ldots, m, j = 1, 2, \ldots, n \\ & \bullet\ r_j \in \{0, 1\} & j = 1, 2, \ldots, n \end{cases}$$

De la même façon, pour l'objectif m, il suffit d'introduire une variable booléenne par boîte : b_i, valant 1 si et seulement si la boîte B_i est utilisée. Une borne supérieure m^+ pour le nombre de boîtes est alors nécessaire (p.ex. $m^+ = \sum_{j=1}^{n} size_j$).

$$(B|size_j|m) \quad \begin{cases} \min & \displaystyle\sum_{i=1}^{m^+} b_i & \\ s.c. & \bullet \displaystyle\sum_{j=1}^{n} h_j\, x_{ij} \leq H & i = 1, 2, \ldots, m^+ \\ & \bullet \displaystyle\sum_{i=1}^{m^+} x_{ij} = size_j & j = 1, 2, \ldots, n \\ & \bullet\ b_i \geq x_{ij} & i = 1, 2, \ldots, m^+, j = 1, 2, \ldots, n \\ & \bullet\ x_{ij} \in \{0, 1\} & i = 1, 2, \ldots, m^+, j = 1, 2, \ldots, n \\ & \bullet\ b_i \in \{0, 1\} & i = 1, 2, \ldots, m^+ \end{cases}$$

Pour ces deux modèles, les méthodes évoquées précédemment peuvent aussi être mises en œuvre afin de réduire les symétries.

Pour prendre en compte d'autres types d'objets comme set_j et any_j, il semble que, mis à part pour certains cas particuliers comme $allow_j$ (voir chapitre 6), une explosion du nombre de variables, booléennes ou entières, soit le prix à payer. Cela amène à des modèles à mon avis inopérants et inutilement compliqués pour permettre une meilleure appréhension du problème. Aussi, dans la suite de ce document, je ne considèrerai pas de tels modèles linéaires.

Complexité, approximabilité

S'il n'y a pas de solution, c'est qu'il n'y a pas de problème.
— proverbe Shadok

Ce chapitre a pour objectif de positionner les problèmes de rangements d'objets multiboîtes au sein de la hiérarchie des classes de complexité. Pour la définition des termes utilisés, le lecteur est renvoyé à l'annexe C. Une connaissance préalable de la théorie de la complexité est toutefois indispensable ; je conseille [60, 10] comme ouvrages de référence et d'apprentissage.

La première section présente la $\mathcal{NP}$-complétude pour les différents types d'objets ($size_j$, set_j, any_j) et un objectif quelconque. La deuxième section aborde brièvement la question de l'approximabilité des problèmes de rangements multiboîtes. Enfin, dans la troisième section, plusieurs variantes d'un même algorithme sont proposées pour résoudre polynomialement le cas d'objets de hauteur unitaire, pour les objectifs $H_{\max}$, m et N.

Une quatrième section présente une synthèse des résultats. Toutefois, de nombreux compléments et précisions sont apportés au cas par cas dans les chapitres ultérieurs. Pour une vision synthétique et globale de la complexité des problèmes de rangement d'objets multiboîtes, j'invite le lecteur à se reporter à l'annexe A.

3.1 Complexité des problèmes multiboîtes

Commençons par le plus intuitif des modèles de rangements d'objets multiboîtes, le cas $size_j$, et formulons-le de manière générale sous forme de problème de décision :

Problème B|size$_j$|·

Instance. Un ensemble de n objets $\mathbf{O} = \{O_j = (h_j, size_j, w_j), j = 1, 2, \ldots, n\} \subset \mathbb{N}^{3n}$; une borne $H \in \mathbb{N}$; une borne $m \in \mathbb{N}$; une borne $W \in \mathbb{N}$.

Question. Existe-t-il un sous-ensemble $\mathbf{O'}$ d'objets ($\mathbf{O'} = \{O'_j\} \subset \mathbf{O}$) dont le poids total dépasse W (*i.e.* $\sum_{O'_j \in \mathbf{O'}} w_j \geq W$) et tel qu'il existe un rangement de $\mathbf{O'}$ dans m boîtes $(B_1, B_2, \ldots, B_m)$ de hauteur H (*i.e.* $\sum_{O'_j \in B_i} h_j \leq H, \forall i = 1, 2, \ldots, m$ et $|\{B_i; O'_j \in B_i\}| = size_j, \forall j = 1, 2, \ldots, n$) ?

L'essentiel des résultats de complexité (sections 3.1 et 3.3) de ce chapitre a été publié dans [96] et a été présenté en diverses occasions [90, 97, 96].

Selon les objectifs, des simplifications interviennent, ce qui peut réduire la taille d'une instance : il n'y a de fonction de poids w_j que pour le critère W, tandis que pour les critères m et $H_{\max}$ la borne W disparaît elle aussi. Toutefois ces simplifications sont rarement significatives.

Le problème général a une taille $\mathcal{O}(n \ln(mHW))$. Or une solution (*i.e.* la donnée, pour chaque boîte, des objets qu'elle contient) a une taille $\mathcal{O}(nm)$, c'est-à-dire exponentielle en la taille de l'instance si le nombre de boîtes n'est pas fixé ! En fait, dans ce cas, aucun « bon certificat » (au sens de la théorie de la complexité) n'est connu pour attester d'une réponse affirmative au problème $B|size_j|\cdot$. En d'autres termes, on ne sait pas si ce problème est dans la classe $\mathcal{NP}$ ou pas... d'où une première question ouverte :

Question 3.1 : Le problème $B|size_j|\cdot$ appartient-il à $\mathcal{NP}$?

Toutefois, comme je le montre plus loin, le problème est au moins $\mathcal{NP}$-difficile ; en fait il est $\mathcal{NP}$-complet pour n'importe quel nombre fixé de boîtes $m \geq 2$.

Commençons par quelques cas particuliers, puisque nombreux sont ceux que l'on trouve dans la littérature. $B2|size_j|H_{\max}$ correspond au problème Bipartition ; $Bm|size_j = 1|H_{\max}$ est le problème d'Ordonnancement sur machines parallèles ($Pm||C_{\max}$) ; $B^1|size_j|W$ est le problème du Sac-à-dos. Tous ces problèmes sont $\mathcal{NP}$-complets, mais solubles en temps pseudo-polynomial [60, 10]. $B|size_j = 1|H_{\max}$ est le problème d'Ordonnancement sur machines parallèles ($P||C_{\max}$) et $B|size_j = 1|m$ est le problème de Bin-Packing. Ces deux problèmes sont $\mathcal{NP}$-complets au sens fort [60].

Étant donnés tous ces cas particuliers $\mathcal{NP}$-complets, le résultat suivant n'a rien de surprenant.

Théorème 3.1. *$Bm|size_j|\cdot$ est $\mathcal{NP}$-complet ; $B|size_j|\cdot$ est au moins $\mathcal{NP}$-difficile au sens fort.*

preuve — $Pm||C_{\max}$, problème $\mathcal{NP}$-complet, est une restriction polynomiale de $Bm|size_j|\cdot$ où seules sont admises les instances vérifiant : $W = n$, $H = C_{\max}$ et $\forall j = 1, 2, \ldots, n : size_j = 1, w_j = 1$. De ce fait, $Bm|size_j|\cdot$ est $\mathcal{NP}$-complet.

Considérons maintenant une instance de Partition en triplets : $3q$ objets de taille l_j et une borne X. La question est de savoir s'il est possible de ranger les objets dans q boîtes de manière à ce que la somme des tailles des objets dans chaque boîte soit exactement X.

Une instance de Partition en triplets se transforme de manière pseudo-polynomiale (selon le sens donné par Garey et Johnson [60, page 101] — voir annexe C) en une instance de $B|size_j|\cdot$ en fixant : $m = q$, $H = X$, $W = n$, $n = 3m$ et $\forall j = 1, 2, \ldots, n : h_j = l_j, size_j = 1, w_j = 1$. Comme Partition en triplets est $\mathcal{NP}$-complet au sens fort, $B|size_j|\cdot$ est au moins $\mathcal{NP}$-difficile au sens fort. ⊗

Rappelons ici que la formulation et la discussion ci-dessus sont générales pour n'importe lequel des objectifs $H_{\max}$, m, N et W et que même lorsque des simplifications s'opèrent (comme la disparition de la fonction de poids w_j et de la borne W) les résultats demeurent valides. Il faut toutefois se méfier puisque de telles simplifications peuvent avoir de l'importance : le problème du Sac-à-Dos, $B^1|size_j|W$, est $\mathcal{NP}$-complet alors que pour les mêmes contraintes, mais un autre objectif (N ou m), il devient trivial.

Passons maintenant à la généralisation de $size_j$: le cas any_j. Pour ce problème, on doit indiquer, pour chaque objet, sa hauteur en fonction de sa largeur. Aussi faut-il remplacer chaque couple $(h_j, size_j)$ par un vecteur de taille m et le problème devient :

Problème B|any$_j$|·

Instance. Une borne $m \in \mathbb{N}$; un ensemble de n objets $\mathbf{O} = \{O_j = ([h_j(s), s = 1, 2, \ldots, m], w_j), j = 1, 2, \ldots, n\} \subset \mathbb{N}^{(m+1)n}$; une borne $H \in \mathbb{N}$; une borne $W \in \mathbb{N}$.

Question. Existe-t-il un sous-ensemble $\mathbf{O}'$ d'objets ($\mathbf{O}' = \{O'_j\} \subset \mathbf{O}$) dont le poids total dépasse W (*i.e.* $\sum_{O'_j \in \mathbf{O}'} w_j \geq W$) et tel qu'il existe un rangement de $\mathbf{O}'$ dans m boîtes $(B_1, B_2 \ldots, B_m)$ de hauteur H (*i.e.* $\sum_{O'_j \in B_i} h_j(s_j) \leq H, \forall i = 1, 2, \ldots, m$, avec $s_j = |\{B_i; O'_j \in B_i\}|, \forall j = 1, 2, \ldots, n$) ?

La taille de ce problème est $\mathcal{O}(n(m \ln H + \ln W))$. Contrairement au problème $B|size_j|\cdot$, la taille d'une solution est bornée polynomialement par cette quantité. Ainsi, les problèmes $Bm|any_j|\cdot$ et aussi $B|any_j|\cdot$ appartiennent à la classe $\mathcal{NP}$. Ils sont en fait tous les deux $\mathcal{NP}$-complets :

Théorème 3.2. *$Bm|any_j|\cdot$ est $\mathcal{NP}$-complet ; $B|any_j|\cdot$ est $\mathcal{NP}$-complet au sens fort.*

preuve — Utilisons le fait que any_j généralise $size_j$ pour en déduire sa complexité. Rappelons que $Bm|size_j|\cdot$ est $\mathcal{NP}$-complet et $B|size_j|\cdot$ au moins $\mathcal{NP}$-difficile au sens fort (théorème 3.1).

Soit une instance I de $B|size_j|\cdot$. Celle-ci peut être transformée en une instance I' de $B|any_j|\cdot$ en posant, pour un objet $O_j = (h_j, size_j)$ de I : $h_j(s) = h_j$ si $s = size_j$ et $h_j(s) = +\infty$ sinon. Cette transformation n'est pas polynomiale (la taille de l'instance passe de $\mathcal{O}(n(\ln H + \ln m + \ln W))$ à $\mathcal{O}(n(m \ln H + \ln W))$), mais elle est suffisante pour nos besoins.

En effet, si $B|any_j|\cdot$ n'est pas $\mathcal{NP}$-complet au sens fort, alors il existe un algorithme pseudo-polynomial pour le résoudre. Par la transformation précédente, un tel algorithme résoudrait aussi $B|size_j|\cdot$ en un temps polynomial en n, H, m, W, ce que la forte $\mathcal{NP}$-difficulté de $B|size_j|\cdot$ interdit. Le problème $B|any_j|\cdot$ est donc $\mathcal{NP}$-complet au sens fort.

Une transformation, similaire à la précédente, d'une instance de $Bm|size_j|\cdot$ en une instance de $Bm|any_j|\cdot$ est polynomiale (m est fixé). Aussi, si $Bm|any_j|\cdot$ était polynomial, $Bm|size_j|\cdot$ le serait aussi, ce qui aboutirait à une contradiction. ⊗

Comme dans le cas des problèmes de type $size_j$, des simplifications des instances sont possibles pour certains objectifs, comme $H_{\max}$, m ou N. Les résultats précédents, pour des raisons identiques, restent valides.

Le cas des problèmes de type set_j est plus délicat : la complexité dépend grandement des possibles restrictions sur les ensembles de boîtes considérés, puisque cela peut modifier la taille d'une instance de manière exponentielle !

Par exemple, si tous les ensembles sont *a priori* possibles, il faut préciser, pour chaque objet, son appartenance aux $2^m - 1$ ensembles potentiels et la taille de l'instance explose en $\mathcal{O}(n(2^m + \ln(HW)))$. Bien sûr un tel cas n'aurait pas beaucoup de sens, ne serait-ce que parce que, pour chaque objet, seuls quelques ensembles seraient utiles (ceux n'étant inclus dans aucun autre). Il permet toutefois de douter de la pertinence d'exprimer la complexité de $B|set_j|\cdot$ dans toute sa généralité. Pour des cas particuliers, nous avons cependant les résultats suivants :

Proposition 3.1. *$Bm|set_j|\cdot$ est $\mathcal{NP}$-complet ; $B|set_j, compl|\cdot$ est $\mathcal{NP}$-complet (« compl » signifie que set_j se réduit, pour chaque objet, à deux ensembles complémentaires d'objets).*

PREUVE — Lorsque le nombre de boîtes m est fixé, la taille d'une instance de $Bm|set_j|\cdot$ est $\mathcal{O}(n \ln(HW))$. Il s'ensuit que le problème $Pm||C_{\max}$, $\mathcal{NP}$-complet, en est une restriction polynomiale : $Bm|set_j|\cdot$ est donc $\mathcal{NP}$-complet.

Le problème $B|set_j, compl|\cdot$ est connu dans la littérature sous le nom de RING LOADING PROBLEM (voir aussi chapitre 1, section 1.3.1). Schrijver, Seymour et Winkler ont montré sa $\mathcal{NP}$-complétude [114]. ⊗

Dans l'annexe A, le lecteur trouvera une table récapitulative des résultats de complexité. Certains des résultats sont des conséquences de la discussion précédente, mais l'on trouvera aussi de nombreux compléments provenant des sections et chapitres suivants.

3.2 Approximabilité des problèmes multiboîtes

Cette section entre peu dans les détails et ne donne que quelques résultats basiques et généraux sur l'approximabilité des problèmes de rangements multiboîtes. Dans le cas des problèmes $B|size_j|H_{\max}$, la question sera beaucoup plus approfondie dans la deuxième partie de cet ouvrage.

Tout d'abord, rappelons qu'un problème $\mathcal{NP}$-difficile au sens fort n'admet pas de FPTAS si $\mathcal{P} \neq \mathcal{NP}$ [60, 10]. Il découle alors immédiatement des résultats de la section précédente :

Proposition 3.2. *Si $\mathcal{P} \neq \mathcal{NP}$, alors les problèmes $B|size_j|\cdot$ et $B|any_j|\cdot$ n'admettent de FPTAS pour aucun des objectifs $H_{\max}$, m, N et W.*

Toutefois, si le nombre m de boîtes est fixé alors les problèmes correspondants, $Bm|size_j|H_{\max}$, $Bm|set_j|H_{\max}$ et $Bm|any_j|H_{\max}$, admettent un FPTAS (voir chapitre 5, section 5.4).

Un résultat classique de Garey et Johnson [60, théorème 6.6] peut être adapté afin d'éliminer la possibilité d'une approximation à un facteur additif constant près :

Proposition 3.3. *Si $\mathcal{P} \neq \mathcal{NP}$, alors aucun algorithme A ne résout en temps polynomial un problème $X = \alpha|\beta|H_{\max}$ avec la garantie :*

$$|H^A_{\max} - H^*_{\max}| \leq K$$

pour K constante fixée et $\alpha \in \{Bm, B\}$, $\beta \in \{size_j, set_j, any_j\}$.

PREUVE — Supposons qu'un tel algorithme A existe pour un des problèmes X considérés (p.ex. $X = B|size_j|H_{\max}$).

À partir d'une instance I, on construit une instance I' en remplaçant chaque hauteur h_j par $(K+1)h_j$ (K étant fixé, cette transformation est polynomiale). Les solutions de I' sont en bijection avec celles de I et la valeur de chacune est exactement $K+1$ fois plus grande que celle de la solution correspondante.

En résolvant I' avec A, on a la garantie : $|H^A_{\max}(I') - H^*_{\max}(I')| \leq K$, ce qui, puisque les hauteurs sont entières, implique que la solution correspondante pour I est optimale : l'algorithme A permet donc de résoudre optimalement I en temps polynomial, ce qui contredit la $\mathcal{NP}$-complétude du problème X, si $\mathcal{P} \neq \mathcal{NP}$. ⊗

Si l'approximation à un facteur additif constant près n'est pas possible, je propose au chapitre 8 plusieurs algorithmes polynomiaux pour $Bm|size_j|H_{\max}$ qui garantissent une approximation à un facteur additif près : la plus grande hauteur d'un objet de l'instance considérée.

En ce qui concerne l'approximation à un terme multiplicatif près, il suffit de remarquer que, pour le problème $Bm|size_j|H_{\max}$, la pire solution vaut au plus $mh_{\max} \leq mH^*_{\max}$; il en découle le résultat trivial suivant :

Proposition 3.4. *Tout algorithme résolvant $Bm|size_j|H_{\max}$ est une m-approximation.*

On peut toutefois faire beaucoup mieux. Ainsi, il existe des algorithmes très rapides pour résoudre ce même problème avec un facteur inférieur à 2. Il existe même des PTAS et des FPTAS pour ce problème. Pour plus de détails, le lecteur est renvoyé aux chapitres 5, 7, 8 et 9 traitant respectivement d'un FPTAS, de bornes inférieures et supérieures, d'algorithmes d'approximation et d'un PTAS pour $B|size_j|H_{\max}$.

L'approximabilité pour les autres objectifs n'a pas été particulièrement étudiée dans ce travail. Dans le cas $B|size_j|m$ les résultats existants sur le BIN-PACKING

impliquent qu'il n'est pas possible (à moins que $\mathcal{P} = \mathcal{NP}$) d'avoir une approximation à un facteur inférieur à 3/2. De bien meilleures garanties asymptotiques sont toutefois envisageables, notamment en se basant sur les nombreux résultats déjà établis pour le Bin-Packing (se référer à [37]).

3.3 Résolutions polynomiales pour $B|size_j, h_j = 1|\cdot$

Nous avons vu précédemment que les problèmes $B|size_j|\cdot$ sont $\mathcal{NP}$-difficiles pour chacun des quatre objectifs $H_{\max}$, m, N et W. Une restriction naturelle est de ne considérer que des objets de hauteur unitaire. Que deviennent alors ces problèmes ? Dans la plupart des cas, ils deviennent polynomiaux...

Ces résolutions polynomiales passent par ce que l'on peut voir comme une adaptation de l'algorithme de McNaughton pour l'ordonnancement avec préemptions $P|pmtn|C_{\max}$ [103, 16]. Cela permet de résoudre optimalement et en temps polynomial le problème $B|size_j, h_j = 1|\cdot$ pour trois objectifs : $H_{\max}$, m et N[1].

Dans les trois cas, la clé est que l'on connaît à l'avance le volume nécessaire pour ranger optimalement les objets. Ensuite, il suffit de remplir cet espace par des solutions particulières « par blocs » où les différentes parties d'un objet sont placées dans des boîtes adjacentes. En d'autres termes : pour chaque objet O_j il existe, dans une telle solution, un indice i_j tel que O_j est rangé dans les boîtes $B_{i_j}, B_{i_j+1}, \ldots, B_{i_j+size_j-1}$ (les indices doivent être compris modulo m). Grâce à ce type de solutions, il suffira de préciser un indice par objet pour préciser entièrement une solution.

Algorithme 3.1. Résolution de $B|size_j, h_j = 1|H_{\max}$

```
- entrées : ensemble d'objets O, nombre de boîtes m -
 i = 1
 pour O_j ∈ O faire
       i_j = i
       i = ((i + size_j − 1) mod m) + 1
```

Pour les objectifs $H_{\max}$ et m, un simple algorithme (l'algorithme 3.1) qui « déroule » les objets les uns à la suite des autres suffit. Il place O_1 dans les $size_1$ premières boîtes, puis O_2 dans les $size_2$ boîtes suivantes, et ainsi de suite. En fin de procédure, chaque objet occupe $size_j$ boîtes différentes (la solution est réalisable), et la différence de hauteur entre deux boîtes est au plus de 1, puisque chaque objet est de hauteur unitaire.

Cet algorithme construit une solution optimale pour $H_{\max}$ et m :

[1] La résolution polynomiale de $B|size_j|\cdot$ avait déjà été signalée pour $H_{\max}$ [96] et m [1].

Proposition 3.5. *$B|size_j, h_j = 1|H_{\max}$ se résout en temps $\mathcal{O}(n)$. La valeur optimale est :*

$$H^*_{\max} = \left\lceil \frac{\sum_{j=1}^{n} size_j}{m} \right\rceil .$$

PREUVE — La valeur annoncée est une borne inférieure évidente. Elle est atteinte par l'algorithme 3.1, puisque la différence de hauteur entre deux boîtes est au plus un. Cet algorithme est donc optimal pour $B|size_j, h_j = 1|H_{\max}$; de plus, son temps d'exécution est $\mathcal{O}(n)$. ⊗

Proposition 3.6. *$B|size_j, h_j = 1|m$ se résout en temps $\mathcal{O}(n)$. La valeur optimale est :*

$$m^* = \max \left\{ size_{\max} \; ; \; \left\lceil \frac{\sum_{j=1}^{n} size_j}{H} \right\rceil \right\} .$$

PREUVE — La valeur annoncée est une borne inférieure évidente. L'algorithme 3.1 construit une solution réalisable (dont les boîtes ne dépassent pas la borne H) si on lui donne pour m cette valeur. Cet algorithme est donc optimal pour $B|size_j, h_j = 1|m$; de plus, son temps d'exécution est $\mathcal{O}(n)$ (comme celui du calcul de m^*). ⊗

Le cas du critère N est légèrement différent : généralement, on ne peut pas ranger tous les objets. Il suffit cependant d'adapter légèrement l'algorithme précédent pour tenir compte de ce phénomène, et ranger en priorité les objets les plus utiles (pour l'objectif considéré) : les plus petits, puisque seul leur nombre compte. Ceci donne l'algorithme 3.2, qui est optimal pour $B|size_j, h_j = 1|N$.

Algorithme 3.2. Résolution de $B|size_j, h_j = 1|N$

- **entrées :** ensemble d'objets **O**, nombre de boîtes m, hauteur de boîte H -

1. trier les objets : $size_1 \leq size_2 \leq \ldots \leq size_n$
2. $i = 1$, $V = mH$
   pour $O_j \in \mathbf{O}$ faire
     si $size_j \leq V$ alors
       $i_j = i$
       $i = ((i + size_j - 1) \bmod m) + 1$
       $V = V - size_j$
     sinon STOP

Proposition 3.7. *$B|size_j, h_j = 1|N$ se résout en temps $\mathcal{O}(n\ln(n))$. La valeur optimale est :*

$$N^* = arg\max\left\{k \leq n | \sum_{j=1}^{k} size_j \leq mH; size_1 \leq size_2 \leq \ldots \leq size_n\right\}.$$

PREUVE — La valeur annoncée est celle trouvée par l'algorithme 3.2, en temps $\mathcal{O}(n\ln(n))$ (le tri des objets domine leur rangement). De plus, cette solution est optimale : puisque les objets les plus petits sont placés les premiers, il n'est pas possible de supprimer un ensemble d'objets pour le remplacer par un plus grand ensemble d'objets non rangés. ⊗

Pour de futures extensions, il faut remarquer que ce principe de ranger d'abord les plus petits objets est toujours valide pour le critère N : même si certaines solutions optimales ne le vérifient pas, il existe toujours une solution optimale ne contenant que les N^* plus petits objets.

Pour ce qui est du quatrième objectif, W, il est sans doute bon de rappeler que le problème $B^1|size_j, h_j = 1|W$ est le problème, $\mathcal{NP}$-difficile, du SAC-À-DOS. Ainsi, même avec la restriction additionnelle que les boîtes sont aussi de hauteur unitaire, le problème n'est pas soluble en temps polynomial (à moins, bien sûr, que $\mathcal{P} = \mathcal{NP}$).

3.4 En résumé

La complexité des principaux modèles de rangements d'objets multiboîtes a été présentée dans ce chapitre. Deux tendances sont à noter : lorsque les objets ne sont pas de hauteur unitaire, il y a peu d'espoir de trouver des cas polynomiaux ; de plus, le nombre de boîtes est un facteur plus critique que leur hauteur (ce phénomène se trouve déjà dans les cas limites de l'ORDONNANCEMENT SUR MACHINES PARALLÈLES par rapport au BIN-PACKING : tandis que le premier admet des PTAS même pour un nombre de machines arbitraire, le second n'admet d'approximations à moins de 3/2 qu'asymptotiques).

Les chapitres suivants précisent et complètent ces résultats. Pour une vue générale de la complexité et de l'approximabilité des problèmes multiboîtes, le lecteur est renvoyé à l'annexe A.

Une piste de recherche intéressante ressort de ce qui précède : l'étude des « solutions par blocs », où les bouts d'un même objet sont rangés dans des boîtes adjacentes. Cette particularité a déjà permis de résoudre polynomialement $B|size_j, h_j = 1|\cdot$ pour les critères $H_{\max}$, m et N, et cela même pour un nombre de boîtes arbitraire. Elle me semble notamment être la voie la plus prometteuse pour obtenir un « bon certificat du oui » afin de prouver que les problèmes $B|size_j|\cdot$ sont dans $\mathcal{NP}$ (si tant est qu'ils le sont).

Dans une optique de résolution pratique des problèmes multiboîtes, une telle propriété permettrait également de simplifier grandement les algorithmes et pourrait ainsi se révéler être une approche très efficace. Pour valider cela, il faudrait toutefois contrôler les pertes éventuelles ainsi introduites :

Question 3.2 : Que vaut la meilleure solution « par blocs » par rapport à une solution optimale ? et ainsi : quelles garanties peut-on avoir grâce à des solutions « par blocs » ?

Sur les liens entre rangements et ordonnancements

Autant que savoir, douter me plaît.
— Dante Alighieri

Ce court chapitre permet de préciser la relation qui existe entre les rangements d'objets multiboîtes et les ordonnancements de tâches multiprocesseurs. Le lien entre les deux problèmes est tout d'abord formalisé. Dans une deuxième section, le passage d'un ordonnancement à un rangement est étudié tandis que la troisième section est dédiée à la transformation inverse. Une brève conclusion présente enfin le parti pris vis-à-vis de l'utilisation des ordonnancements pour les rangements, et réciproquement.

4.1 Problèmes associés

La classification des rangements d'objets multiboîtes, adaptée de celle existant pour les ordonnancements de tâches multiprocesseurs, suggérait déjà une forte relation entre ces deux problèmes (voir chapitre 1, section 1.2). En fait, celui-là est une relaxation de celui-ci : la notion de temps disparaît pour les problèmes de rangement, et avec elle la notion de simultanéité d'exécution des différentes composantes d'une tâche.

Cette relation se définit en termes de « problèmes associés » :

Définition 4.1. (problèmes associés) *Soit $\mathcal{B}$ un problème d'objets multiboîtes et $\mathcal{P}$ un problème de tâches multiprocesseurs. $\mathcal{B}$ et $\mathcal{P}$ sont des* problèmes associés *s'ils correspondent à des modèles identiques lorsqu'on identifie machines et boîtes ainsi que tâches et objets (*i.e. *: mêmes contraintes (éventuelles) sur le nombre de boîtes/machines et leur taille ; mêmes contraintes sur les objets/tâches (taille, type) ; objectifs similaires).*

Ainsi $P|size_j|C_{\max}$ et $B|size_j|H_{\max}$ sont des problèmes associés, mais aucun de $Pm|size_j|C_{\max}$, $P|set_j|C_{\max}$ ni $P|size_j|m$ n'est associé à $B|size_j|H_{\max}$. Afin de mieux fixer les idées, la table 4.1 présente quelques exemples supplémentaires de problèmes associés.

Le contenu de ce chapitre, paru dans [96], a été présenté en diverses occasions [56, 90, 96].

Table 4.1 – Quelques exemples de problèmes associés.

rangements	**ordonnancements**
$B\|\|H_{\max}$	$P\|\|C_{\max}$
$Bm\|size_j\|H_{\max}$	$Pm\|size_j\|C_{\max}$
$B\|set_j, h_j = 1\|H_{\max}$	$P\|set_j, p_j = 1\|C_{\max}$
$B3\|\|\sum H_i$	$P3\|\|\sum C_i$

Si deux problèmes sont associés, alors une instance de l'un est aussi une instance de l'autre : un objet $O_j = (size_j, h_j)$ est aussi une tâche de largeur $size_j$ et de temps d'exécution $p_j = h_j$; m boîtes sont autant de processeurs ; la hauteur totale d'une boîte/machine est sa date de fin d'exécution ; *etc.*

Pour compléter cette présentation de problèmes associés, la table 4.2 donne les complexités comparées des principaux couples pour le critère $H_{\max}/C_{\max}$. Les résultats de complexité pour les problèmes de rangement sont prouvés dans ce document ; les références pour les ordonnancements sont données.

Table 4.2 – Complexités comparées de problèmes associés.
Les notations sont données dans l'annexe D.

Problème	**Rangement**	**Ordonnancement**	
$Bm\|size_j\|H_{\max}$, $m = 2, 3$	$\mathcal{NP}$-d, p$\mathcal{P}$	$\mathcal{NP}$-d, p$\mathcal{P}$	[44]
$Bm\|size_j\|H_{\max}$, $m = 4$	$\mathcal{NP}$-d, p$\mathcal{P}$	$\mathcal{NP}$-d	[44]
$Bm\|size_j\|H_{\max}$, $m \geq 5$	$\mathcal{NP}$-d, p$\mathcal{P}$	$\mathcal{NP}$-f	[44]
$B\|size_j = 1\|H_{\max}$	$\mathcal{NP}$-f	$\mathcal{NP}$-f	[60]
$Bm\|size_j \in \{1, m\}\|H_{\max}$	$\mathcal{NP}$-d, p$\mathcal{P}$	$\mathcal{NP}$-d, p$\mathcal{P}$	[60]
$B\|size_j \in \{1, k\}, h_j = 1\|H_{\max}$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	[13]
$B\|size_j \leq k, h_j = 1\|H_{\max}$, k fixé	$\mathcal{O}(n)$	$\mathcal{O}(n)$	[13]
$B\|size_j \leq k, h_j = 1\|H_{\max}$, k qcq	$\mathcal{O}(n)$	$\mathcal{NP}$-f	[13]
$B\|size_j, h_j = 1\|H_{\max}$	$\mathcal{O}(n)$	$\mathcal{NP}$-f	[13]
$Bm\|any_j\|H_{\max}$, $m = 2, 3$	$\mathcal{NP}$-d, p$\mathcal{P}$	$\mathcal{NP}$-d, p$\mathcal{P}$	[44]
$Bm\|any_j\|H_{\max}$, $m = 4$	$\mathcal{NP}$-d, p$\mathcal{P}$	$\mathcal{NP}$-d	[44]
$Bm\|any_j\|H_{\max}$, $m \geq 5$	$\mathcal{NP}$-d, p$\mathcal{P}$	$\mathcal{NP}$-f	[44]
$B\|any_j\|H_{\max}$	$\mathcal{NP}$-f	$\mathcal{NP}$-f	[44]
$Bm\|fix_j\|H_{\max}$, $m \geq 3$	$\mathcal{O}(n)$	$\mathcal{NP}$-f	[15]
$Bm\|fix_j, h_j = 1\|H_{\max}$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	[15]
$B\|fix_j, h_j = 1\|H_{\max}$	$\mathcal{O}(n)$	$\mathcal{NP}$-f	[15]
$B2\|set_j\|H_{\max}$	$\mathcal{NP}$-d, p$\mathcal{P}$	$\mathcal{NP}$-d, p$\mathcal{P}$	[15]
$Bm\|set_j\|H_{\max}$	$\mathcal{NP}$-d, p$\mathcal{P}$	?	
$Bm\|set_j, h_j = 1\|H_{\max}$	polynomial	?	

4.2 D'un ordonnancement à un rangement

Prenons une solution, représentée par son diagramme de Gantt, d'un problème d'ordonnancement. Tournons cette solution de 90^o et laissons les tâches tomber et boucher les temps morts. Le résultat est un rangement, qui est une solution réalisable pour le problème de rangement associé au problème d'ordonnancement. En d'autres termes : en supprimant les temps morts intermédiaires d'un ordonnancement, on obtient un rangement valide. La figure 4.1 illustre cette transformation, donnée formellement par la définition suivante :

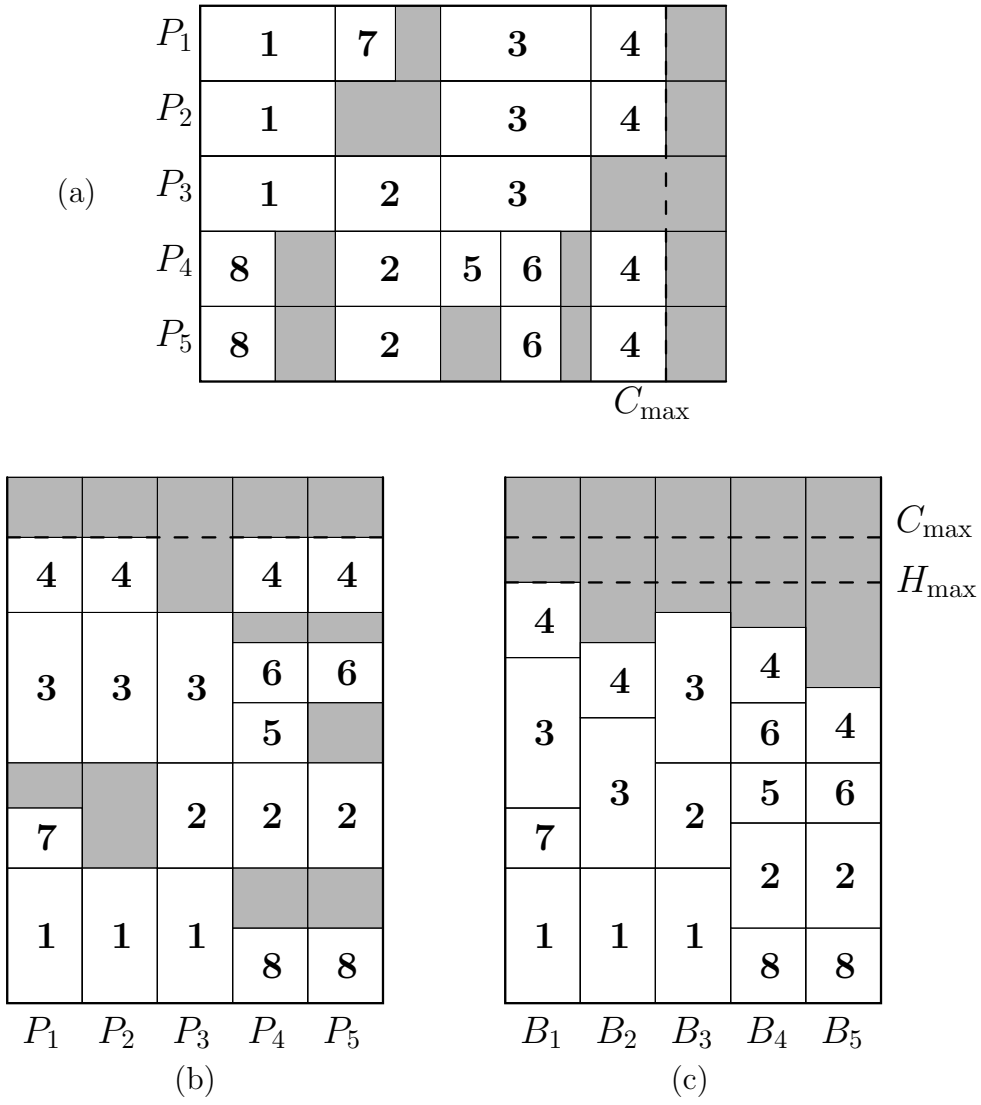


Figure 4.1 – Un exemple d'abaissement d'un ordonnancement.
(a) un ordonnancement ; (b) le même, tourné ; (c) le rangement associé.

Définition 4.2. (abaissement) *Soit I une instance pour deux problèmes associés $\mathcal{P}$ et $\mathcal{B}$; soit S un ordonnancement de I. L'*abaissement de S, $S^{\downarrow}$, *est défini comme étant le rangement ayant les mêmes affectations,* i.e. *qu'un objet O_j est rangé dans la boîte B_i pour $S^{\downarrow}$ si et seulement si la tâche T_j est effectuée par la machine P_i pour S.*

On peut remarquer qu'un abaissement est toujours une solution réalisable pour le problème vu en tant que problème de rangement. Il peut toutefois y avoir des pertes lors de la transformation : même si l'ordonnancement initial est optimal, son abaissement n'est pas nécessairement un rangement optimal, comme l'illustre la figure 4.2

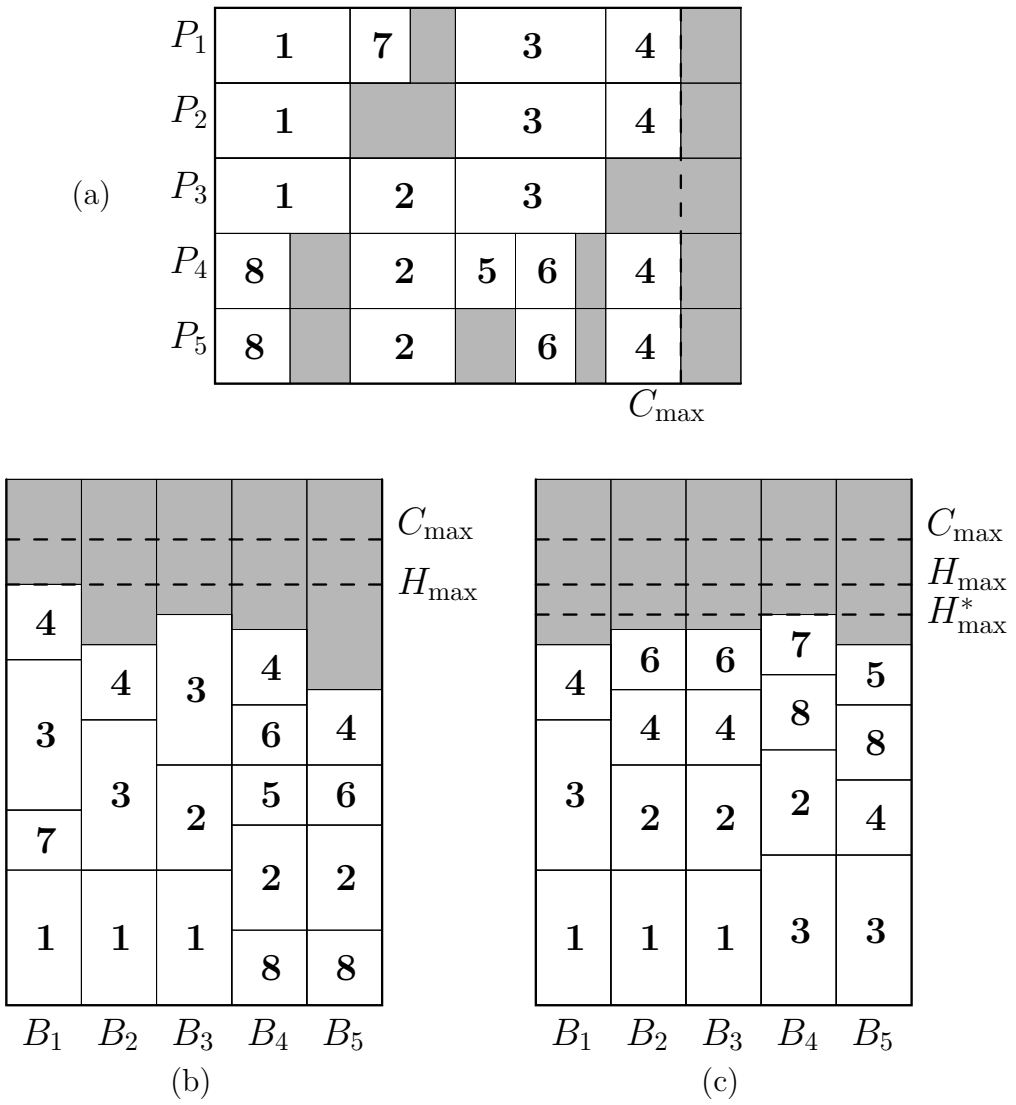


Figure 4.2 – Un exemple d'abaissement non optimal.
(a) un ordonnancement optimal ; (b) l'abaissement de cet ordonnancement ; (c) un rangement optimal.

Il existe toutefois des cas où un abaissement est toujours optimal (par exemple, dans le cas de $P|size_j = 1|C_{\max}$ où ordonnancements et rangements sont identiques) et des cas où, à défaut d'avoir l'optimalité, au moins a-t-on des garanties. La sous-section suivante présente un tel cas.

4.2.1 Garanties pour un abaissement de $Pm|size_j, p_j = 1|C_{\max}$

Dans cette section, je me préoccupe de problèmes de type $Pm|size_j, p_j = 1|C_{\max}$, c'est-à-dire avec un nombre fixé de machines/boîtes, des tâches/objets de largeur fixe et de hauteur unitaire, et en minimisant la hauteur/durée totale.

Rappelons tout d'abord que, d'un point de vue ordonnancement, ce problème se résout en temps linéaire $\mathcal{O}(n)$ [13]. D'un point de vue rangement, ce problème se résout également optimalement en temps linéaire (voir chapitre 3, section 3.3). Cette section ne donne donc pas un moyen de résoudre les problèmes $Bm|size_j, h_j = 1|H_{\max}$, par ailleurs déjà très bien résolus. Elle permet simplement d'illustrer le genre de garanties auxquelles on peut s'attendre lorsqu'on passe d'un ordonnancement à un rangement.

Pour le reste de cette section, I est une instance (de $Pm|size_j, p_j = 1|C_{\max}$ ou de $Bm|size_j, h_j = 1|H_{\max}$, indifféremment) ; $H^*_{\max}$ est la valeur optimale pour I vue en tant que rangement ; $C^*_{\max}$ est la valeur optimale pour I vue en tant qu'ordonnancement ; $S^{\downarrow}$ est l'abaissement d'un ordonnancement optimal et sa valeur est $H^{\downarrow}_{\max}$.

Proposition 4.1. *Avec les conventions précédentes :*

$$H^{\downarrow}_{\max} \leq \frac{2m}{m+1} H^*_{\max} + \frac{m-1}{m+1},$$

et cette borne est asymptotiquement atteinte.

PREUVE — Soit $A = \sum_j size_j$ le temps total d'activité des processeurs. Considérons un ordonnancement optimal S^*. Comme chaque tâche est de durée unitaire, il y a au plus un intervalle de temps $t \in \{1, 2, \ldots, C^*_{\max}\}$ durant lequel il y a strictement moins de $\lfloor m/2 \rfloor + 1$ processeurs actifs (autrement, les tâches effectuées dans ces deux intervalles pourraient être exécutées en parallèle, pour un gain de 1). Aussi, au moins $\lfloor m/2 \rfloor + 1$ machines sont actives pendant au moins $C^*_{\max} - 1$ unités de temps. Lors de la dernière unité de temps, au moins une machine est active. Il en découle :

$$\begin{aligned} A &\geq (C^*_{\max} - 1)(\left\lfloor \frac{m}{2} \right\rfloor + 1) + 1 \\ &\geq (C^*_{\max} - 1)\frac{m+1}{2} + 1, \end{aligned}$$

soit :

$$A \geq \left(C^*_{\max} - \frac{m-1}{m+1} \right) \frac{m+1}{2}. \qquad (\heartsuit)$$

A est aussi le volume total des objets. Comme chaque objet est rangé, on a :

$$H^*_{\max} \geq \frac{A}{m}. \qquad (\diamondsuit)$$

En intégrant de plus le fait que $S^{\downarrow}$ est une relaxation de S^*, et donc que $H^{\downarrow}_{\max} \leq C^*_{\max}$, on obtient grâce à $(\diamondsuit)$ et $(\heartsuit)$:

$$H^*_{\max} \geq \frac{1}{m}\left(H^{\downarrow}_{\max} - \frac{m-1}{m+1}\right)\frac{m+1}{2},$$

c'est-à-dire :

$$H^{\downarrow}_{\max} \leq \frac{2m}{m+1}H^*_{\max} + \frac{m-1}{m+1}.$$

Voici maintenant une famille d'instances qui atteint, asymptotiquement, cette borne. Pour cela, prenons m impair et n tel que $n \equiv 0 \bmod 2m$ et définissons $I_{m,n}$ comme ayant n objets identiques de hauteur $h_j = 1$ et de largeur $size_j = \left\lfloor \frac{m}{2} \right\rfloor + 1 = \frac{m+1}{2}$ (m est impair).

Pour une telle instance, deux tâches ne peuvent pas être effectuées simultanément. La solution S^* consistant à effectuer la tâche T_j au temps j sur les $size_j = \frac{m+1}{2}$ premières machines est donc optimale, et $C^*_{\max} = n$. L'abaissement de S^* n'apporte aucun gain : $S^{\downarrow} = S^*$, alors que la solution optimale pour un rangement a pour valeur $H^*_{\max} = \left\lceil \frac{n\frac{m+1}{2}}{m} \right\rceil = \frac{n(m+1)}{2m}$ pour $n \equiv 0 \bmod 2m$ (voir chapitre 3, section 3.3 ainsi que la figure 4.3). Aussi avons nous :

$$H^{\downarrow}_{\max} = \frac{2m}{m+1}H^*_{\max},$$

ce qui permet d'approcher la borne de garantie d'aussi près que l'on veut, en faisant tendre m vers l'infini. $\otimes$

4.2.2 Quelques remarques

Dans cette section, j'ai présenté quelques résultats visant à illustrer le passage d'un ordonnancement à un rangement. Ils n'incitent pas à continuer dans cette voie, qui semble peu adéquate.

Ainsi, même sur un cas simple et bien résolu ($Bm|size_j, h_j = 1|H_{\max}$), relâcher un ordonnancement optimal peut donner des rangements deux fois trop grands. Il ne faut donc pas trop compter sur de bonnes garanties pour des cas plus compliqués.

Ces conclusions pessimistes n'ont toutefois rien de surprenant : comme l'illustre le début de ce chapitre (table 4.2), les problèmes d'ordonnancement sont plus difficiles que les problèmes de rangements associés.

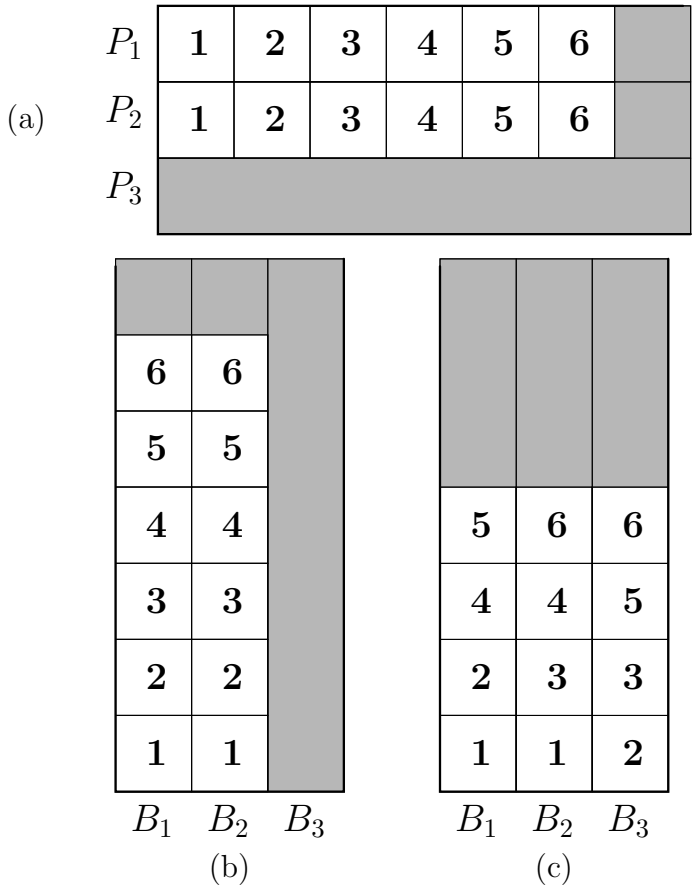


Figure 4.3 – Un exemple de mauvais abaissement pour $Pm|size_j, p_j = 1|C_{\max}$.
(a) un ordonnancement optimal ; (b) son abaissement ; (c) un rangement optimal.

4.3 D'un rangement à un ordonnancement

Je considère maintenant le problème inverse à celui de la section précédente : ayant un rangement, je veux le transformer en un ordonnancement valide. Formellement :

Définition 4.3. (mise-à-niveaux) *Soit I une instance et S un rangement réalisable pour I. Une* mise-à-niveaux de S *est un ordonnancement $S^{\uparrow}$, réalisable, tel que son abaissement soit S.*

De plus, une mise-à-niveaux $S^{\uparrow *}$ est optimale *si elle est le meilleur ordonnancement parmi les mises-à-niveaux de S.*

Une mise-à-niveaux est toujours possible : il suffit, par exemple, de mettre l'objet O_1 au début des machines, puis l'objet O_2 après lui, *etc.* En fait, il existe une correspondance entre les ordres sur les objets et les mises-à-niveaux : chaque ordre implique une mise-à-niveaux précise (si aucun temps mort n'est inséré, inutilement, entre les tâches). Trouver le bon ordre est alors essentiel : un mauvais choix a des effets catastrophiques, le pire ordonnancement pouvant être obtenu. Aussi je m'intéresse aux mises-à-niveaux optimales.

Dans le cas de deux boîtes, tout se passe bien, comme l'illustre la figure 4.4 et l'exprime la propriété suivante :

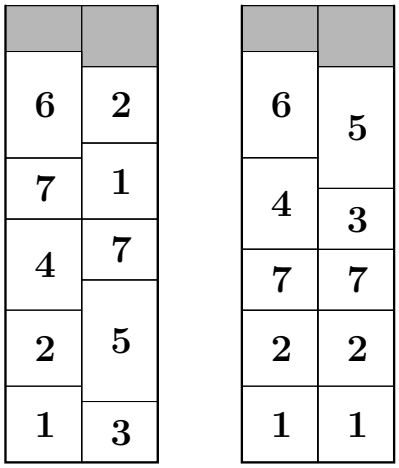


Figure 4.4 – Un exemple de mise-à-niveaux optimale pour $B2|size_j|H_{\max}$.
(a) un rangement ; (b) une mise-à-niveaux optimale.

Proposition 4.2. *Une mise-à-niveaux optimale de $B2|size_j|H_{\max}$ — ou plus généralement $Bm|size_j \in \{1, m\}|H_{\max}$ — se calcule en temps $\mathcal{O}(n \log n)$.*

PREUVE — Un objet a besoin soit de toutes les boîtes, soit d'une seule. Soit l'ordre (non strict) $\leq$ défini par $O_{j_1} \leq O_{j_2}$ si, et seulement si, O_{j_1} a besoin de plus de boîtes que O_{j_2}. Trions, au sein de chaque boîte, les objets selon cet ordre. Cela se fait en temps $\mathcal{O}(n \log n)$. D'un point de vue rangement, rien n'a changé. D'un point de vue ordonnancement, la solution est maintenant réalisable : c'est une mise-à-niveaux, nécessairement optimale puisque de même valeur que le rangement associé. $\otimes$

Malheureusement, dès trois boîtes, les choses se gâtent. En fait, le problème de la mise-à-niveaux optimale est identique aux problèmes d'ordonnancement de tâches pour lesquelles l'allocation est fixée (les problèmes de type $P|fix_j|C_{\max}$). Il s'ensuit le résultat suivant :

Proposition 4.3. *Trouver une mise-à-niveaux optimale est $\mathcal{NP}$-difficile, même pour 3 boîtes ou bien pour des objets de hauteurs unitaires.*

PREUVE — D'un point de vue rangement, un problème de type $B|fix_j|H_{\max}$ est facile : il n'y a aucun choix et l'instance est elle-même la solution. Par contre, vus en tant que problèmes d'ordonnancement, $P3|fix_j|C_{\max}$ et $P|fix_j, p_j = 1|C_{\max}$ sont $\mathcal{NP}$-difficiles (au sens fort) [15]. Un algorithme polynomial pour la mise-à-niveaux optimale permettrait donc de résoudre en temps polynomial ces problèmes, ce qui est impossible (à moins que $\mathcal{P} = \mathcal{NP}$). $\otimes$

4.4 Parti pris

Ordonnancements de tâches multiprocesseurs et rangements d'objets multiboîtes sont indubitablement liés. Toutefois, il semble que l'utilisation directe de l'un pour résoudre l'autre ne soit pas très prometteuse : des ordonnancements optimaux, bien qu'obtenus avec difficultés, ne permettent pas d'obtenir des rangements avec des garanties exceptionnelles[1] ; quant au passage inverse, il apparaît tout aussi problématique.

En définitive, les fortes relation et interconnexion entre les problèmes de rangements et les problèmes d'ordonnancements ne semblent pas devoir être utilisées directement, la solution d'un problème servant de base à la résolution de l'autre. C'est en fait dans l'adaptation des idées et des méthodes que l'échange sera, par contre, fructueux. C'est l'approche adoptée dans les chapitres suivants.

[1] En effet, les heuristiques du chapitre 8 impliquent des garanties meilleures, pour des cas plus généraux et pour un prix calculatoire bien moindre.

Le cas $Bm|any_j|\cdot$

Avoir entendu sans retenir ne fait pas de la science.
— Dante Alighieri

Il a été montré au chapitre 3 que les problèmes de type $B|any_j|\cdot$ sont $\mathcal{NP}$-difficiles pour les objectifs $H_{\max}$ et W. Dans ce chapitre, je propose des algorithmes pseudo-polynomiaux pour les résoudre lorsque le nombre de boîtes est fixé, *i.e.* pour résoudre les problèmes $Bm|any_j|H_{\max}$, et $Bm|any_j|W$. En découlent les résolutions de plusieurs cas particuliers, notamment pour les modèles $size_j$ et set_j et pour l'objectif N, ainsi qu'un FPTAS pour $Bm|any_j|H_{\max}$.

5.1 Principes généraux ; le cas $Bm|any_j|H_{\max}$

Pour le problème $Bm|any_j|H_{\max}$, chaque objet peut être mis dans un nombre quelconque de boîtes et sa hauteur dépend de ce nombre. Le but est de minimiser la hauteur de la plus grande boîte. Une approche par programmation dynamique, permet une résolution en temps pseudo-polynomial pour tout m fixé.

Lorsque m est deux ou trois, des résultats similaires existent pour l'ordonnancement de tâches multiprocesseurs [44, 15] et l'algorithme pseudo-polynomial est essentiellement le même que celui proposé par Du et Leung [44]. Toutefois leur approche, basée sur l'existence d'ordonnancements « canoniques », ne s'étend pas à des valeurs de m plus grandes (pour être précis : le cas $P4|any|C_{\max}$ est ouvert, mais $P5|any|C_{\max}$ est $\mathcal{NP}$-difficile au sens fort). Dans le cas du rangement d'objets, dans la mesure ou l'ordre au sein d'une boîte est indifférent, tout rangement est en quelque sorte « canonique », ce qui permet à l'approche d'être valide pour m quelconque, fixé.

Pour commencer, afin d'expliciter les idées, prenons le cas de deux boîtes. Chaque objet peut être placé selon trois configurations : dans la première boîte, dans la seconde, ou dans les deux en même temps (voir figure 5.1). Dans les deux premiers cas, il occupera une hauteur $h_j(1)$ plus grande (*a priori*) que dans le troisième cas, où il occupera une hauteur $h_j(2)$. Il « suffit » donc, pour chacun des trois choix, de compléter optimalement avec les autres objets et de choisir la meilleure solution.

La résolution pseudo-polynomiale pour le critère $H_{\max}$ a été publiée dans [96] et a été présentée en diverses occasions [90, 96].

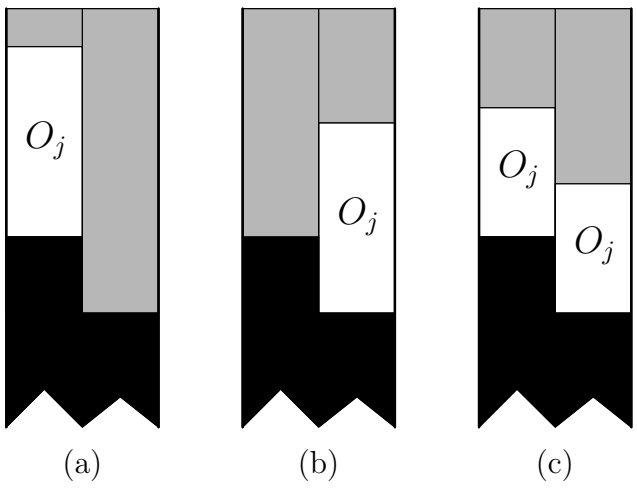


Figure 5.1 – Principe de résolution de $B2|any_j|H_{\max}$.
L'objet O_j est rangé dans (a) B_1, (b) B_2 ou (c) B_1 et B_2.

Ceci est fait récursivement par le biais des relations suivantes :

$$\begin{array}{lrcl} j=1,2,\ldots,n: & P(j,H_1,H_2) & = & \min\{P(j+1,H_1+h_j(1),H_2),\\ & & & \quad P(j+1,H_1,H_2+h_j(1)),\\ & & & \quad P(j+1,H_1+h_j(2),H_2+h_j(2))\}\\ j=n+1: & P(j,H_1,H_2) & = & \max\{H_1,H_2\} \end{array}$$

Dans ces équations, $P(j,H_1,H_2)$ représente la meilleure valeur possible (*i.e.* la plus petite hauteur de boîte) pour ranger les objets $j,j+1,\ldots,n$, sachant que les boîtes sont déjà remplies respectivement par les hauteurs H_1 et H_2. La valeur optimale cherchée est ainsi $P(1,0,0)$. Il s'ensuit le résultat suivant :

Lemme 5.1. *Le problème $B2|any_j|H_{\max}$ se résout à l'optimum en temps pseudo-polynomial $\mathcal{O}(nA^2)$, où $A=\sum_{j=1}^{n}\max(h_j(1),h_j(2))$.*

PREUVE — La discussion précédente justifie la validité des équations. Il reste à évaluer le temps de calcul, donc la taille du tableau P.

Le tableau P est de taille $\mathcal{O}(nA^2)$, où A est une borne supérieure sur la hauteur maximale d'une solution. $A=\sum_{j=1}^{n}\max(h_j(1),h_j(2))$ est une telle borne. Aussi, pour résoudre $B2|any_j|H_{\max}$ à l'optimum, il suffit de calculer P, ce qui est effectué par les algorithmes 5.1 ou 5.2 en temps $\mathcal{O}(nA^2)$. ⊗

La même démarche se généralise directement pour plus de deux boîtes. Nous obtenons ainsi le théorème suivant :

Théorème 5.1. *Pour tout entier m fixé, le problème $Bm|any_j|H_{\max}$ se résout en temps pseudo-polynomial $\mathcal{O}((2^m-1)nA^m)$, où $A=\sum_{j=1}^{n}\max_{k=1..m}h_j(k)$.*

Algorithme 5.1. Un algorithme récursif pour $B2|any_j|H_{\max}$

- **entrées :** ensemble d'objets $\mathbf{O}$, nombre de boîtes m -
ranger(j, H_1, H_2)
  si $j > n$ renvoyer $\max(H_1, H_2)$ — *c'est fini*
  si $P(j, H_1, H_2) > 0$ renvoyer $p(j, H_1, H_2)$ — *déjà connu*
  $x =$ ranger$(j+1, H_1 + h_j(1), H_2)$ — *calcul de* $p(j, H_1, H_2)$
  $y =$ ranger$(j+1, H_1, H_2 + h_j(1))$
  $z =$ ranger$(j+1, H_1 + h_j(2), H_2 + h_j(2))$
  $P(j, H_1, H_2) = \min(x, y, z)$
  renvoyer $P(j, H_1, H_2)$

renvoyer ranger$(1, 0, 0)$

Algorithme 5.2. Un algorithme itératif pour $B2|any_j|H_{\max}$

- **entrées :** ensemble d'objets $\mathbf{O}$, nombre de boîtes m -
1. pour $H_1 = 0$ à $A-1$ et $H_2 = 0$ à $A-1$ faire — *initialisation*
   $P(n, H_1, H_2) = \max(H_1, H_2)$

2. pour $j = n-1$ à 1 faire — *calcul*
   pour $H_1 = 0$ à $M-1$ et $H_2 = 0$ à $M-1$ faire
   $P(j, H_1, H_2) = \min(P(j+1, H_1 + h(j,2), H_2 + h(j,2)),$
   $P(j+1, H_1 + h(j,1), H_2),$
   $P(j+1, H_1, H_2 + h(j,1)))$
3. renvoyer $P(1, 0, 0)$

PREUVE — Les mêmes principes que pour le lemme 5.1 s'appliquent. On utilise maintenant $P(j, H_1, H_2, \ldots, H_m)$ et on a, pour toute hauteur $H_i \in \{1, 2, \ldots, A\}$, les équations suivantes :

$$\begin{aligned} j = 1, 2, \ldots, n: \quad & P(j, H_1, \ldots, H_m) = \min_K P(j+1, H_1 + h_j(|K|)\delta_{1K}, \ldots \\ & \qquad \ldots, H_m + h_j(|K|)\delta_{mK}) \\ j = n+1: \quad & P(j, H_1, \ldots, H_m) = \max_{i=1,2,\ldots,m} H_i \end{aligned}$$

où K est un ensemble non vide de boîtes, $|K|$ la cardinalité de cet ensemble, et δ_{iK} est 1 si B_i appartient à K, 0 sinon.

Là encore, le but est de calculer $P(1, 0, \ldots, 0)$. Ceci peut être fait itérativement en temps $\mathcal{O}(nA^m)$, puisque chaque itération (la recherche du minimum de $2^m - 1$ éléments) s'effectue en temps constant pour m fixé. Plus précisément, l'algorithme a un temps d'exécution en $\mathcal{O}((2^m - 1)nA^m)$. ⊗

5.2 Adaptation au cas $Bm|any_j|W$

Dans le cas d'un problème $Bm|any_j|W$, les objets sont de même type que pour un problème $Bm|any_j|H_{\max}$, mais l'objectif est maintenant de maximiser le nombre, pondéré, d'objets rangés dans m boîtes de hauteur H. La démarche adoptée pour la résolution de $B|any_j|H_{\max}$ s'adapte très bien à ce cas. Ainsi obtenons-nous un résultat semblable :

Théorème 5.2. *Pour tout entier m fixé, le problème $Bm|any_j|W$ se résout en temps pseudo-polynomial $\mathcal{O}(n(2H)^m)$.*

PREUVE — Définissons $Q(j, \bar{H}_1, \bar{H}_2, \ldots, \bar{H}_m)$ comme étant le plus grand nombre pondéré d'objets de $\{O_j, O_{j+1}, \ldots, O_n\}$ qui puissent être rangés, sachant qu'il reste dans la boîte B_i la hauteur $\bar{H}_i$. Il faut calculer $Q(1, H, \ldots, H)$, ce qui s'effectue *via* les équations suivantes :

$$\begin{aligned} j = 1, 2, \ldots, n: \quad & Q(j, \bar{H}_1, \ldots, \bar{H}_m) = \max_K \delta_K w_j + Q(j+1, \bar{H}_1 - h_j(|K|)\delta_{1K}, \ldots \\ & \qquad \ldots, \bar{H}_m - h_j(|K|)\delta_{mK}) \\ j = n+1: \quad & Q(j, \bar{H}_1, \ldots, \bar{H}_m) = 0 \end{aligned}$$

où K est un sous-ensemble de $\{1, 2, \ldots, m\}$ dans lequel O_j peut être rangé (c'est-à-dire tel que : $\forall B_i \in K : \bar{H}_i \geq h_j(|K|)$) ; $|K|$ est la cardinalité de cet ensemble ; δ_{iK} est 1 si $B_i \in K$ et 0 sinon ; δ_K est 0 si K est vide, 1 sinon ; et w_j est le poids accordé à l'objet O_j. Cette relation signifie simplement qu'un objet O_j peut soit ne pas être rangé (cela correspond à $K = \emptyset$ et $\delta_K = 0$), soit être rangé dans l'un des ensembles de boîtes qui peut le contenir.

Calculer $Q(1, H, \ldots, H)$ itérativement mène à un algorithme en temps $\mathcal{O}(nH^m)$, puisque chaque itération se fait en temps constant pour m fixé. Plus précisément, une itération coûte $\mathcal{O}(2^m)$ opérations (trouver le maximum de 2^m entiers), et donc l'algorithme a un temps d'exécution en $\mathcal{O}(n(2H)^m)$. ⊗

5.3 Quelques conséquences

Les deux théorèmes précédents — théorèmes 5.1 et 5.2 — ne rendent en fait pas complètement compte de la généralité des équations de récurrence qui les soutiennent. Ainsi, il n'est fait aucune hypothèse particulière sur la hauteur des objets, aussi celle-ci peut-elle varier d'un ensemble de boîtes à l'autre, même pour des ensembles de même cardinalité. En conséquence, les mêmes principes permettraient de résoudre le cas, par exemple, de boîtes « dédiées » où la hauteur d'un objet dépendrait non seulement du nombre de boîtes utilisées pour le ranger, mais aussi des boîtes elles-mêmes.

En particulier, les algorithmes précédents s'adaptent très bien au cas de hauteurs infinies, ce qui revient à interdire certains ensembles de boîtes. Il est donc possible de les utiliser pour le cas où seuls les ensembles de cardinalité $size_j$ sont autorisés (problème $Bm|size_j|\cdot$) ou bien lorsque seuls certains ensembles spécifiés sont possibles (problème $Bm|set_j|\cdot$).

De plus, dans ces deux cas, la restriction à des hauteurs fixes est pertinente, et les algorithmes deviennent alors polynomiaux[1]. Pour résumer, on a le résultat suivant, pour le critère $H_{\max}$:

Théorème 5.3. *Les problèmes $Bm|size_j|H_{\max}$ et $Bm|set_j|H_{\max}$, pour tout entier m fixé, se résolvent en temps pseudo-polynomial. Les problèmes $Bm|size_j, h_j = 1|H_{\max}$ et $Bm|set_j, h_j = 1|H_{\max}$ se résolvent en temps polynomial.*

Ce qui précède est également valable pour le critère W ; de plus l'algorithme reste pseudo-polynomial pour une fonction de poids constante, c'est-à-dire pour le critère N.

Théorème 5.4. *Pour tout entier m fixé, les problèmes $Bm|size_j|W$, $Bm|set_j|W$, $Bm|size_j|N$, $Bm|set_j|N$ et $Bm|any_j|N$ se résolvent en temps pseudo-polynomial. Les problèmes $Bm|size_j, h_j = 1|W$, $Bm|set_j, h_j = 1|W$, $Bm|size_j, h_j = 1|N$ et $Bm|set_j, h_j = 1|N$ se résolvent en temps polynomial.*

5.4 Un FPTAS pour $Bm|any_j|H_{\max}$

Dans les sections précédentes, j'ai montré qu'il existe un algorithme pseudo-polynomial pour résoudre le problème $Bm|any_j|H_{\max}$. Sur cette base, nous allons bâtir un FPTAS pour ce problème. La technique utilisée, « par redimensionnement » (*scaling technique* ou *fixed partitioning technique* dans la littérature anglophone), est classique ; elle a été appliquée avec succès sur différents problèmes voisins des problèmes de rangements d'objets multiboîtes, comme les problèmes Bi-Partition ou Sac-à-dos. Pour plus de précisions sur la technique elle-même et sur ses applications, le lecteur est renvoyé à l'ouvrage de Ausiello *et al.* [10].

[1]Ce résultat a déjà été prouvé, plus généralement, pour $B|size_j, h_j = 1|H_{\max}$ et $B|size_j, h_j = 1|N$ (voir chapitre 3, propositions 3.5 et 3.7).

La méthode est générale pour un problème numérique (ici, les nombres sont les hauteurs des objets) pseudo-polynomial, c'est-à-dire qui se résout en temps polynomial en ses nombres. L'idée est que, pour le rendre polynomial, il « suffit » de réduire le nombre de valeurs distinctes à une quantité polynomiale. Cela s'effectue, pour nous, en prenant comme hauteurs, à la place des h_j, les valeurs $\lfloor h_j/\delta \rfloor$; pour δ judicieusement choisi, le nombre de hauteurs d'objets distinctes est grandement diminué (ce qui réduit d'autant le temps de calcul), sans perdre le contrôle sur la qualité de la solution. Voyons maintenant comment cela s'applique sur notre problème.

Le théorème 5.1 affirme l'existence d'un algorithme pseudo-polynomial, que j'appellerai Alg, pour résoudre $Bm|any_j|H_{\max}$ en temps $\mathcal{O}((2^m-1)nA^m)$, avec $A = \sum_{j=1}^{n} \max_{k=1..m} h_j(k)$.

Redimensionnons les objets par un facteur δ : $h_j \to h'_j = \lfloor h_j/\delta \rfloor$. À partir d'une instance initiale I, on obtient ainsi une instance I' pour laquelle le temps de résolution est réduit par δ, par rapport au temps nécessaire pour I. En résolvant I' pour obtenir une solution S', optimale pour I', on déduit une solution S pour I en redonnant aux objets de S' leurs hauteurs initiales. L'erreur contingente à cette dernière transformation est en fait bornée par δ ; aussi, pour un bon choix de δ, la méthode est un FPTAS, comme cela est exprimé ci-dessous et mis en œuvre par l'algorithme 5.3.

Algorithme 5.3. Un FPTAS pour $Bm|any_j|H_{\max}$

1. `calculer` $\delta = \varepsilon \frac{h_{\max}}{n}$
2. `transformer` I `pour obtenir` I' : $h'_j = \left\lfloor \frac{h_j}{\delta} \right\rfloor$
3. `résoudre` I' : $S' = Alg(I')$
4. `renvoyer` S (S' `avec les hauteurs d'origines`)

Théorème 5.5. *L'algorithme 5.3 est un FPTAS pour* $Bm|any_j|H_{\max}$. *Son temps d'exécution est* $\mathcal{O}((2^m-1)\frac{n^{2m+1}}{\varepsilon^m})$.

PREUVE — Pour cette preuve, je vais me servir de plusieurs instances et solutions intermédiaires, dont certaines ont déjà été introduites dans la discussion précédente. À chaque fois, seules les hauteurs des objets diffèrent. Par ordre d'apparition, nous avons...

I est l'instance à résoudre, avec les hauteurs h_j. I' est l'instance redimensionnée, avec les hauteurs $\lfloor h_j/\delta \rfloor$; S' est une solution optimale pour I'. S'' est S' avec les hauteurs multipliées par δ (en d'autres termes, S'' est une solution optimale pour I'', instance avec les hauteurs $\delta \lfloor h_j/\delta \rfloor$). S est la solution finale, en rendant à S'' les hauteurs initiales.

Remarquons tout d'abord que $\forall \delta > 0 : \delta \lfloor h_j/\delta \rfloor \leq h_j$. De ce fait, $H_{\max}^{S''} \leq H_{\max}^{*}$ (où $H_{\max}^{*}$ est la valeur optimale pour I).

Lorsque les hauteurs de S'' sont remises à leurs valeurs initiales, pour obtenir S, l'erreur additive introduite est bornée. En fait, l'erreur sur un objet est au plus de δ ($\delta \lfloor h_j/\delta \rfloor + \delta \geq h_j$) ; comme il y a au plus n objets par boîte, l'erreur additive totale est au plus de $n\delta$. Il en découle : $H^S_{\max} \leq H^{S''}_{\max} + n\delta$, d'où :

$$H^S_{\max} \leq H^*_{\max} + n\delta.$$

Il ne reste qu'à constater que $H^*_{\max} \geq h_{\max}$ pour conclure :

$$\frac{H^S_{\max}}{H^*_{\max}} \leq 1 + \frac{n\delta}{h_{\max}} = 1 + \varepsilon$$

et donc que l'algorithme 5.3 est bien une $(1+\varepsilon)$-approximation.

Venons en maintenant au temps de calcul. L'étape (1) est en temps constant ; les étapes (2) et (4) sont en temps linéaire. Le temps de calcul global est donc dominé par l'étape (3). Le théorème 5.1 nous assure qu'elle s'effectue en temps :

$$\begin{aligned}
\mathcal{O}\left((2^m-1)n(\sum_{j=1}^{n} h'_j)^m\right) &\leq \mathcal{O}\left((2^m-1)n(n\frac{h_{\max}}{\delta})^m\right) \\
&\leq \mathcal{O}\left((2^m-1)n(\frac{n^2}{\varepsilon})^m\right) \\
&\leq \mathcal{O}\left((2^m-1)\frac{n^{2m+1}}{\varepsilon^m}\right)
\end{aligned}$$

ce qui est polynomial à la fois en la taille de l'instance, si m est fixé, et en $1/\varepsilon$. L'algorithme 5.3 est donc bien un FPTAS. ⊗

Corollaire 5.5.1. *L'algorithme 5.3 est un FPTAS pour $Bm|size_j|H_{\max}$ et pour $Bm|set_j|H_{\max}$.*

5.5 Pour aller plus loin

La résolution en temps pseudo-polynomiale de problèmes multiboîtes est réglée par ce qui précède, au moins pour les modèles généraux. En effet, les cas $\mathcal{NP}$-complets (sans l'être fortement) sont pris en compte par les traitements proposés. Bien entendu, certains cas particuliers sont encore à envisager.

Pour poursuivre dans l'esprit de ce chapitre, il serait plus approprié de continuer l'étude de l'approximabilité des problèmes $Bm|any_j|\cdot$, notamment :

Question 5.1 : Peut-on obtenir un FPTAS pour $Bm|any_j|W$?

De substantielles améliorations, principalement du FPTAS pour $Bm|any_j|H_{\max}$, sont également envisageables. Pour cela, le lecteur est renvoyé à la discussion du chapitre 9, section 9.2.

Le cas $B|allow_j|H_{\max}$

Il a toujours pensé qu'une idée de plus
n'était pas une addition.
— *Henri Michaux*

Ce chapitre est consacré à un cas particulier de rangement d'objets dans des boîtes dédiées : le problème $B|allow_j|H_{\max}$. Dans ce modèle, un objet O_j a une largeur fixe $size_j$, mais il n'est compatible qu'avec certaines boîtes. Ces compatibilités sont indiquées par le vecteur caractéristique $allow_j$: $allow_{j,i}$ vaut 1 si O_j peut être rangé dans B_i et 0 sinon. Ce problème généralise les classiques couplages dans les graphes bipartis (voir ci-après). Deux exemples d'applications sont donnés au chapitre 1, sections 1.3.4 et 1.3.5.

Dans la suite de ce chapitre, je formalise le problème et je montre que, dans le cas de hauteurs unitaires, le problème est polynomial. Une deuxième section présente ensuite une approche plus spécifique du problème, basée sur les (g, f)-facteurs.

À noter que, pour la durée de ce chapitre, la notation Δ n'indique pas la plus grande différence de hauteur entre deux boîtes, mais l'opérateur de différence symétrique ; le contexte devrait effacer toute ambiguïté, une telle utilisation de Δ étant fréquente.

6.1 Formalisation de $B|allow_j|H_{\max}$

Afin de fixer clairement les idées, commençons par donner une représentation sous forme de modèle linéaire en 0-1 du problème $B|allow_j|H_{\max}$:

$$(B|allow_j|H_{\max}) \quad \begin{cases} \min & H_{\max} & & \\ s.c. & \bullet \sum_{j=1}^{n} h_j\, x_{ij} \leq H_{\max} & i = 1, 2, \ldots, m \\ & \bullet \sum_{i=1}^{m} x_{ij} = size_j & j = 1, 2, \ldots, n \\ & \bullet\ x_{ij} \leq allow_{ji} & i = 1, 2, \ldots, m, j = 1, 2, \ldots, n \\ & \bullet\ x_{ij} \in \{0, 1\} & i = 1, 2, \ldots, m, j = 1, 2, \ldots, n \end{cases}$$

Ce chapitre a grandement bénéficié des conseils et des suggestions de Maurice Queyranne, ainsi que de remarques d'András Sebő. Il a été présenté à [88].

Lorsque l'on cherche juste à déterminer si, pour une hauteur $H_{\max}$ donnée, il existe une solution, ce modèle s'apparente à un problème de transport avec capacités [107]. Chaque objet O_j a une offre de $size_j \times h_j$, et chaque boîte une demande de $H_{\max}$. Le transport d'un objet O_j vers une boîte B_i est limité à h_j si boîte et objet sont compatibles, et est interdit sinon. La différence, fondamentale, avec le classique problème de transport avec capacités, est que le long d'une route autorisée, il faut transporter soit rien du tout, soit h_j : les valeurs intermédiaires ne sont pas permises (cela reviendrait à casser un objet).

Si les hauteurs des objets sont unitaires, l'algorithme de transport avec capacités peut être appliqué. Cependant, d'un point de vue du transport, le problème est fortement dégénéré, ce qui nuit à une bonne résolution. Cela permettrait toutefois une résolution polynomiale et optimale du problème.

6.2 $B|allow_j, h_j = 1|H_{\max}$ et flots maximaux

Cette section présente une méthode polynomiale pour résoudre $B|allow_j, h_j = 1|H_{\max}$; celle-ci est basée sur les flots dans un graphe biparti.

L'ensemble des vecteurs $allow_j$ forme une matrice en 0-1 qui peut être interprétée comme la matrice d'adjacence d'un graphe biparti, caractéristique de l'instance considérée : le graphe associé à l'instance I.

Définition 6.1. (graphe associé) *Soit I une instance de $B|allow_j, h_j = 1|H_{\max}$: un ensemble* **O** *d'objets de largeurs constantes, un ensemble* **B** *de boîtes et des compatibilités $allow_{j,i}$. Soit également une borne H.*

Le graphe associé à I pour la borne H, *$G^I(H)$ ou plus simplement $G(H)$, est défini par :*

- *$G(H)$ est biparti et orienté ;*
- *les deux ensembles de sommets sont indexés respectivement par* **O** *et* **B**, *auxquels une source S et un puit P sont ajoutés ;*
- *un arc existe de O_j vers B_i si, et seulement si, O_j peut être rangé dans B_i ($allow_{j,i} = 1$) ; si cet arc existe, il est de capacité 1 ;*
- *il y a un arc de capacité $size_j$ de S vers chaque O_j ;*
- *il y a un arc de capacité H de chaque B_i vers P.*

La taille de ce graphe est polynomiale en la taille de l'instance. De plus, le problème de décision $B|allow_j, h_j = 1|H_{\max}$, pour une instance I et une borne H, a une réponse « oui » si, et seulement si, le flot maximal sur $G^I(H)$ est de $\sum_{j=1}^{n} size_j$. Par une recherche dichotomique, on peut ainsi chercher le H optimal, c'est-à-dire le plus petit pour lequel le problème de décision a une réponse affirmative. Cette recherche dichotomique est polynomiale : $H^*_{\max}$ est borné par n (le nombre d'objets). Il découle de cette discussion la propriété suivante[1] :

[1]Nous savions déjà que $Bm|allow_j, h_j = 1|H_{\max}$ pouvait être résolu en temps polynomial (théorème 5.3).

Théorème 6.1. *$B|allow_j, h_j = 1|H_{\max}$ est soluble à l'optimalité en temps polynomial $\mathcal{O}(\ln(n)F(n+m, nm, \max(n,m)))$, où $F(x,y,U)$ est le temps de calcul d'un flot maximal sur un graphe (biparti) à x sommets, y arêtes et une capacité maximale U.*

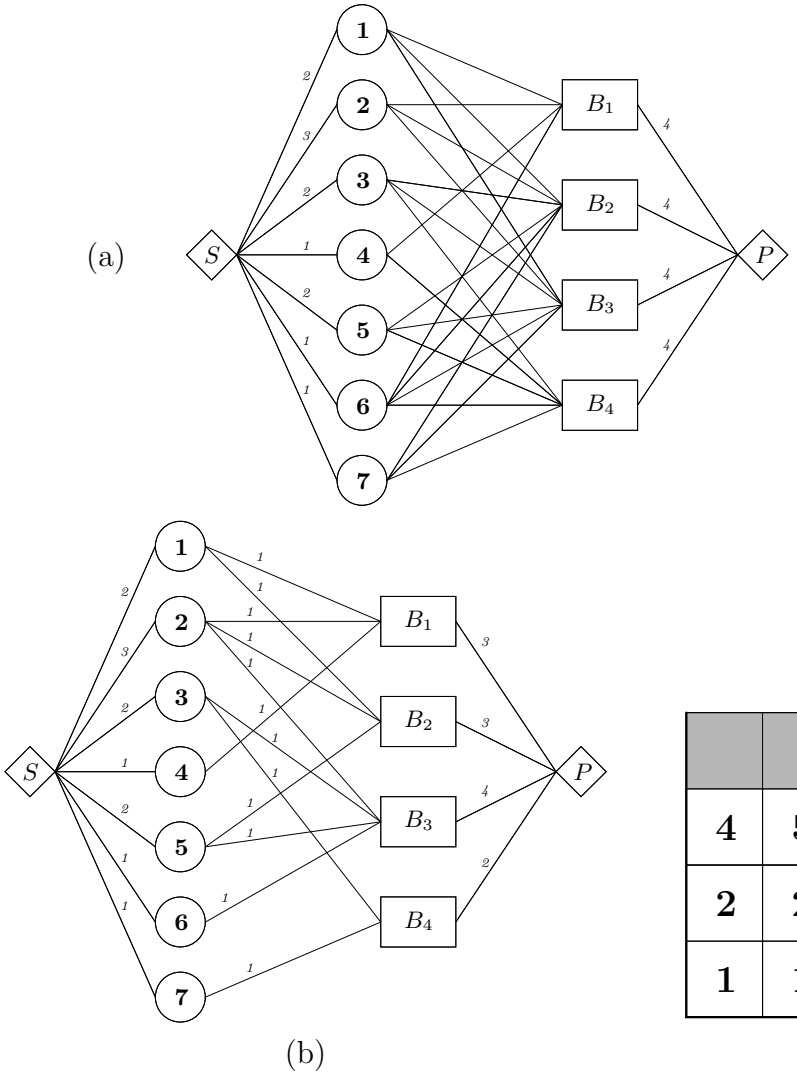


Figure 6.1 – Résolution de $B|allow_j, h_j = 1|H_{\max}$ par un flot maximal.
(a) l'instance représentée par son graphe associé (arcs de capacité 1 ou indiquée) ;
(b) une solution de flot maximal pour $H = 4$ (valeur du flot sur les arcs) ;
(c) le rangement correspondant.

La figure 6.1 illustre une telle résolution. De nombreuses méthodes de calcul de flots existent ; pour les principales, la table 6.1 présente les complexités de résolution de $B|allow_j, h_j = 1|H_{\max}$ qui en découlent. Ces algorithmes ont été proposés pour des graphes quelconques ; aussi, pour le cas de graphes bipartis, de substantielles améliorations sont prévisibles, et à envisager pour résoudre en pratique les problèmes $B|allow_j, h_j = 1|H_{\max}$.

Table 6.1 – Complexité de $B|allow_j, h_j = 1|H_{\max}$ selon l'algorithme de flot. n est le nombre d'objets, m est le nombre de boîtes, et l'on suppose $n \geq m$. Pour plus de détails sur ces algorithmes, voir [2].

Algorithme de flot	**Résolution de** $B\|allow_j, h_j = 1\|H_{\max}$
Labeling algorithm	$\mathcal{O}(\ln(n)(n+m)n^2m)$
Capacity scaling algorithm	$\mathcal{O}(\ln^2(n)(n+m)nm)$
Successive shortest path algorithm	$\mathcal{O}(\ln(n)(n+m)^2nm)$
Generic preflow-push algorithm	$\mathcal{O}(\ln(n)(n+m)^2nm)$
FIFO preflow-push algorithm	$\mathcal{O}(\ln(n)(n+m)^3)$
Highest-label preflow-push algorithm	$\mathcal{O}(\ln(n)(n+m)^2\sqrt{nm})$
Excess scaling algorithm	$\mathcal{O}(\ln(n)((n+m)nm + (n+m)^2\ln n))$

De plus, on peut dériver de cette résolution par flot une caractérisation de type min-max, adaptée du célèbre « flot-max/coupe-min » de Ford et Fulkerson [58, 2]. Ainsi, pour une hauteur de boîte H donnée, il existe une solution réalisable (H est alors dit réalisable) si, et seulement si, le flot maximal sur $G(H)$ est au moins de $\sum_{j=1}^{n} size_j$, donc si, et seulement si, la coupe minimale de $G(H)$ est au moins de $\sum_{j=1}^{n} size_j$, c'est-à-dire si, et seulement si, $\forall(\mathbf{O}', \mathbf{B}') \subset (\mathbf{O}, \mathbf{B})$:

$$\sum_{O_j \in \mathbf{O}\setminus\mathbf{O}'} size_j + |E(\mathbf{O}', \mathbf{B}\setminus\mathbf{B}')| + H|\mathbf{B}'| \geq \sum_{j=1}^{n} size_j,$$

c'est-à-dire

$$H \geq \left\lceil \frac{\sum_{j\in\mathbf{O}'} size_j - |E(\mathbf{O}', \mathbf{B}\setminus\mathbf{B}')|}{|\mathbf{B}'|} \right\rceil.$$

Ceci mène à la caractérisation suivante :

Théorème 6.2.

$$H^*_{\max} = \min_{H \text{ réalisable}} H = \max_{(\mathbf{O}', \mathbf{B}') \subset (\mathbf{O}, \mathbf{B})} \left\lceil \frac{\sum_{j\in\mathbf{O}'} size_j - |E(\mathbf{O}', \mathbf{B}\setminus\mathbf{B}')|}{|\mathbf{B}'|} \right\rceil.$$

Cette caractérisation permet, notamment d'obtenir une borne inférieure en prenant $\mathbf{O}' = \mathbf{O}$ et $\mathbf{B}' = \mathbf{B}$:

Corollaire 6.2.1.

$$H^*_{\max} \geq \left\lceil \frac{\sum_j size_j}{m} \right\rceil.$$

6.3 $B|allow_j, h_j = 1|H_{\max}$ et (g, f)-facteurs

L'approche de la section précédente permet une résolution polynomiale de $B|allow_j, h_j = 1|H_{\max}$. Toutefois, elle ne tire pas parti des particularités de ce

problème, par rapport aux problèmes de flots en général. Cette section met en avant ces différences et ce que l'on peut en déduire ; le but est ainsi de déterminer une caractérisation efficace et algorithmique des solutions optimales de $B|allow_j, h_j = 1|H_{\max}$. Pour cela, je m'appuie sur des facteurs de graphes particuliers.

6.3.1 Un (g, f)-facteur particulier

Un facteur d'un graphe est un sous-graphe validant certaines contraintes de degrés : ainsi, un (g, f)-facteur est un sous-graphe dont le degré est, pour chaque sommet v, dans l'intervalle $[g(v), f(v)]$. Le plus connu des facteurs est le 1-facteur, ou couplage parfait : il consiste à trouver un ensemble d'arêtes touchant chaque sommet exactement une fois.

Les premiers travaux sur les facteurs ont été réalisés par Tutte [121] et Ore [111]. Lovász [101] a caractérisé les f-facteurs (cas où $f = g$). Pour plus de détails sur le problème général des facteurs, le lecteur est renvoyé au livre de Lovász et Plummer [102].

Pour le cas particulier des graphes bipartis, qui nous intéresse ici, Heinrich *et al.* [66] ont proposé un critère simplifié pour l'existence d'un $(g < f)$-facteur (ce critère a été étendu par Anstee [5] ; voir aussi [8]). Leurs résultats sont issus de l'approche algorithmique classique des problèmes de couplage : celle de l'utilisation de chaînes alternées, et est basée sur les travaux de Lovász. J'adopte une approche semblable.

Considérons une instance de $B|allow_j, h_j = 1|H_{\max}$ et son graphe associé G, indépendamment de toute borne H (voir définition 6.1, section précédente). Il existe une bijection triviale entre les solutions réalisables pour I et les (g, f)-facteurs de G qui satisfont : $\deg(O_j) = size_j$ pour tout sommet $O_j \in \mathbf{O}$. La valeur d'un tel facteur (*i.e.* la valeur de la solution correspondante) est le degré maximum pour un sommet $B_i \in \mathbf{B}$. Nous voulons donc résoudre le problème FMFMB (Facteur Mi-Fixé Mi-Borné) :

Problème FMFMB

Instance. Un graphe biparti $G = (A \cup B, E)$, des tailles $a_j \in \mathbb{N}$ pour chaque $j \in A$ et une borne $\beta \in \mathbb{N}$.

Question. Existe-t-il un sous-graphe F de G tel que : $\forall j \in A : \deg_F(j) = a_j$ et $\forall i \in B : \deg_F(i) \leq \beta$?

Pour caractériser l'existence d'un (g, f)-facteur, nous disposons de plusieurs résultats, dont le plus approprié à notre cas particulier est celui de Heinrich *et al.* [66], repris par Anstee [5] : un graphe biparti G a un (g, f)-facteur si, et seulement si, pour chaque sous-ensemble de sommets S :

$$\sum_{z \notin S} \max\{0, g(z) - \deg_{G \setminus S}(z)\} \leq f(S). \tag{6.1}$$

La preuve constructive de ce résultat, alliée à la recherche dichotomique, permet d'obtenir un algorithme polynomial pour résoudre $B|allow_j, h_j = 1|H_{\max}$. À noter d'ailleurs que, le graphe étant biparti, des implémentations particulièrement performantes sont possibles (voir p.ex. [4]).

6.3.2 Résolution par (g, f)-facteur

Le but de ce qui suit est de caractériser les solutions optimales de FMFMB, dans un sens décrit ci-après. Cette caractérisation est basée sur des chemins alternés particuliers. Avant tout, commençons par quelques définitions et propriétés.

Définition 6.2. (facteur réalisable) *Soient un graphe biparti $G = (A \cup B, E)$ et des tailles entières $a_j, j \in A$. Un facteur F de G est dit* réalisable *(pour G et les a_j) s'il existe un β tel que F est une solution de* FMFMB *pour le graphe G, les tailles a_j et la borne β. La* valeur *d'un F réalisable est le plus petit β pour lequel il est solution de* FMFMB*,* i.e. *: $\beta = \max_{i \in B} \deg_F(i)$.*

Propriété 6.1. *Un facteur réalisable F vérifie : $|F| = \sum_{j \in A} a_j = \sum_{i \in B} \deg_F(i)$, où $|F|$ est le nombre d'arêtes de F.*

Définition 6.3. (facteur optimal[2]) *Un facteur F, réalisable et de valeur β, est* optimal *s'il n'existe pas de solution réalisable de* FMFMB *pour $\beta - 1$. La valeur optimale est notée β^*.*

Propriété 6.2. *La valeur optimale β^* de* FMFMB *vérifie :*

$$\beta^* \geq \left\lceil \frac{\sum_{j \in A} a_j}{|B|} \right\rceil .$$

PREUVE — Cette propriété n'est en fait qu'une réécriture du corollaire 6.2.1. Elle découle aussi de l'équation 6.1, en prenant $S = B$: $\sum_{j \in A} \max(0, a_j) \leq \sum_{i \in B} \beta = |B|\beta$. L'entièreté de β permet de conclure. ⊗

Définition 6.4. (sommet (in)saturé) *Un sommet $i \in B$ est dit* insaturé *par un facteur F si $\deg_F(i) < \deg_G(i)$. Autrement, il est dit* saturé*.*

Propriété 6.3. *Si F est un facteur de valeur $\beta > \beta^*$ alors il existe un sommet $i \in B$ à la fois insaturé et tel que $\deg_F(i) < \beta - 1$.*

[2]Les lecteurs de Lovász et Plummer [102] devront faire attention : ces derniers définissent aussi une notion de « facteur optimal » qui, bien que proche, n'est pas équivalente.

PREUVE — Soit F^* un facteur optimal ; puisque $\beta > \beta^*$, il existe un sommet $z \in B$ tel que $\deg_F(z) = \beta > \deg_{F^*}(z)$. La propriété 6.1 implique :

$$\sum_{i \in B} \deg_F(i) = \sum_{i \in B} \deg_{F^*}(i).$$

Donc

$$\sum_{\substack{i \in B \\ i \neq z}} \deg_F(i) < \sum_{\substack{i \in B \\ i \neq z}} \deg_{F^*}(i).$$

En conséquence, il y a un $i \in B$ tel que $\deg_F(i) < \deg_{F^*}(i)$. Puisque F^* est réalisable : $\deg_{F^*}(i) \leq \deg_G(i)$; puisque sa valeur est β^* : $\deg_{F^*}(i) \leq \beta^* \leq \beta - 1$. De là découle $\deg_F(i) < \deg_G(i)$ (i est insaturé) et $\deg_F(i) < \beta - 1$. ⊗

La propriété précédente est importante : elle indique l'existence de boîtes trop faiblement remplies, candidates pour accueillir des objets. Elle a ainsi pour conséquence directe la proposition suivante :

Proposition 6.1. *Un facteur F de valeur β est optimal si $\forall i \in B$: $\deg_F(i) \in \{\beta, \beta - 1, \deg_G(i)\}$.*

Après ces préliminaires, nous voici prêts à rentrer dans le vif du sujet et l'on peut maintenant introduire une notion de chemin alterné, puis présenter le théorème caractéristique des solutions minimales pour FMFMB.

Définition 6.5. (chemin alterné) *Un chemin d'un sommet $u \in B$ vers un sommet $v \in B$ est un* chemin alterné *(par rapport à un facteur F) si ses arêtes appartiennent alternativement à F et à $G \backslash F$.*

Définition 6.6. (chemin réducteur, réduction) *Un chemin alterné de u vers v est* réducteur *si $\deg_F(u) = \beta$ et $\deg_F(v) < \beta - 1$ et v est insaturé. De plus, un facteur F' est une* réduction *de F (notée $F \succ F'$) s'il existe un chemin réducteur P tel que $F' = \{F \backslash P\} \cup \{P \backslash F\} = F \Delta P$.*

On peut remarquer que, le graphe étant biparti, un chemin alterné a un nombre pair d'arêtes puisqu'il a ses deux extrémités dans B.

Définition 6.7. (ensemble maximal et facteur minimal) *L'ensemble des sommets de B de degré maximal, $S_F = \{i \in B, \deg_F(i) = \beta\}$, est l'*ensemble maximal *d'un facteur F de valeur β. De plus, un facteur optimal F est dit* minimal *si son ensemble maximal est de cardinalité minimale parmi les facteurs optimaux.*

Le concept de chemin réducteur joue pour FMFMB le même rôle que les chaînes augmentantes pour les couplages ; ainsi une réduction F' est toujours strictement meilleure (au sens de la définition précédente) que la solution F dont elle est la réduction :

Lemme 6.1. *Soit F et F' deux facteurs pour la même instance de* FMFMB.

$$\begin{array}{lll} (a) & F \succ F', |S_F| > 1 & \Longrightarrow \quad S_{F'} \subset S_F \text{ et } S_{F'} \neq S_F \\ (b) & F \succ F', |S_F| = 1 & \Longrightarrow \quad \beta_{F'} < \beta_F \end{array}$$

PREUVE — Si F' est une réduction de F alors il existe un chemin réducteur P tel que $F' = F\Delta P$. Soit u l'extrémité de P qui est dans F, et v l'autre extrémité. Puisque $\deg_F(u) = \beta_F$ et que $\deg_F(v) < \beta_F - 1$, l'ensemble maximal S_F perd un élément. Deux cas peuvent se produire. Si $|S_F| > 1$, alors $S_F \backslash \{u\}$ est l'ensemble maximal de F' : $S_{F'} \subset S_F$ et $S_{F'} \neq S_F$. Si $|S_F| = 1$ alors il n'y a plus de sommet de degré β_F après réduction : $\beta_{F'} < \beta_F$. ⊗

Venons-en maintenant à la caractérisation des solutions minimales.

Théorème 6.3. *Un facteur F est minimal si, et seulement si, il n'existe pas de chemin réducteur vis-à-vis de F.*

PREUVE — L'implication directe est une conséquence immédiate du lemme précédent. En effet, s'il existe un chemin réducteur P alors $F' = F\Delta P$ soit est strictement meilleur (cas (b)), soit a un plus petit ensemble maximal (cas (a)). Dans les deux cas, F n'est pas minimal.

Considérons maintenant un facteur F, non minimal, et construisons un chemin réducteur pour F, par l'intermédiaire d'un facteur minimal F^*.

Commençons par un sommet $i_0 \in B$ tel que $\deg_F(i_0) = \beta_F > \deg_{F^*}(i_0)$ (il existe, sinon F serait minimal). D'un tel sommet part une arête $(i_0, i_1) \in F \backslash F^*$.

Le sommet i_1 appartient à A. Puisque $\deg_F(i_1) = \deg_{F^*}(i_1)$ (ceci est vrai $\forall i \in A$), il existe une arête $(i_1, i_2) \in F^* \backslash F$. On marque (i_0, i_1) ainsi que (i_1, i_2) (une arête marquée ne sera jamais réutilisée).

Le sommet i_2 est dans B et il n'est pas saturé par F (il a été atteint par une arête de $F^* \backslash F$). Aussi, si $\deg_F(i_2) < \beta - 1$, alors un chemin réducteur a été construit et c'est fini.

Si $\deg_F(i_2) \geq \beta - 1$ et qu'il existe une arête (non marquée) $(i_2, i_3) \in F \backslash F^*$, on l'utilise pour continuer la recherche, pour i_3 comme cela a été fait pour i_1.

Si une telle arête n'existe pas, cela signifie que $\deg_{F^*}(i_2) > \deg_F(i_2) \geq \beta - 1$. Dans ce cas-là, F est optimal. Puisqu'il n'est pas minimal, il existe un sommet i_4, différent de i_0 tel que $\deg_F(i_4) = \beta_F > \deg_{F^*}(i_4)$. Une nouvelle recherche commence alors à partir de i_4.

La recherche ci-dessus n'explore jamais deux fois la même arête : elle s'arrête donc. De plus, elle ne s'arrête que si elle trouve un chemin réducteur : un tel chemin existe donc. ⊗

En fait, pour un facteur quelconque F et un facteur minimal F^*, le graphe $F\Delta F^*$ peut être décomposé en cycles alternés et en chemins réducteurs (plus d'éventuels chemins alternés non réducteurs pour un F optimal mais non minimal). De plus, si un facteur optimal est suffisant, la recherche peut être arrêtée au premier échec (il confirme l'optimalité) au lieu de prendre un nouveau point de départ. Ainsi :

Corollaire 6.3.1. *Soit F^* un facteur minimal ; un facteur F n'est pas optimal si, et seulement si, il existe un chemin réducteur à partir de n'importe quel sommet $i \in B$ tel que* $\deg_F(i) = \beta_F > \deg_{F^*}(i)$.

Corollaire 6.3.2. *S'il n'existe pas de chemin réducteur pour un facteur F, alors celui-ci est optimal.*

Il faut se méfier de ce corollaire : sa réciproque est fausse, comme le montre la figure 6.2.

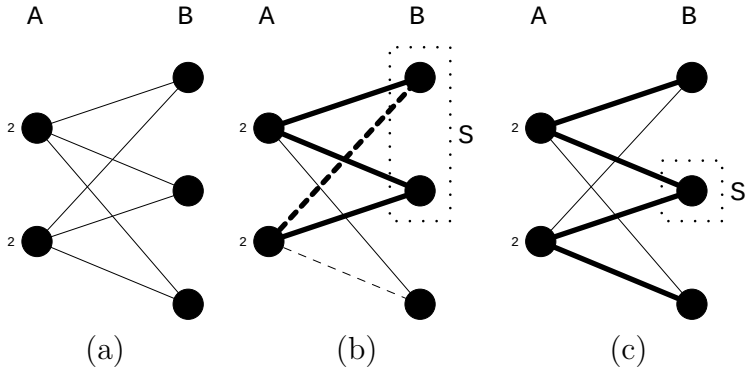


Figure 6.2 – Un exemple de facteur optimal admettant un chemin réducteur. (a) l'instance ; (b) un facteur optimal (lignes épaisses) avec un chemin réducteur (pointillés) ; (c) un facteur minimal.

À partir de la preuve du théorème 6.3, un algorithme de recherche d'un chemin réducteur se déduit :

Algorithme 6.1. Recherche d'un chemin réducteur

- **entrées :** graphe G, facteur F -
1. pour $u \in B$ tel que $\deg_F(u) = \beta_F$, faire :
   marquer u « à explorer ».
2. tant qu'il existe un u « à explorer », faire :
   a. si $u \in B$ alors, pour $v \in succ_F(u)$, faire :
      si v non marqué, le marquer « à explorer ».
   b. si $u \in A$ alors pour $v \in succ_{G \setminus F}(u)$, faire
      si $\deg_F(v) < \min(\beta - 1, \deg_G(v))$
      alors STOP (v est la fin du chemin réducteur) ;
      sinon marquer v « à explorer ».
   c. marquer u « exploré ».
3. il n'y a pas de chemin réducteur.

Corollaire 6.3.3. *L'algorithme 6.1 calcule un chemin réducteur — ou détermine qu'il n'en existe pas — en temps $\mathcal{O}(|E|)$, où $|E|$ est le nombre d'arêtes de G.*

PREUVE — Pour la clarté de la démonstration, supposons que les sommets « à explorer » (étape 2) sont explorés selon le principe FIFO (premier rentré, premier sorti). On a alors l'invariant suivant : au début de la $k^{ème}$ itération de la boucle (étape 2), un sommet u a été marqué si, et seulement si, il existe un chemin alterné (vis-à-vis de F) d'un sommet v, de degré dans F $\deg_F(v) = \beta_F$, à u, de longueur au plus $k-1$. Cet invariant permet de conclure immédiatement quant à la validité de l'algorithme.

Une arête n'est considérée qu'une fois : l'algorithme est donc en $\mathcal{O}(|E|)$. ⊗

Dans la démonstration précédente, l'hypothèse de gestion FIFO des sommets marqués permet en fait de trouver un plus court chemin réducteur. Sans cette hypothèse, l'algorithme reste correct, mais ne trouve qu'un chemin réducteur quelconque.

À partir de cette recherche d'un chemin réducteur, se déduit facilement un algorithme polynomial pour résoudre optimalement (et même minimalement) FMFMB :

Corollaire 6.3.4. *Un facteur minimal peut être construit en temps polynomial $\mathcal{O}(|E||B|(|A| - \left\lceil \frac{\sum_{j\in A} a_j}{|B|} \right\rceil))$, pour $G = (A \cup B, E)$.*

PREUVE — Selon le corollaire 6.3.3, un chemin réducteur P se calcule en temps $\mathcal{O}(|E|)$. Lorsqu'un tel chemin est trouvé, réduire le facteur (*i.e.* calculer $F' = F\Delta P$) se fait également en temps $\mathcal{O}(|E|)$. Une telle réduction soit diminue la cardinalité de l'ensemble maximal de 1, soit réduit la valeur du facteur d'au moins 1 (lemme 6.1).

Il ne peut pas y avoir plus de $|B|$ réductions successives de l'ensemble maximal, et β vérifie : $|A| \geq \beta \geq \left\lceil \sum_{j\in A} a_j/|B| \right\rceil$ (propriété 6.2). Il y a donc au plus $|B|(|A| - \left\lceil \sum_{j\in A} a_j/|B| \right\rceil + 1)$ réductions avant de déduire qu'il n'existe plus de chemin réducteur, et donc que la solution est minimale.

Globalement, cela donne un algorithme en $\mathcal{O}(|E||B|(|A| - \left\lceil \sum_{j\in A} a_j/|B| \right\rceil))$. ⊗

Avant de conclure cette section, signalons une autre propriété des facteurs minimaux :

Proposition 6.2. *Un facteur réalisable est minimal si*

$$\sum_{j\in A} \max\{0, a_j - \deg_{G\setminus S_F}(j)\} = |S_F|\beta_F.$$

De manière équivalente, s'il existe un chemin réducteur, alors :

$$\sum_{j\in A} \max\{0, a_j - \deg_{G\setminus S_F}(j)\} < |S_F|\beta_F.$$

PREUVE — L'équation 6.1 appliquée pour l'ensemble maximal S_F prouve que, si F est réalisable, alors

$$\sum_{j\in A}\max\{0, a_j - \deg_{G\setminus S_F}(j)\} \leq |S_F|\beta_F,$$

ce qui implique l'équivalence des deux affirmations.

Supposons maintenant qu'il existe un chemin réducteur P. Deux cas sont à envisager, selon la cardinalité de l'ensemble maximal.

cas 1 : $|S_F| > 1$.

Soit F' la réduction de F par P. D'après le lemme 6.1, $S_{F'} \subset |S_F|$ et $S_{F'} \neq |S_F|$ et $\beta_F = \beta_{F'}$. Alors :

$$\sum_{j\in A}\max\{0, a_j - \deg_{G\setminus S_F}(j)\} \leq \sum_{j\in A}\max\{0, a_j - \deg_{G\setminus S_{F'}}(j)\}.$$

L'équation 6.1 utilisée avec l'ensemble $S_{F'}$ donne

$$\sum_{j\in A}\max\{0, a_j - \deg_{G\setminus S_{F'}}(j)\} \leq |S_{F'}|\beta_{F'},$$

puisque $S_{F'} \subset |S_F|$ et $S_{F'} \neq |S_F|$, on a $|S_F| < |S_{F'}|$. Tout ceci remis ensemble implique que

$$\sum_{j\in A}\max\{0, a_j - \deg_{G\setminus S_F}(j)\} < |S_F|\beta_F.$$

cas 2 : $|S_F| = 1$.

Soit u le sommet de S_F extrémité de P. Décomposons alors les sommets de A en :

$$\begin{aligned} U &= \{z \in A | (z,u) \in E\setminus F\} \\ T &= \{z \in A | (z,u) \in F\} \end{aligned}$$

(on peut remarquer que $|T| = \beta_F$).

P est un chemin alterné à partir de u : soit v le voisin de u dans P. Par définition, $v \in T \cap P$. De plus :

$$\begin{aligned} &\forall z \notin U \cup T : \ \deg_G(z) = \deg_{G\setminus S_F}(z) \\ &\forall z \in U \cup T : \ \deg_G(z) = \deg_{G\setminus S_F}(z) + 1, \end{aligned}$$

puisque $|S_F| = 1$. Il s'ensuit que

$$\begin{aligned} &\sum_{z\in A}\max(0; a_z - \deg_{G\setminus S_F}(z)) \\ &= \sum_{z\notin U\cup T}\max(0; a_z - \deg_{G\setminus S_F}(z)) + \sum_{z\in U\cup T}\max(0; a_z - \deg_{G\setminus S_F}(z)) \\ &= \sum_{z\notin U\cup T}\max(0; \underbrace{a_z - \deg_G(z)}_{\leq 0}) + \sum_{z\in U\cup T}\max(0; a_z - \deg_G(z) + 1). \end{aligned}$$

De plus : $\forall z \in U : \deg_G(z) > a_z$, puisque z n'est pas saturé par F. En outre, v est dans P et donc il n'est pas non plus saturé. Finalement :

$$\begin{aligned}
&\sum_{z\in A} \max(0; a_z - \deg_{G\setminus S_F}(z)) \\
&= \underbrace{\sum_{z\in U\cup\{v\}} \max(0; \underbrace{a_z - \deg_G(z)}_{<0} + 1)}_{=0} + \underbrace{\sum_{z\in T\setminus\{v\}} \max(0; a_z - \deg_G(z) + 1)}_{\leq |T|-1} \\
&< |T| = \beta_F.
\end{aligned}$$

⊗

La réciproque de cette propriété est fausse : un facteur peut être minimal bien que $\sum_{j\in A} \max\{0, a_j - \deg_{G\setminus S_F}(j)\} < |S_F|\beta_F$, comme le montre la figure 6.3.

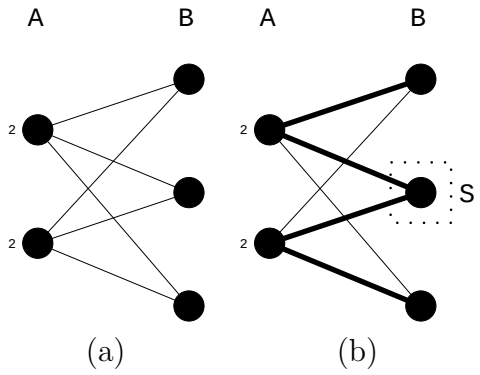


Figure 6.3 – Un contre exemple pour la réciproque de la proposition 6.2.
(a) une instance ; (b) un facteur minimal tel que
$\sum_{j\in A} \max\{0, a_j - \deg_{G\setminus S}(j)\} = 0 < |S|\beta_F = 1 \times 2 = 2$.

6.3.3 Conclusions sur l'approche par (g, f)-facteurs

L'algorithme de construction d'un facteur minimal par le biais de chemins réducteurs, basé sur le théorème 6.3, utilise des principes similaires à ceux de l'algorithme « d'annulation des cycles » (*Cycle-Canceling Algorithm*, dû à Klein [80, 2]). On peut ainsi voir le théorème 6.3 comme un corollaire des résultats de Klein.

Cette section sur les (g, f)-facteurs avait pour but essentiel de dégager les particularités du problème par rapport à la généralité du modèle de flot. Les facteurs minimaux ont ainsi été bien caractérisés. De plus, si l'on revient au problème $B|allow_j, h_j = 1, |H_{\max}$, la complexité de l'algorithme basé sur les chemins réducteurs est de $\mathcal{O}((mn)^2)$: l'approche est ainsi plus performante que l'utilisation d'algorithmes génériques de flots (se rappeler de la table 6.1). De plus, cet algorithme n'est qu'une conséquence immédiate de la caractérisation proposée, réalisée de manière directe. Pour une implémentation véritablement très efficace, le lecteur peut s'inspirer des travaux sur les couplages dans des graphes bipartis, notamment [70] et [4].

6.4 Pour mettre un peu de couleur

Dans les sections précédentes, le problème $B|allow_j|H_{\max}$ a été abordé sous l'angle des flots et des facteurs. Il peut aussi être vu sous un aspect de coloration de graphes.

Par exemple, on peut faire correspondre des couleurs aux hauteurs dans les boîtes. En particulier, une coloration valide des arêtes (*i.e.* pour laquelle deux arêtes adjacentes n'ont pas la même couleur), définit un rangement réalisable ; ainsi le nombre chromatique du graphe associé à une instance est une borne supérieure pour la valeur optimale de cette instance. Une telle borne n'est cependant pas très utile, puisqu'une meilleure borne est donnée simplement par le plus grand degré d'un sommet indexé par une boîte.

Il ne faut cependant pas dénigrer une telle approche. Cette manière de colorer des arêtes fait penser aux méthodes utilisées pour les emplois du temps ; il serait donc intéressant de se pencher plus avant sur les relations que peuvent entretenir ces problèmes avec $B|allow_j|H_{\max}$.

Une autre approche qu'il serait pertinent d'explorer pour $B|allow_j|H_{\max}$ est celle des *extrema* dans les problèmes d'étranglements (voir *Bottleneck Extrema* d'Edmonds et Fulkerson [45]).

Par exemple, on peut résoudre le problème optimalement directement (*i.e.* sans dichotomie) en utilisant une fonction de coût exponentielle (p.ex. mettre un objet dans une boîte coûte x, en mettre un deuxième x^2, un troisième x^3, *etc.*). Il faut toutefois se préoccuper alors de la taille de l'instance et de la complexité des algorithmes de flots convexes utilisés.

Deuxième partie

Résolution des problèmes $B|size_j|H_{\max}$

Maman, avait-il besoin d'être aussi haut ?
— *Roger Waters*

Rappels sur le problème $B|size_j|H_{\max}$

La première partie de ce document a permis de définir le problème $B|size_j|H_{\max}$ et d'en déterminer la complexité et l'approximabilité. Rappelons-en ici les principales caractéristiques et précisons la direction générale de ce qui suit.

Une instance de $B|size_j|H_{\max}$ est composée de n objets et de m boîtes. Chaque objet est défini par sa largeur $size_j$ et sa hauteur h_j : il doit être rangé dans $size_j$ boîtes différentes et occupe alors une hauteur h_j dans chacune d'entre elles. Les boîtes sont toutes identiques, et l'ordre des objets en leur sein est indifférent ; la hauteur d'une boîte est en fait la somme des hauteurs des objets qui y sont rangés. L'objectif est de minimiser $H_{\max}$, la hauteur de la plus grande boîte.

Ce problème est au moins fortement $\mathcal{NP}$-difficile dans le cas général, et $\mathcal{NP}$-difficile si le nombre de boîtes est fixé (théorème 3.1). Dans ce second cas, le problème admet un FPTAS (théorème 5.3), qui n'est malheureusement pas réellement exploitable en pratique.

L'objectif annoncé pour cette seconde partie est de résoudre pratiquement les problèmes $B|size_j|H_{\max}$. Étant donnée leur complexité, une résolution à la fois rapide et optimale est (quasiment) exclue. Entre résoudre optimalement aussi vite que possible et résoudre rapidement aussi bien que possible, je me suis concentré sur la deuxième approche.

Il n'y a ainsi aucune méthode exacte de résolution. On trouvera par contre des bornes inférieures (chapitre 7), des heuristiques avec des garanties de performances (chapitres 8 et 9) et un algorithme génétique (chapitre 10) qui permettent de résoudre très rapidement et très bien la grande majorité des instances (chapitre 11).

Ceci ayant été reprécisé, la deuxième partie de ce travail peut commencer.

Bornes inférieures (et une pincée de bornes supérieures)

Dans les sciences, l'intuition du dilettante peut avoir
une portée parfaitement identique à celle du spécialiste,
et même parfois plus grande.
— *Max Webber*

Débute ici la seconde partie de ce document : la résolution de $B|size_j|H_{\max}$. C'est donc tout naturellement que ce chapitre traite des bornes inférieures (et supérieures) pour ce problème : elles sont nécessaires à l'évaluation de la qualité des solutions produites dans les chapitres suivants.

On trouve dans ce chapitre des bornes de trois sortes. La première est une borne simple mais efficace sur tout type d'instances, basée sur le volume, et qui servira de comparaison théorique pour les algorithmes à performances garanties du chapitre 8 (section 7.1). La seconde est inspirée du « raisonnement énergétique » ; c'est une borne à vocation plus expérimentale, permettant de détecter certaines instances, *a priori* petites, pour lesquelles la borne de volume faut (section 7.2). Le troisième type de bornes vient en complément : il s'agit d'une méthode d'amélioration systématique de bornes, provenant de résultats sur le Bin-Packing (section 7.3). Pour une évaluation expérimentale de ces bornes, le lecteur est renvoyé au chapitre 11.

Pour évaluer la qualité (théorique) d'une borne, j'utilise les ratios de performance absolu (ou du pire cas) définis classiquement de la manière suivante, pour un problème de minimisation $\mathcal{P}$.

Définition 7.1. (ratio de performance absolu) *Soit L une borne inférieure pour $\mathcal{P}$ et $L(I)$ sa valeur pour une instance I dont la valeur optimale est $OPT(I)$. Le* ratio de performance absolu de L *est :*

$$r(L) = \sup\left\{\frac{OPT(I)}{L(I)}\ ;\ I \text{ est une instance de } \mathcal{P}\right\}.$$

Le ratio de performance asymptotique de L *est :*

$$r_\infty(L) = \lim_{s\to\infty}\sup\left\{\frac{OPT(I)}{L(I)}\ ;\ I \text{ est une instance de } \mathcal{P} \text{ avec } OPT(I) \geq s\right\}.$$

Pour une borne supérieure U, une définition symétrique est utilisée en inversant le rapport $(U(I)/OPT(I)$ au lieu de $OPT(I)/L(I))$. Avec de telles définitions, le ratio de performance d'une borne est toujours plus grand que 1, et une borne est d'autant meilleure que son rapport en est proche.

7.1 Bornes de volume

Ces premières bornes sont très simples, mais néanmoins très utiles. Elles sont basées sur un argument de volume : la hauteur minimale doit être plus grande que le volume total réparti équitablement entre chaque boîte. Une telle borne peut cependant être facilement mise en défaut par des instances pourtant très simples : par exemple un unique objet très haut. Pour éviter cet écueil, il suffit de prendre en considération le fait que la hauteur $h_{\max}$ du plus grand objet est aussi une borne inférieure valide :

Proposition 7.1.

$$L_V = \max\left\{h_{\max}, \left\lceil \frac{\sum_{j=1}^{n} h_j size_j}{m} \right\rceil\right\}$$

est une borne inférieure valide.

La composante $\left\lceil \sum_{j=1}^{n} h_j size_j/m \right\rceil$ a déjà été signalée comme étant une borne inférieure valide (chapitre 2, proposition 2.1) : c'est la valeur de la relaxation continue du problème $B|size_j|H_{\max}$, *i.e.* lorsque les objets peuvent être coupés. Si les objets sont tous de hauteur unitaire, elle donne la valeur de la solution optimale (chapitre 3, proposition 3.5).

Cette borne possède une sœur jumelle :

Proposition 7.2.

$$U_V = \frac{\sum_{j=1}^{n} h_j size_j}{m} + h_{\max}$$

est une borne supérieure valide.

PREUVE — Soit S^* une solution optimale. Supposons que la plus grande différence de hauteur entre deux boîtes vérifie $\Delta^{S^*} \leq h_{\max}$. Appelons B_1 la plus grande boîte et B_2 la plus petite. Puisque $\Delta^{S^*} \leq h_{\max}$:

$$H_1 = H^*_{\max} \qquad \text{et} \qquad H_1 - H_2 = \Delta^{S^*} \leq h_{\max}. \tag{$\heartsuit$}$$

Si $H_1 = H_2$ alors toutes les boîtes sont de même hauteur et $H^*_{\max} = \frac{\sum_{j=1}^{n} h_j size_j}{m}$. Si $H_1 > H_2$ alors :

$$H_2 \leq \frac{\sum_{j=1}^{n} h_j size_j}{m}. \tag{$\diamondsuit$}$$

(autrement le volume total serait trop grand). De (♡) et (♢), il découle :

$$H^*_{\max} - \frac{\sum_{j=1}^{n} h_j size_j}{m} \leq h_{\max},$$

c'est-à-dire ce qui est annoncé.

Revenons sur l'hypothèse $\Delta^{S^*} \leq h_{\max}$. Celle-ci n'est pas une perte de généralité, puisqu'une telle solution optimale existe toujours. En effet, une solution optimale quelconque qui ne vérifie pas cette hypothèse possède un objet dans sa plus haute boîte qui n'est pas dans la boîte la plus basse ; déplacer cet objet, et réitérer l'opération tant que cela est nécessaire, permet d'obtenir une S^* comme souhaitée (le processus s'arrête : la quantité $\sum_{i_1<i_2} |H_{i_1} - H_{i_2}|$ est entière, strictement décroissante et bornée par 0). ⊗

Ces deux bornes sont très liées et permettent un encadrement relativement bon de la valeur optimale, notamment pour des grandes instances (avec beaucoup d'objets). Dans le pire cas, elles ont une performance correcte, et sont même asymptotiquement optimales si on impose une borne sur la hauteur des objets.

Proposition 7.3.

$$r_\infty(L_V) = r(L_V) = 2 = r(U_V) = r_\infty(U_V)$$

De plus, pour les instances vérifiant $h_{\max} \leq K$, pour K fixé :

$$r^K_\infty(L_V) = 1 = r^K_\infty(U_V).$$

PREUVE — Par définition $U_V \leq 2L_V$. Les deux bornes étant valides, cela implique immédiatement que $r(U_V) \leq 2$ et $r(L_V) \leq 2$.

Pour prouver que la performance de U_V est atteinte, considérons la famille d'instances suivante (voir aussi figure 7.1(a)). I_m est composée de m boîtes et de m objets de hauteur m et de largeur 1. La valeur optimale est trivialement de $H^*_{\max} = m$, alors que $U_V(I_m) = 2m$. On a donc : $r(U_V) = 2$. De plus, lorsque m tend vers l'infini, le rapport reste le même mais $H^*_{\max}$ tend vers l'infini ; on a donc aussi $r_\infty(U_V) = 2$.

Pour prouver que la performance de L_V est atteinte, considérons la famille d'instances suivante (voir aussi figure 7.1(b)). J_m est composée de m boîtes et $m+1$ objets de hauteur m et de largeur 1. La valeur optimale est $H^*_{\max} = 2m$ alors que $L_V(J_m) = m+1$. Lorsque m tend vers l'infini, le rapport tend donc vers 2, donc $r(L_V) = 2$, et comme la valeur optimale tend aussi vers l'infini : $r_\infty(L_V) = 2$.

Si l'on impose une borne constante K à la hauteur des objets alors $L_V \leq H^*_{\max} \leq U_V \leq L_V + K$. Les quantités L_V, U_V et $H^*_{\max}$ sont donc équivalentes et leurs rapports asymptotiquement égaux à un : $r^K_\infty(L_V) = 1 = r^K_\infty(U_V)$. ⊗

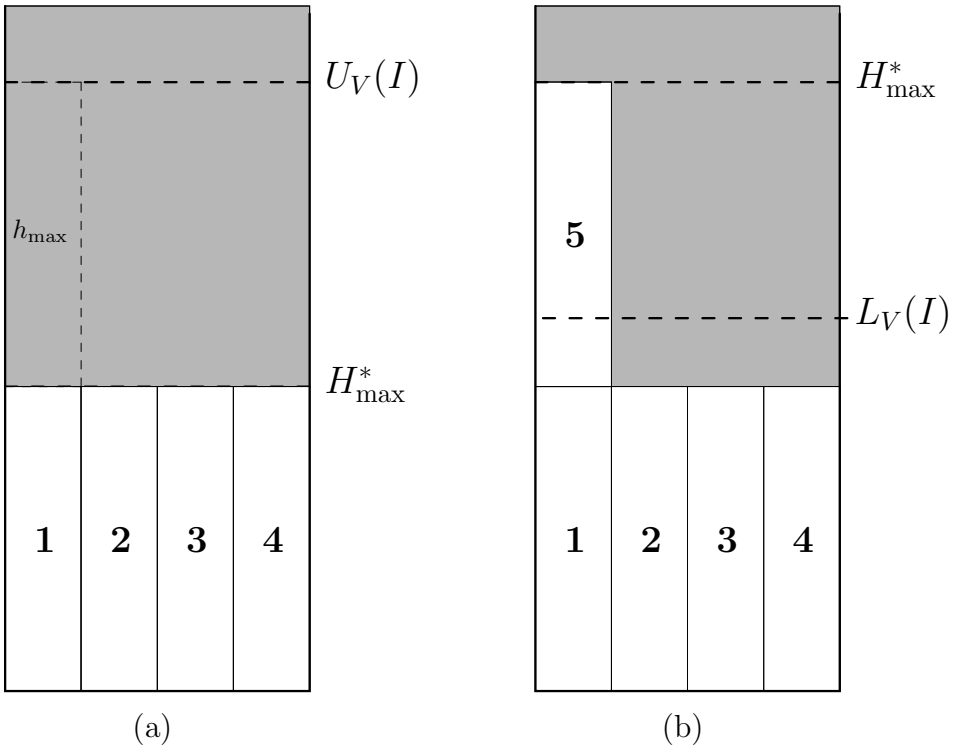


Figure 7.1 – Instances problématiques pour les bornes de volume.
(a) une instance pour laquelle U_V est mauvaise ;
(b) une instance pour laquelle L_V est mauvaise.

Dans la démonstration précédente, les « mauvaises » instances, qui atteignent les bornes, ont des objets de largeur unitaire : même pour des objets simples (donc pour le problème $P||C_{\max}$), les ratios sont ceux indiqués. Ce n'est pas surprenant : ces deux bornes ne tiennent compte que du volume et absolument pas de comment celui-ci est réparti.

7.2 Raisonnement énergétique

Pour le calcul de bornes inférieures pour les problèmes d'utilisation concurrente de ressources, une méthode est le « raisonnement énergétique », développée essentiellement pour le problème d'ordonnancement avec contraintes de ressources (RCPSP) [12, 27]. Cette approche consiste, pour un intervalle d'utilisation donné, à évaluer le besoin total minimum de la ressource, cumulé pour tous les utilisateurs : si ce besoin dépasse l'offre totale sur l'intervalle considéré, la solution n'est pas réalisable.

Cette démarche est particulièrement adaptée aux problèmes d'ordonnancement avec des précédences et des dates de début au plus tôt et au plus tard qui permettent de déterminer assez précisément les périodes d'exécution d'une tâche, et ainsi de connaître suffisamment bien le besoin minimal de cette tâche sur un intervalle quelconque. Nous ne sommes précisément pas dans ce cas. Toutefois, nous pouvons évaluer le nombre minimum d'objets dans une boîte, et en déduire une hauteur minimum de la plus haute boîte :

Proposition 7.4. *Soit :* $n_{\min} = \left\lceil \sum_{j=1}^{n} size_j/m \right\rceil$ *:*

$$L_{n_{\min}} = \sum_{j=1}^{n_{\min}} h_j$$

est une borne inférieure valide, pour $h_1 \leq h_2 \leq \ldots \leq h_n$*.*

Cette borne n'est pas adaptée aux grandes instances, et elle atteint rapidement ses limites, du fait que les plus petits objets sont systématiquement considérés en premier :

Proposition 7.5.

$$r(L_{n_{\min}}) = \infty$$

PREUVE — Il suffit d'exhiber une famille d'instances pour laquelle la performance de $L_{n_{\min}}$ tend vers l'infini. On prendra par exemple m boîtes et deux objets, l'un de largeur 1 et de hauteur K, l'autre de hauteur 1 est de largeur $m-1$. La valeur optimale est alors K, tandis que $L_{n_{\min}}$ donne 1 : le rapport tend vers l'infini avec K (voir figure 7.2(a)). ⊗

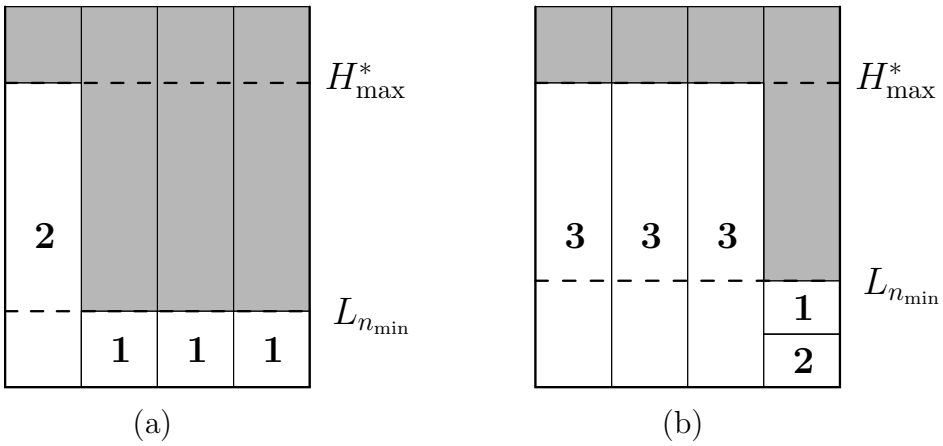


Figure 7.2 – Instances problématiques pour la borne $L_{n_{\min}}$.
(a) une instance mauvaise pour la version initiale de $L_{n_{\min}}$;
(b) une instance mauvaise pour $L_{n_{\min}}$ calculée par l'algorithme 7.1.

Une manière d'améliorer ces piètres performances théoriques est de considérer un sous-ensemble d'objets (p.ex. les plus gros) : on peut ainsi supprimer les objets un à un, en commençant par les plus petits, tant que cela n'affecte pas la valeur de $n_{\min}$. Pour calculer $L_{n_{\min}}$ efficacement, on utilisera ainsi l'algorithme 7.1.

Algorithme 7.1. Calcul de $L_{n_{\min}}$

- **entrées :** ensemble d'objets **O**, nombre de boîtes m -
0. trier les objets pour que $h_1 \leq h_2 \leq \ldots \leq h_n$
1. calculer $s = \sum_{j=1}^{n} size_j$
   calculer $t = m \lfloor s/m \rfloor$
2. $k = 1$
   tant que $s - size_k > t$ et $n - k > n_{\min}$ faire
      $s = s - size_k$
      $k = k + 1$
3. renvoyer $L = \sum_{j=k}^{k+n_{\min}} h_j$

Grâce à une telle amélioration, $L_{n_{\min}}$ ne se fait plus berner par les instances utilisées pour prouver la proposition 7.5. Mais des instances tout aussi simples sont suffisantes pour la tromper : toujours dans m boîtes, ranger un objet de hauteur K et de largeur $m-1$, et deux objets de largeur et hauteur unitaires (voir figure 7.2(b)).

Pour rendre la borne $L_{n_{\min}}$ véritablement intéressante, théoriquement[1] et pour toute instance, il faudrait répondre à la question suivante :

Question 7.1 : Est-il difficile de trouver le sous-ensemble d'objets telle que $L_{n_{\min}}$

[1]Les expériences du chapitre 11 montrent que le comportement en pratique de $L_{n_{\min}}$ est plus qu'honorable, quel que soit le type d'instances.

est la meilleure possible ? Quelle serait alors la performance de cette borne ?

Il me semble toutefois que cette borne est de toute façon, et par essence, inadaptée aux instances ayant un grand nombre de petits objets. Elle n'a d'ailleurs été proposée que pour détecter de petites instances « faussement difficiles », c'est-à-dire pour lesquelles il est facile de trouver une solution optimale mais plus difficile de s'en convaincre (typiquement, la borne de volume ne le permet pas) ; un rôle qu'elle remplit très bien, comme nous le verrons au chapitre 11.

7.3 Amélioration systématique des bornes

Afin d'améliorer les performances en pratique des bornes précédentes, ou bien de n'importe qu'elle autre borne, je présente dans cette section une adaptation à $B|size_j|H_{\max}$ de résultats classiques pour le Bin-Packing, récemment ressortis sur le devant de la scène : l'utilisation de fonctions dites « duales-réalisables » [37, 53] ou « redondantes » [27, 28]. Commençons par en expliciter l'idée directrice.

Généralement, lors d'un rangement, les petits objets peuvent être glissés à peu près n'importe où et ne posent pas de grosses difficultés, au contraire des grands. L'idée est donc d'augmenter le contraste, c'est-à-dire les différences de taille, afin de faire ressortir les objets problématiques et disparaître les autres.

On ne peut cependant pas se permettre n'importe quelle transformation des tailles. Afin d'avoir une information pertinente, l'approche utilisée dans le cadre du Bin-Packing est de transformer les tailles de manière à ce que toute solution de l'instance initiale soit aussi une solution pour l'instance transformée. En d'autres termes : il faut que quel que soit l'ensemble d'objets considéré, s'il tient dans une boîte avant transformation, il doit aussi tenir après. Pour le Bin-Packing, les boîtes sont de hauteur fixée 1, et l'on veut en minimiser le nombre[2] ; il suffit donc d'utiliser une fonction $f : [0, 1] \to [0, 1]$ vérifiant :

$$\forall X \text{ (fini)} \ : \sum_{x \in X} x \leq 1 \Longrightarrow \sum_{x \in X} f(x) \leq 1,$$

appelée *fonction duale-réalisable*. Ainsi, une solution optimale de l'instance initiale est une solution de l'instance transformée, et donc toute borne inférieure pour l'instance transformée est une borne inférieure pour l'instance initiale — ce que l'on cherche.

Pour $B|size_j|H_{\max}$, l'objectif est dual : le nombre de boîtes est connu, mais pas leur taille. Des bornes inférieures et supérieures sont cependant suffisantes, comme nous le verrons dans la suite. L'adaptation de la notion de fonction duale-réalisable s'effectue par le biais des fonctions redondantes[3], introduites dans [27, 28].

[2]La convention, dans la communauté du Bin-Packing, est de prendre des boîtes de hauteur unitaire, ce qui correspond à diviser toutes les hauteurs des objets par la hauteur H des boîtes. Il n'y a cependant qu'ici où je considère ce problème sous cet aspect, avec des dimensions non entières.

[3]Le terme correspond à une réalité : introduites dans le cadre des ordonnancements avec

Définition 7.2. (fonction redondante) *Soit $H \in \mathbb{N}$. Une fonction discrète $f : \{1, 2, \ldots, H\} \to \{0, 1, \ldots, H\}$ est dite* redondante pour H *si, pour tout ensemble fini X d'entiers positifs :*

$$\sum_{x \in X} x \leq H \Longrightarrow \sum_{x \in X} f(x) \leq H.$$

Pour une fonction entière f, pas nécessairement redondante, $f(I)$ est la transformation d'une instance I par f : chaque objet $O_j = (h_j, size_j)$ est remplacé par $f(O_j) = (f(h_j), size_j)$. Une convention similaire est utilisée pour une solution S et sa transformation $f(S)$. Il faut bien noter que seules les hauteurs des objets sont modifiées.

Avec ces définitions et conventions, comme pour le Bin-Packing, la transformation par une fonction redondante préserve les bornes inférieures :

Propriété 7.1. *Soit I une instance et $H^*_{\max}$ sa valeur optimale. Soit f une fonction redondante pour $H^*_{\max}$. Toute borne inférieure valide pour $f(I)$ l'est aussi pour I.*

preuve — Soit S^* une solution optimale pour I : $f(S^*)$ est une solution réalisable pour $f(I)$, par hypothèse, et sa valeur est inférieure à $H^*_{\max}$. Aussi toute borne inférieure pour $f(I)$ est plus petite que $H^*_{\max}$. ⊗

Bien que la valeur $H^*_{\max}$ soit inconnue, cette propriété peut être exploitée : en effet une borne inférieure L et une borne supérieure U suffisent. Ainsi :

Propriété 7.2. *Pour $L \leq H^*_{\max} \leq U$ et $\varepsilon \in \mathbb{N}$:*

$$\begin{array}{rccl} F_{\varepsilon,L,U} : & \{0, 1, \ldots, H^*_{\max}\} & \to & \{0, 1, \ldots, H^*_{\max}\} \\ & x & \mapsto & \begin{cases} \max(L, x) & \text{si } x \geq U - \varepsilon \\ x & \text{si } U - \varepsilon > x > \varepsilon \\ 0 & \text{si } \varepsilon \geq x \end{cases} \end{array}$$

est redondante pour $H^*_{\max}$.

preuve — Ceci est une adaptation directe d'un résultat similaire de [53]. Il suffit de remarquer que les objets dont la taille augmente ne peuvent être rangés qu'avec des objets dont la taille devient nulle. ⊗

Cette fonction $F_{\varepsilon,L,U}$ permet d'améliorer n'importe quelle borne inférieure :

contraintes de ressources, ces fonctions correspondent à l'ajout d'une ressource supplémentaire qui ne modifie pas la réalisabilité des instances.

Proposition 7.6. *Soient L_1, L_2 deux bornes inférieures et U_1 une borne supérieure :*

$$L_+^{L_2}(I) = \max_{\varepsilon \in \mathbb{N}} L_2(F_{\varepsilon,L_1(I),U_1(I)}(I))$$

est une borne inférieure valide pour I et

$$L_+^{L_2}(I) \geq L_2(I).$$

PREUVE — L_1 et U_1 étant des bornes valides, la propriété 7.2 implique que, pour tout ε, la fonction $F_{\varepsilon,L_1(I),U_1(I)}$ est redondante pour $H^*_{\max}$; la propriété 7.1 implique alors que, pour tout ε, $L_2(F_{\varepsilon,L_1(I),U_1(I)}(I))$ est une borne inférieure valide. Indépendamment de L_1 et U_1, le calcul de $L_+^{L_2}$ inclut celui de L_2 (pour $\varepsilon = 0$) : $L_+^{L_2}(I) \geq L_2(I)$. ⊗

Pour le calcul de $L_+^{L_2}$, la valeur de ε n'a pas à parcourir l'ensemble des entiers et il suffit de considérer $\varepsilon \in \{0, 1, \ldots, h_{\max}\}$. En fait, seules les valeurs pour lesquelles la transformation d'un objet change sont intéressantes : cela donne au plus $2n$ valeurs, chaque objet ne pouvant être transformé qu'au plus deux fois.

Si l'on prend comme borne de référence celle basée sur le volume, L_V (section 7.1), le calcul de $L_+^{L_V}$ peut être effectué en un seul parcours de la liste des objets, si ceux-ci sont triés par hauteur.

Proposition 7.7. *L'algorithme 7.2 calcule $L_+^{L_V}$ en temps $\mathcal{O}(n)$ si les objets sont initialement triés par hauteurs décroissantes.*

Algorithme 7.2. Calcul de $L_+^{L_V}$

- **entrées :** `ensemble d'objets` **O** $(h_1 \geq h_2 \geq \ldots \geq h_n)$, `nombre de boîtes` m, `bornes` L_1 `et` U_1 -

1. $V = \sum_{j=1}^{n} size_j h_j$ — *initialisations*
   $L = \max(L_1, h_1, \lceil V/m \rceil)$
   $g = 1$ `et` $p = n$
2. `tant que` $g \leq p$ `faire` — *calcul*
   a. $\varepsilon = U_1 - h_g$
      $V = V + size_g(\max(L_1, h_g) - h_g)$
      $g = g + 1$
   b. `tant que` $p > g$ `et` $h_p > \varepsilon$ `faire`
      $V = V - size_p h_p$
      $p = p - 1$
   c. $L = \max(L, \lceil V/m \rceil$
3. `renvoyer` L

PREUVE — L'algorithme est basé sur un découpage de l'ensemble des objets en trois : les grands ($G = \{O_j \ ; \ h_j \geq U_1 - \varepsilon\}$), les moyens ($M = \{O_j \ ; \ U_1 - \varepsilon > h_j > \varepsilon\}$) et les petits ($P = \{O_j \ ; \ \varepsilon \geq h_j\}$). Les objets étant triés, il suffit de retenir l'indice du premier et du dernier éléments de M : c'est le rôle de g et p, selon le schéma :

$$\underbrace{O_1 \geq \ldots \geq O_{g-1}}_{G} \geq \underbrace{O_g \geq \ldots \geq O_p}_{M} \geq \underbrace{O_{p+1} \geq \ldots \geq O_n}_{P}.$$

V est le volume total pour les hauteurs courantes ; L est la meilleure borne inférieure déterminée jusque là. Initialement, tous les objets sont considérés comme « moyens » ($\varepsilon = 0$).

La borne ne peut être améliorée que lorsqu'un objet passe de « moyen » à « grand » : aussi suffit-il d'ajouter, par ordre de taille, les objets à G, jusqu'à ce que M soit vide, en ajustant en même temps P. C'est ce que réalise l'étape 2. En effet, O_g est, en début d'itération, le prochain candidat pour devenir « grand ». Pour cela, il faut avoir $\varepsilon = U_1 - h_g$ puis mettre à jour la valeur du volume (étape 2a). La valeur ε ayant changé, il faut mettre à jour P et modifier V en conséquence (étape 2b). Les ensembles G, M et P sont alors à jour, pour cette nouvelle valeur pertinente de ε : on peut calculer la valeur du volume correspond (étape 2c).

Dans cette procédure, un objet n'est considéré qu'une seule fois, et chaque opération est de temps constant ; l'algorithme se déroule en temps $\mathcal{O}(n)$. ⊗

Pour le problème du BIN-PACKING, Fekete et Schepers [53] rapportent d'excellents résultats pour cette approche ; ainsi, la borne dont $L_+^{L_V}$ est l'adaptation permet de diviser par 10 l'écart relatif avec la meilleure borne supérieure connue, par rapport à l'utilisation de la seule borne de volume. Par le biais d'autres fonctions duales-réalisables, et par leur composition, les résultats sont encore meilleurs.

Pour le problème qui nous préoccupe, le fait de devoir utiliser des bornes pour estimer la taille d'une boîte à l'optimum nuit grandement aux performances. Ainsi, on constate que $L_+^{L_V}$ n'apporte qu'un gain marginal (voir chapitre 11).

Il reste toutefois de nombreuses autres combinaisons de bornes à tester et de fonctions redondantes à proposer, ce qui laisse ouvertes des perspectives d'amélioration de la méthode.

7.4 Synthèse

Différentes bornes inférieures ont été proposées pour $B|size_j|H_{\max}$, correspondant à différentes attentes. Ainsi dispose-t-on maintenant de bornes efficaces, tant sur un plan théorique que pratique : L_V garantie d'assez bornes performances, et est très efficace ; $L_{n_{\min}}$ permet d'écarter certaines instances pathologiques qui échappent à L_V. Ces bornes apparaissent ainsi très complémentaires. Pour avoir une idée de leur qualité réelle, la lecture du chapitre 11 est obligatoire.

Au niveau des bornes supérieures, les meilleures développées sont en fait les solutions des algorithmes à performances garanties du chapitre suivant. Par ailleurs,

du fait de ces garanties, ces algorithmes fournissent également des bornes inférieures (d'autant meilleures que la solution correspondante est mauvaise !).

Pour continuer cette étude, je privilégierais deux pistes. La première est la poursuite de l'utilisation des fonctions redondantes ; il y a là matière à amélioration. Je pointerais notamment les travaux très complémentaires de Carlier et Néron [27, 28] sur les problèmes avec contraintes de ressources, où la notion de « fonction redondante » est introduite, et où certaines fonctions dominantes sont présentées. Ces auteurs proposent aussi la « relaxation préemptive multiplement élastique », qui s'avère être très proche des problèmes multiboîtes.

La deuxième direction est qu'il faudrait prendre davantage en compte le caractère... multiboîte des objets ! en effet, pour l'instant, seule la borne $L_{n_{\min}}$ tire parti des largeurs, et ne le fait que de manière globale. Des considérations plus précises permettraient sans doute d'effectuer de meilleurs regroupements d'objets devant nécessairement être rangés dans les mêmes boîtes.

Heuristiques à performances garanties

Si tu prétends être bon, le monde te prend très au sérieux.
— Oscar Wilde

Après les bornes (inférieures), j'aborde maintenant la deuxième étape de la résolution de $B|size_j|H_{\max}$. Il s'agit d'avoir des heuristiques rapides et avec de bonnes performances (à la fois garanties et en pratique).

Dans ce chapitre, je propose cinq méthodes pour résoudre $B|size_j|H_{\max}$. Les trois premières sont gloutonnes et basées sur des règles particulières de placement d'un objet ; la quatrième est tirée du principe de « diviser pour régner » ; quant à la cinquième, c'est avant tout une méthode de recherche locale. Pour la plupart de ces méthodes, considérer un ordre particulier des objets, généralement par hauteurs décroissantes, améliore les performances. Il faut ainsi bien avoir en tête que les ensembles d'objets considérés sont ordonnés, et que l'ordre considéré a une importance.

Toutes ces méthodes ont des garanties de performance, résumées dans la dernière section de ce chapitre, et leurs aptitudes pratiques sont analysées au chapitre 11. Avant de présenter la première de ces méthodes, je donne une garantie pour une solution quelconque, indépendamment de la manière dont elle a été construite.

8.1 Garanties pour une solution quelconque

Indépendamment d'une quelconque manière de la construire, une garantie sur la qualité d'une solution est possible. C'est même, en fait, cette garantie qui sert à prouver les performances de plusieurs des algorithmes des sections suivantes. Elle est basée sur Δ, la différence de hauteur maximale entre deux boîtes de la solution considérée :

Lemme 8.1. *Soit S une solution réalisable d'une instance de $B|size_j|H_{\max}$:*

$$H_{\max} \leq H^*_{\max} + \frac{m-1}{m}\Delta.$$

Une version préliminaire de la section 8.2 (algorithme AMP) a été publiée dans [95], et présentée à [93, 95]. L'algorithme ADR a été présenté à [92].

PREUVE — Il y a dans S au moins une boîte de hauteur $H_{\max}$; les autres boîtes sont de hauteur au moins $H_{\max} - \Delta$. En raison du volume total des objets :

$$\sum_{j=1}^{n} h_j size_j \geq H_{\max} + (m-1)(H_{\max} - \Delta) = mH_{\max} - (m-1)\Delta.$$

La proposition 7.1 implique que $H^*_{\max} \geq \sum_{i=1}^{n} h_j size_j/m$. En injectant cela dans l'inégalité précédente, il vient :

$$H^*_{\max} \geq H_{\max} - \frac{m-1}{m}\Delta.$$

⊗

Armés de ce résultat, nous pouvons nous attaquer à l'analyse de quelques heuristiques.

8.2 Meilleur placement

Les algorithmes de meilleur placement sont des méthodes simples mais très efficaces. Cette section les décrits et en dégage de nombreuses caractéristiques.

8.2.1 La règle du meilleur placement

La règle de *meilleur placement* (MP) consiste à toujours mettre un objet O_j dans les $size_j$ boîtes les moins pleines. Les objets sont considérés de manière séquentielle, et rangés immédiatement.

Cette règle peut être vue comme une adaptation aux problèmes de rangement d'objets multiboîtes de la méthode d'« ordonnancement par liste » proposée par Graham pour l'ORDONNANCEMENT SUR MACHINES PARALLÈLES [64]. Elle est aussi très similaire à la règle du « pire placement » pour le BIN-PACKING [37] ; il n'y a cependant pas de contradiction de terminologie : les objectifs considérés sont duaux.

À partir de la règle MP, un algorithme glouton et en-ligne se construit facilement. Une implémentation naïve consistant à trier les boîtes à chaque itération pour trouver les boîtes les plus vides pour l'objet suivant aboutit à un algorithme en temps $\mathcal{O}(nm \ln m)$. En fait, il est inutile de retrier complètement les boîtes : en les supposant initialement rangées par ordre croissant, il suffit de fusionner deux sous-listes de boîtes déjà triées (les boîtes ayant reçu l'objet et les autres). Cela ne nécessite qu'un parcours de la liste des boîtes (voir figure 8.1) et la méthode peut ainsi être implémentée en $\mathcal{O}(nm)$, comme le réalise l'algorithme 8.1 (AMP).

Cette complexité de $\mathcal{O}(nm)$ est optimale pour un algorithme construisant effectivement une solution, puisque c'est aussi la taille de cette dernière. Elle s'avère aussi être excellente en pratique (voir le chapitre 11 et l'annexe B.2 pour plus de détails). D'un point de vue théorique, toutefois, rappelons qu'une telle complexité n'est que pseudo-polynomiale si le nombre de boîtes m n'est pas fixé. Dans ce cas-là, de toutes

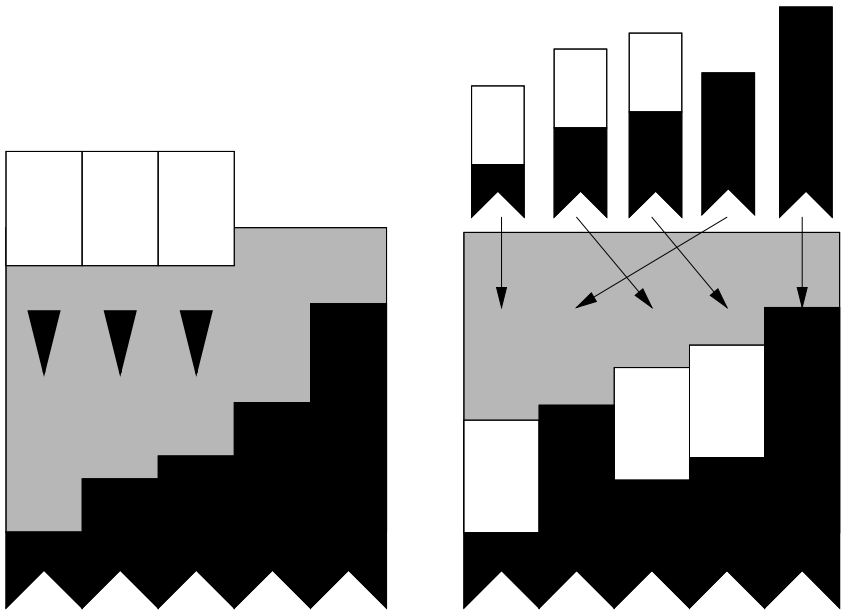


Figure 8.1 – Ajout d'un objet selon la règle MP.
(a) insertion d'un objet ; (b) tri des boîtes.

Algorithme 8.1. AMP

- **entrées :** ensemble d'objets $\mathbf{O}$, nombre de boîtes m -

1. pour $O_j \in \mathbf{O}$ faire
   a. ranger O_j dans $B_1, B_2, \ldots, B_{size_j}$ — *ajout de l'objet* O_j
   b. soit $k = 1, l = size_j + 1$ — *tri des boîtes (auxiliaires* C_i*)*
      pour $i = 1$ à m faire
        si $k >= size_j$ alors $C_i = B_l$ et $l = l + 1$
        sinon si $l >= m$ alors $C_i = B_k$ et $k = k + 1$
        sinon si $H_k \leq H_l$ alors $C_i = B_k$ et $k = k + 1$
        sinon $C_i = B_l$ et $l = l + 1$
   c. pour $i = 1$ à m faire $B_i = C_i$

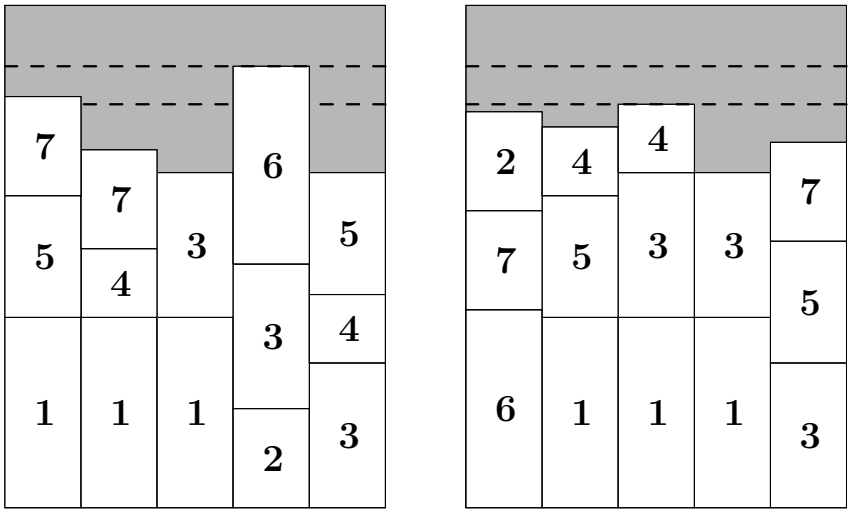


Figure 8.2 – Influence de l'ordre des objets sur la règle MP.
(a) ordre naturel ; (b) par hauteurs décroissantes.

façons, le simple fait de construire une solution réalisable implique un algorithme au mieux pseudo-polynomial[1].

Dans l'algorithme AMP, l'ordre dans lequel les objets sont considérés est essentiel : ainsi, pour différentes listes L, l'algorithme 8.1 pourra construire des solutions très différentes (voir figure 8.2). Un point important est de savoir s'il existe toujours, pour chaque instance I, un ordre particulier des objects $<_I$ telle que AMP appliqué avec cet ordre produise une solution optimale. Je n'ai réussi n'y à confirmer ni à infirmer cela, mais de nombreux tests expérimentaux me poussent à la conjecture suivante :

Conjecture 8.1. *Pour toute instance I de $B|size_j|H_{\max}$, il existe un ordre des objets $<_I$ tel que AMP appliqué pour cet ordre donne une solution optimale.*

Bien sûr, même dans l'affirmative, une telle propriété ne permettrait pas de résoudre rapidement et optimalement $B|size_j|H_{\max}$ (un phénomène identique existe pour le problème de la coloration des sommets d'un graphe). Toutefois, elle permettrait de restreindre considérablement l'espace de recherche : l'ensemble des permutations de n éléments à la place de l'ensemble de toutes les positions possibles pour chaque objet. Ceci permettrait de meilleures garanties et/ou performances pour les extensions basées sur la règle MP (voir les chapitres 9 et 10).

Indépendamment de tout ordre, nous pouvons toutefois prouver certaines garanties (section 8.2.2). Ces garanties peuvent être améliorées en imposant un ordre

[1]Cette taille de $\mathcal{O}(nm)$ pour une solution est nécessaire pour pouvoir coder toutes les solutions ; si l'on se restreint à certains types de solutions, comme par exemple celles où les différents bouts d'un même objet sont mis dans des boîtes adjacentes, la taille peut être grandement diminuée : ainsi peut-on proposer des algorithmes de résolution polynomiaux, même pour m quelconque.

précis : que les objets soient rangés par ordre décroissant des hauteurs (section 8.2.3).

8.2.2 Performances de la règle MP

La règle MP permet d'avoir des garanties sur les solutions construites, comme le montrent les résultats suivants. En fait, il n'est pas nécessaire qu'une solution soit entièrement construite selon cette règle : on peut imposer une affectation initiale pour certains objets, puis compléter avec les autres objets selon la règle MP. Pour cette raison, j'introduis les notions de différences de hauteur initiale : $\Delta_{i_1 i_2}(0)$ est la différence de hauteur initiale (*i.e.* avant l'application de la règle MP) entre les boîtes B_{i_1} et B_{i_2} ; $\Delta(0)$ est la différence de hauteur initiale maximale.

La règle MP consistant en mettre un objet dans les boîtes les moins remplies, il n'est pas étonnant qu'elle implique un contrôle de la différence de hauteur entre les boîtes, et qu'elle tende même à réduire celle-ci :

Propriété 8.1. *Soient n objets à ranger selon la règle MP. Soit $\Delta_{i_1 i_2}(k)$ la différence de hauteur entre les boîtes B_{i_1} et B_{i_2} après que le $k^{ème}$ objet a été rangé :*

$$\forall k = 1, 2, \ldots, n : \Delta_{i_1 i_2}(k) \leq \max(\Delta_{i_1 i_2}(k-1), h_k - \Delta_{i_1 i_2}(k-1)).$$

PREUVE — Soit $H_i(k)$ la hauteur de la boîte B_i après que l'objet O_k a été rangé. Par définition de $\Delta_{i_1 i_2}(0)$, la propriété est vraie pour $k = 1$; supposons-la vraie pour $0 \leq k - 1 < n$. Si O_k est rangé dans B_{i_1} et B_{i_2}, ou dans aucune des deux, alors $\Delta_{i_1 i_2}(k) = \Delta_{i_1 i_2}(k-1)$. S'il n'est rangé que dans l'une (la plus petite, disons B_{i_1}), alors : si $H_{i_2}(k) \geq H_{i_1}(k)$ alors $\Delta_{i_1 i_2}(k) = \Delta_{i_1 i_2}(k-1) - h_k \leq \Delta_{i_1 i_2}(k-1)$; sinon $\Delta_{i_1 i_2}(k) = h_k - \Delta_{i_1 i_2}(k-1)$. ⊗

Cette propriété peut être reformulée de la manière suivante :

Propriété 8.2. *Avec les hypothèses de la propriété 8.1 :*
(a) *si $\Delta_{i_1 i_2}(k) \leq h_{\max}$ alors $\forall k', k \leq k' \leq n : \Delta_{i_1 i_2}(k') \leq h_{\max}$;*
(b) *si $\Delta_{i_1 i_2}(n) \geq h_{\max}$ alors $\forall k, k \leq n : \Delta_{i_1 i_2}(k) \geq h_{\max}$.*

De plus, le contrôle de la différence de hauteur n'est pas seulement local, entre deux boîtes, mais est aussi global, sur la différence de hauteur maximale :

Propriété 8.3. *Soient n objets à ranger selon la règle MP. Soit $\Delta(k)$ la différence de hauteur maximale entre deux boîtes après que le $k^{ème}$ objet a été rangé :*

$$\forall k = 1, 2, \ldots, n : \Delta(k) \leq \max(\Delta(k-1), h_k).$$

PREUVE — Ceci est une conséquence directe de la propriété 8.1. J'utilise les mêmes notations. Pour tout k, il existe deux boîtes B_a, B_b telles que $\Delta(k) = H_a(k) - H_b(k) = \Delta_{ab}(k)$. Nous avons alors :

$$\begin{aligned}\Delta(k) &= \Delta_{ab}(k)\\ &\leq \max_{i_1 i_2}\{\Delta_{i_1 i_2}(k-1), h_k - \Delta_{i_1 i_2}(k-1)\}\\ &\leq \max(\max_{i_1 i_2}\{\Delta_{i_1 i_2}(k-1)\}, h_k)\\ &\leq \max(\Delta(k-1), h_k).\end{aligned}$$

⊗

Sur la base de ce contrôle de Δ par la règle MP découle une garantie sur les solutions construites selon cette règle :

Théorème 8.1. *Soit une solution construite selon la règle MP :*

$$H_{\max} \leq H^*_{\max} + \frac{m-1}{m}\Delta \qquad \text{avec} \qquad \Delta \leq \max(h_{\max}, \Delta(0))$$

et cette borne est atteinte.

PREUVE — D'après la propriété 8.3 : $\Delta = \Delta(n) \leq \max(h_{\max}, \Delta(0))$. Le lemme 8.1 permet alors de conclure la validité de la borne.

Pour montrer que cette borne est atteinte, considérons l'instance suivante, dont les objets sont tous de largeur unitaire : m boîtes et $m^2 + 1$ objets de hauteur $h_1 = h_2 = \ldots = h_{m^2} = 1$ et $h_{m^2+1} = m$. Pour une telle instance, et un tel ordre, AMP construit une solution de valeur $H_{\max} = 2m = H^*_{\max} + \frac{m-1}{m}h_{\max}$ (voir figure 8.3). ⊗

Cette garantie a plusieurs conséquences qui renseignent sur le comportement de AMP en pratique. Tout d'abord :

Corollaire 8.1.1. *AMP est asymptotiquement optimal pour les instances uniformément bornées de $Bm|size_j|H_{\max}$ (*i.e. *il existe des bornes uniformes $h_{\max} \geq h_{\min} > 0$ telles que tout objet O_j vérifie : $h_{\max} \geq h_j \geq h_{\min}$).*

PREUVE — La borne supérieure sur h_j garantit que Δ est borné, tandis que la borne inférieure garantit que $H^*_{\max}$ ne l'est pas, lorsque le nombre d'objets tend vers l'infini :

$$1 \leq \frac{H_{\max}}{H^*_{\max}} \leq 1 + \frac{\Delta}{H^*_{\max}} \qquad \text{et} \qquad \frac{\Delta}{H^*_{\max}} \rightsquigarrow_{n\to\infty} 0.$$

⊗

Ce corollaire est important en pratique : il implique que les grosses instances (avec beaucoup d'objets) seront très bien résolues, ce que les tests expérimentaux confirment (voir chapitre 11).

Pour ce qui est d'une performance multiplicative, il suffit de remarquer que $h_{\max} \leq H^*_{\max}$ pour en déduire :

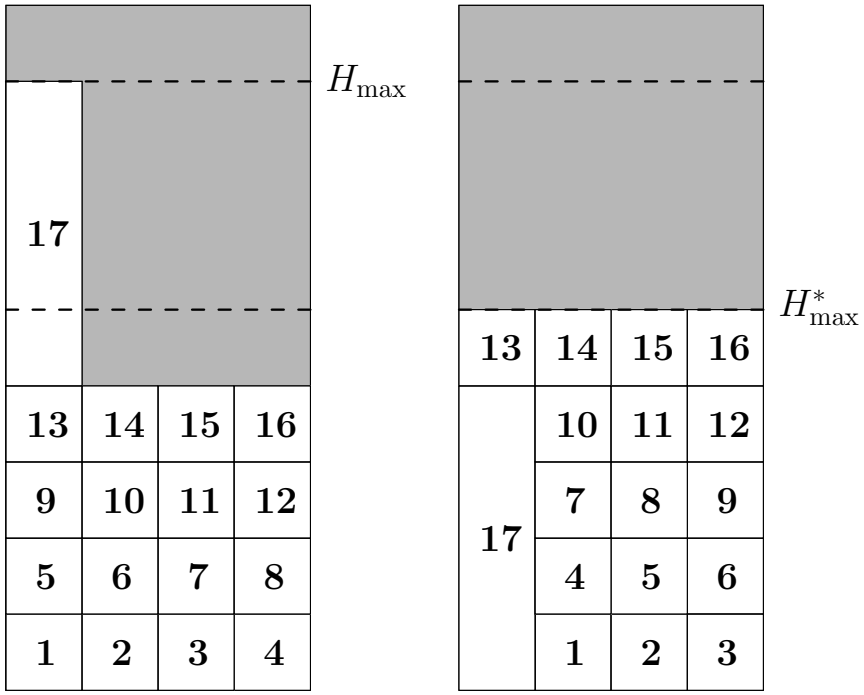


Figure 8.3 – Un exemple d'instance pour lequel AMP est mauvais.
(a) AMP selon l'ordre naturel ; (b) une solution optimale.

Corollaire 8.1.2. *Soit une solution complètement construite (*i.e. $\Delta(0) = 0$*) selon la règle MP :*

$$H_{\max} \leq (2 - \frac{1}{m})H^*_{\max}$$

et cette borne est serrée.

Cette borne est en fait la même que celle prouvée par Graham pour des objets de largeur unitaire [64]. Il produisit des exemples où cette borne est atteinte.

Dans le cas d'objets de hauteur unitaire, le résultat devient même :

Corollaire 8.1.3. *AMP est optimal pour* $B|size_j, h_j = 1|H_{\max}$.

Ainsi, AMP est un algorithme polynomial et optimal pour des objets de hauteur unitaire et un nombre de boîtes fixé. Par une implémentation particulière de la règle MP, possible pour des objets unitaires, on peut toutefois faire mieux et s'affranchir de cette contrainte d'un m fixé : c'est ce qui est réalisé par l'algorithme 3.1 vu précédemment (voir chapitre 3, section 3.3).

Comme cela a été signalé au début de cette section, des objets peuvent avoir une pré-affectation. Typiquement, on peut vouloir placer astucieusement les gros objets (généralement les plus embêtants), puis compléter avec les autres selon la règle MP.

Dans ce cas-là, quelles garanties a-t-on quant à la qualité de la solution finale ? Le théorème suivant répond à cette question.

Théorème 8.2. *Soit I une instance de $B|size_j|H_{\max}$ et $\mathbf{O}_1, \mathbf{O}_2$ une partition de son ensemble d'objets $\mathbf{O}$. Soit $S(\mathbf{O}_1)$ une pré-affectation de $\mathbf{O}_1$ et $H^*_{\max}(\mathbf{O}|\mathbf{O}_1)$ la valeur optimale d'une solution de I qui respecte cette affectation. Soit S la solution obtenue en complétant $S(\mathbf{O}_1)$ par $\mathbf{O}_2$ selon la règle MP :*

$$(a) \qquad H^S_{\max} \leq H^*_{\max} + \frac{m-1}{m}\max(h_{\max}(\mathbf{O}_2), \Delta(\mathbf{O}_1))$$

et $$(b) \qquad H^S_{\max} \leq H^*_{\max}(\mathbf{O}|\mathbf{O}_1) + \frac{(m-1)^2}{m} h_{\max}(\mathbf{O}_2)$$

avec $\Delta(\mathbf{O}_1) = \max\{|H_{i_1}(S(\mathbf{O}_1)) - H_{i_2}(S(\mathbf{O}_1))| \ ; \ 1 \leq i_1 < i_2 \leq m\}$
et $h_{\max}(\mathbf{O}_2) = \max\{h_j \ ; \ O_j \in \mathbf{O}_2\}$.

PREUVE — Puisque l'ordre au sein d'une boîte est sans importance, je suppose que les objets de $\mathbf{O}_1$ occupent le fond des boîtes.

La partie (a) n'est qu'une reformulation du théorème 8.1. Pour la partie (b), la preuve se fait par induction sur le nombre de boîtes m. L'hypothèse d'induction est que le théorème 8.2(b) est vrai pour toute instance avec $k < m$ boîtes. Il suffit donc de prouver que cela implique qu'il est aussi vrai pour m boîtes.

Pour $m = 1$, il n'existe aucune instance avec $k < m$ boîtes : l'hypothèse d'induction est donc vérifiée. Si $m = 2$, les solutions n'ont qu'une unique boîte. Elles sont donc toutes nécessairement optimales et vérifient l'hypothèse d'induction.

Soit I une instance quelconque de m boîtes. Tout d'abord si, en fin de procédure, $\Delta \leq (m-1)h_{\max}(\mathbf{O}_2)$ alors le lemme 8.1 permet de conclure immédiatement, sans même l'hypothèse d'induction, que :

$$H^S_{\max} \leq H^*_{\max} + \frac{(m-1)^2}{m} h_{\max}(\mathbf{O}_2).$$

Ceci implique le résultat annoncé, la valeur $H^*_{\max}$ (optimale pour I) étant inférieure à $H^*_{\max}(\mathbf{O}|\mathbf{O}_1)$.

Plaçons-nous maintenant dans le deuxième cas : $\Delta > (m-1)h_{\max}(\mathbf{O}_2)$, et supposons, quite à les renommer, que les boîtes sont triées par ordre décroissant de leurs tailles en fin de procédure.

Puisque $\Delta > (m-1)h_{\max}(\mathbf{O}_2)$, il existe un indice $i \in \{1, 2, \ldots, m-1\}$ tel que $H_i > H_{i+1} + h_{\max}(\mathbf{O}_2)$. La propriété 8.2$(b)$ implique alors qu'à chaque étape de la procédure, pour $1 \leq i_1 \leq i < i_2 \leq m$: $H_{i_1} < H_{i_2}$. Il s'en suit que tout objet mis dans une boîte $B_{i_1}, i_1 \leq i$ a aussi été mis dans toutes les boîtes $B_{i_2}, i_2 < i$.

En conséquence la solution S est telle que chaque objet O_j de largeur $size_j \leq m - i$ est complètement rangé dans les boîtes $B_{i+1}, B_{i+2}, \ldots, B_m$ et que chaque objet O_j de largeur $size_j > m - i$ occupe toutes les boîtes $B_{i+1}, B_{i+2,}, \ldots, B_m$ et $size_j - (m-i)$ des boîtes de $B_1, B_2, \ldots, B_i$. Ceci signifie entre autre que, pour une solution satisfaisant l'affectation de $S(\mathbf{O}_1)$, on ne peut pas mettre moins (en volume) dans $B_1, B_2, \ldots, B_i$.

Aussi le problème peut-il être réduit aux i premières boîtes : l'ensemble $\mathbf{O}_2$ est réduit aux objets tels que $size_j > m - i$ en remplaçant $size_j$ par $size_j^R = size_j - (m - i)$. La solution S est réduite à S^R en ne gardant que les i premières boîtes (dans cette réduction, la partie pré-affectée est $\mathbf{O}_1^R$, et $\mathbf{O}_2^R$ est rangé selon la règle MP).

Puisque maintenant $i < m$, l'hypothèse d'induction s'applique pour le problème réduit :

$$H_{\max}^{S^R} \leq H_{\max}^*(\mathbf{O}^R|\mathbf{O}_1^R) + \frac{(i-1)^2}{i} h_{\max}(\mathbf{O}_2^R).$$

En outre $H_{\max}^{S^R} = H_1 = H_{\max}^S$ et $H_{\max}^*(\mathbf{O}^R|\mathbf{O}_1^R) \leq H_{\max}^*(\mathbf{O}|\mathbf{O}_1)$ puisqu'on ne peut pas mettre moins dans $B_1, B_2, \ldots, B_i$. Ainsi

$$H_{\max}^S = H_{\max}^{S^R} \leq H_{\max}^*(\mathbf{O}^R|\mathbf{O}_1^R) + \frac{(i-1)^2}{i} h_{\max}(\mathbf{O}_2^R)$$

et comme $\frac{(i-1)^2}{i} < \frac{(m-1)^2}{m}$ et $h_{\max}(\mathbf{O}_2^R) \leq h_{\max}(\mathbf{O}_2)$, on obtient

$$H_{\max}^S \leq H_{\max}^*(\mathbf{O}|\mathbf{O}_1) + \frac{(m-1)^2}{m} h_{\max}(\mathbf{O}_2),$$

ce qui conclut la preuve. ⊗

Le résultat précédent revient à dire que la différence de hauteur entre deux boîtes consécutives ne peut être supérieure à $h_{\max}(\mathbf{O}_2)$ que si la pré-affectation de $\mathbf{O}_1$ l'impose. La version donnée est toutefois pessimiste : un tel écart, de $h_{\max}(\mathbf{O}_2)$, entre chaque boîte est toléré. Il est ainsi sans doute possible de raffiner cette propriété ; d'ailleurs, je ne connais pas d'exemple qui atteigne cette borne. Je préfère cependant me tourner vers une autre amélioration : que se passe-t-il si l'affectation de $\mathbf{O}_1$ est optimale (*i.e.* si $S(\mathbf{O}_1)$ est une solution optimale pour $\mathbf{O}_1$) ?

Pour répondre à cette question, j'utilise la notion de réduction d'une solution (d'une instance, *etc.*) suivante, qui permet de diminuer le nombre de boîtes de m à $m - 1$, de manière similaire à ce qui a été fait dans la preuve précédente.

Définition 8.1. (réduction) *Soit I une instance de $B|size_j|H_{\max}$ avec m boîtes ; soit S une solution pour I. La* réduction R de S par B_i *(pour $i \in \{1, 2, \ldots, m\}$ donné) est la solution S à laquelle la boîte B_i a été supprimée, avec tout ce qu'elle contient. L'instance I^R est alors définie comme étant I privée du contenu de B_i (*i.e. *la largeur de $O_j \in B_i$ est diminuée de 1 ; cela donne l'ensemble d'objets réduits $\mathbf{O}^R$) et $m - 1$ boîtes.*

Avec de telles définitions, une réduction R est une solution réalisable de I^R. De plus, la réduction de solution conserve l'optimalité et la construction selon MP, comme l'indiquent les deux lemmes suivants.

Lemme 8.2. *Soit I une instance de $B|size_j|H_{\max}$ avec m boîtes. Soit S^* une solution optimale pour I. Soient $B_i \in S^*$ telle que $H_i < H_{\max}^{S^*}$ et R la réduction de S^* par B_i. Alors :*

(a) R est une solution optimale de I^R.
(b) si S' est une solution optimale de I^R,
alors $S' \cup B_i$ est une solution optimale de I.
(c) I et I^R ont la même valeur optimale.

PREUVE — Affirmation (a). Si S^R n'est pas optimale, alors considérons une solution optimale pour I^R ; en ajoutant à celle-ci la boîte B_i et son contenu, on construit une solution réalisable de I, strictement meilleure que S^*, ce qui est exclu.

Affirmations (b) et (c). D'après (a), S^R est une solution optimale de I^R et a donc la même valeur que S'. Par conséquent $H_i < H_{\max}^{S'}$ et ainsi $S' \cup B_i$ est une solution réalisable de I de même valeur que S^*. En d'autres termes : $S' \cup B_i$ est optimale et les valeurs optimales de I et I^R sont égales. ⊗

Lemme 8.3. *Soit S une solution construite selon la règle MP, pour un ensemble ordonné $\mathbf{O}$ donné (les boîtes n'étant pas nécessairement initialement vides). Soit R une réduction de S pour une certaine boîte B_i. Dans ce cas :*

$$R = \mathrm{AMP}(\mathbf{O}^R),$$

*où $\mathbf{O}^R$ est la réduction de l'ensemble $\mathbf{O}$ (*i.e. *même ordre des objets, mais largeurs réduites). L'égalité signifie non seulement que les deux solutions ont la même valeur, mais aussi la même structure (*i.e. *les objets sont rangés dans les mêmes boîtes, aux objets identiques près).*

PREUVE — La preuve est récursive : la première partie (n'appartenant pas à B_i) du premier objet est placé dans la boîte la plus basse, qui est la même pour R et $\mathrm{AMP}(\mathbf{O}^R)$. Supposons maintenant que, jusque-là, chaque partie de chaque objet a été placée dans R comme dans $\mathrm{AMP}(\mathbf{O}^R)$, et considérons la prochaine partie p d'un certain objet O_j.

Si p était, dans S, rangée dans B_i, alors elle est exclue de R comme de $\mathrm{AMP}(\mathbf{O}^R)$. Dans l'alternative, elle était placée dans la boîte la plus vide, qui est la même dans R et de $\mathrm{AMP}(\mathbf{O}^R)$, les objets ayant été jusque-là placés de manière identique, par hypothèse. Aussi p sera placée dans la même boîte pour les deux solutions (à l'exception d'échanges éventuels de bouts identiques, lorsqu'un choix existe quant à la boîte la plus vide). ⊗

Ces deux lemmes permettent de mieux appréhender le passage d'une solution à une solution ayant moins de boîtes (approche déjà utilisée pour le théorème 8.2). Avec en plus l'hypothèse de l'optimalité du placement de $\mathbf{O}_1$, la garantie devient :

Théorème 8.3. *Soit I une instance de $B|size_j|H_{\max}$, et $\mathbf{O}_1, \mathbf{O}_2$ une partition de son ensemble d'objets $\mathbf{O}$. Soit S une solution obtenue par un rangement optimal de $\mathbf{O}_1$ complété avec les objets de $\mathbf{O}_2$ selon la règle MP :*

$$H^S_{\max} \leq H^*_{\max} + h_{\max}(\mathbf{O}_2) \qquad \text{avec} \qquad h_{\max}(\mathbf{O}_2) = \max\{h_j, O_j \in \mathbf{O}_2\}$$

et cette borne est atteinte, même pour des objets de hauteur unitaire.

PREUVE — La preuve est une récurrence sur le nombre de boîtes : je montre en fait que, s'il n'existe pas d'instance avec m boîtes telle que $H^S_{\max} > H^*_{\max} + h_{\max}(\mathbf{O}_2)$, pour une certaine partition $\mathbf{O} = \mathbf{O}_1 \cup \mathbf{O}_2$, alors il n'existe pas non plus d'instance avec $m+1$ boîtes et cette propriété.

Pour débuter la récurrence, remarquons qu'avec une seule boîte, toute solution est optimale, et donc aucune solution, quelle que soit la partition adoptée, ne vérifie : $H^S_{\max} > H^*_{\max} + h_{\max}(\mathbf{O}_2)$.

Supposons maintenant l'existence d'une solution S avec $m+1$ boîtes, construite selon le schéma spécifié et telle que $H^S_{\max} > H^*_{\max} + h_{\max}(\mathbf{O}_2)$ pour une certaine partition. Montrons que cela implique l'existence d'une solution similaire sur m boîtes, ce qui contredirait l'hypothèse de récurrence.

Pour cela, soit B_i une boîte de S telle que $H_i < H^S_{\max}$. Elle existe, sinon S serait optimale. Soit R la réduction de S par B_i. D'après le lemme 8.2 $R(\mathbf{O}_1)$ est un rangement optimal de $\mathbf{O}^R_1$. De plus, d'après le lemme 8.3, R est construite comme S : un rangement optimale de $\mathbf{O}^R_1$, complété par $\mathbf{O}^R_2$ selon la règle MP. De plus :

$$\begin{array}{rcll} H^R_{\max} & = & H^S_{\max} & (B_k < H^S_{\max}) \\ & > & H^*_{\max}(I) + h_{\max}(\mathbf{O}_2) & (S \text{ est un contre-exemple}) \\ & \geq & H^*_{\max}(I^R) + h_{\max}(\mathbf{O}^R_2) & (H^*_{\max}(I) = H^*_{\max}(I^R) \text{ [lemme 8.3(c)]} ; \\ & & & \quad h_{\max}(\mathbf{O}_2) \geq h_{\max}(\mathbf{O}^R_2)). \end{array}$$

Finalement, si S est un contre-exemple pour I, avec $m+1$ boîtes, alors R est un contre-exemple pour I^R, avec m boîtes. Ceci prouve le théorème.

Pour montrer que la borne annoncée est serrée, même dans le cas d'objets de hauteur unitaire, considérons l'instance suivante : m boîtes et $n = 2m-1$ objets de hauteur unitaire et de largeur $size_1 = 2$ et $size_j = 1, \forall j \geq 2$. Comme l'illustre la figure 8.4, une solution optimale a pour valeur 2, tandis qu'une solution construite selon le schéma prescrit par le théorème 8.3 peut avoir une valeur de $3 = 2 + h_{\max}(\mathbf{O}_2)$, avec $\mathbf{O}_2 = \{O_1\}$ et $\mathbf{O}_1 = \mathbf{O} \backslash \mathbf{O}_2$. ⊗

Ce dernier théorème est indépendant des autres, dans le sens où il ne repose sur aucun des résultas précédents (à l'exception, bien sûr, des deux lemmes établis spécialement pour lui). De plus, il implique lui aussi que la règle MP garantit de ne pas dépasser l'optimum par plus de $h_{\max}$ (cas où $\mathbf{O}_1 = \emptyset$; avec la généralisation, le terme correctif $(m-1)/m$ a été perdu), et donc que c'est une 2-approximation. De plus, nous avons la conséquence suivante :

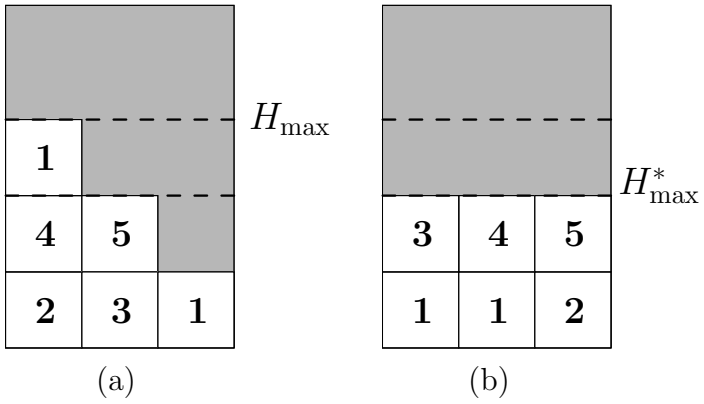


Figure 8.4 – Un exemple atteignant la borne du théorème 8.3.
$\mathbf{O}_1 = \{O_2, O_3, O_4, O_5\}$ et $\mathbf{O}_2 = \{O_1\}$
(a) une solution où $\mathbf{O}_1$ est rangé optimalement, mais $H_{\max} = H^*_{\max} + h_{\max}(\mathbf{O}_2)$;
(b) une solution optimale.

Corollaire 8.3.1. *Soit S une solution obtenue en rangeant optimalement les k objets les plus hauts, puis en complétant avec les autres objets selon la règle MP :*

$$H^S_{\max} \leq \left(1 + \frac{1}{1 + \left\lfloor \frac{k}{m} \right\rfloor}\right) H^*_{\max}.$$

PREUVE — Soit $\mathbf{O}_1$ l'ensemble des k plus hauts objets et soit $\mathbf{O}_2$ l'ensemble des autres objets. Il y a au moins $k+1$ objets de hauteur $h_{\max}(\mathbf{O}_2)$ ou plus. Aussi, au moins une boîte contient au moins $1 + \lfloor k/m \rfloor$ tels objets : $H^*_{\max} \geq (1 + \lfloor k/m \rfloor) h_{\max}(\mathbf{O}_2)$, c'est-à-dire :

$$h_{\max}(\mathbf{O}_2) \leq \frac{1}{1 + \left\lfloor \frac{k}{m} \right\rfloor} H^*_{\max}.$$

D'après le théorème 8.3, S vérifie $H^S_{\max} \leq H^*_{\max} + h_{\max}(\mathbf{O}_2)$ et donc :

$$H^S_{\max} \leq (1 + \frac{1}{1 + \left\lfloor \frac{k}{m} \right\rfloor}) H^*_{\max}.$$

⊗

Ce dernier résultat est très proche de celui de Graham pour l'ordonnancement par liste [64]. Une comparaison plus détaillée est faite à la section 8.2.4. Auparavant, je vais utiliser ces derniers résultats pour déduire de meilleures performances lorsque les objets sont triés par hauteur décroissante.

8.2.3 Meilleur placement décroissant

Jusqu'à présent, aucune hypothèse d'un tri particulier des objets n'a été faite. Je considère maintenant le cas où les objets sont ordonnés par hauteur décroissante.

Un tel algorithme, implémentable en temps $\mathcal{O}(n \ln n + nm)$, est appelé AMPD (Algorithme de Meilleur Placement Décroissant).

Par rapport à un AMP quelconque, AMPD présente l'inconvénient de ne plus être un algorithme en-ligne. Il possède toutefois un atout majeur : il est plus performant. En premier lieu, il est optimal pour certains types d'instances :

Lemme 8.4. *AMPD est optimal si au moins l'une des conditions suivantes est vérifiée :*

(a) $\forall O_j \in \mathbf{O} : h_j > H^*_{\max}/3$ *;*
(b) au plus deux objets sont rangés dans la boîte la plus haute de la solution obtenue ;
(c) il existe une solution optimale avec au plus deux objets par boîte.

PREUVE — Il est clair que l'hypothèse (a) implique (c), aussi suffit-il de prouver le résultat pour (c) et (b).

Pour l'affirmation (b) : s'il n'y a qu'un objet dans la boîte la plus haute, la solution est nécessairement optimale, indépendamment de toute manière de construire la solution. Supposons donc qu'il y a deux objets dans la boîte la plus haute, B_i. Soit O_{j_1} le premier des deux objets rangés, et O_{j_2} le second. Puisque deux objets sont rangés dans la même boîte, d'après la règle MP : il ne reste aucune boîte vide et O_{j_1} est le plus petit objet qui puisse être rangé avec O_{j_2}. En raison de l'ordre sur les objets, O_{j_2} est le plus petit objet rangé (au moment où il est rangé).

Aussi, même s'il peut y avoir des objets mis entre O_{j_1} et O_{j_2}, ces objets sont nécessairement rangés par deux (sinon O_{j_1} ne serait pas choisi par la règle MP) et il sont au moins aussi gros que O_{j_2} (donc un échange ne peut pas améliorer la solution). Ainsi, une solution où O_{j_1} et O_{j_2} ne sont pas ensembles ne peut être que moins bonne que S ; cette dernière est donc optimale.

Pour l'affirmation (c), la technique est similaire à celle de [64], pour un résultat similaire. Considérons un rangement optimal, avec au plus deux objets par boîte. Nous allons le transformer en un rangement MPD sans en changer la valeur.

Étape 1 : tant qu'il existe deux boîtes B_{i_1} et B_{i_2} telles que B_{i_1} contient deux objets O_{j_1} et O_{j_2} et B_{i_2} ne contient qu'un seul objet O_{j_3} tel que : $h_{j_1} > h_{j_3}$ et $O_{j_2} \neq O_{j_3}$, ranger O_{j_2} et O_{j_3} dans B_{i_1} et O_{j_1} dans B_{i_2}. Cette transformation est toujours valide et ne détériore pas la solution (voir figure 8.5(a)).

Étape 2 : tant qu'il existe deux boîtes B_{i_1} et B_{i_2} telles que B_{i_1} contient deux objets O_{j_1} et O_{j_2} et B_{i_2} contient deux objets O_{j_3} et O_{j_4} tels que $h_{j_1} > h_{j_3}$ et $h_{j_2} > h_{j_4}$ et $O_{j_1} \neq O_{j_4}$ et $O_{j_2} \neq O_{j_3}$, ranger O_{j_1} et O_{j_4} dans B_{i_1} et ranger O_{j_2} et O_{j_3} dans B_{i_2}. Cette transformation est toujours valide et ne détériore pas la solution (voir figure 8.5(b)).

Étape 3 : trier les boîtes par ordre décroissant de la hauteur de leur plus grand objet (voir figure 8.5(c)).

Après ces trois étapes, la solution optimale initiale n'a pas changé de valeur, mais elle possède la structure d'une solution MPD. De plus, tout autre solution MPD n'en

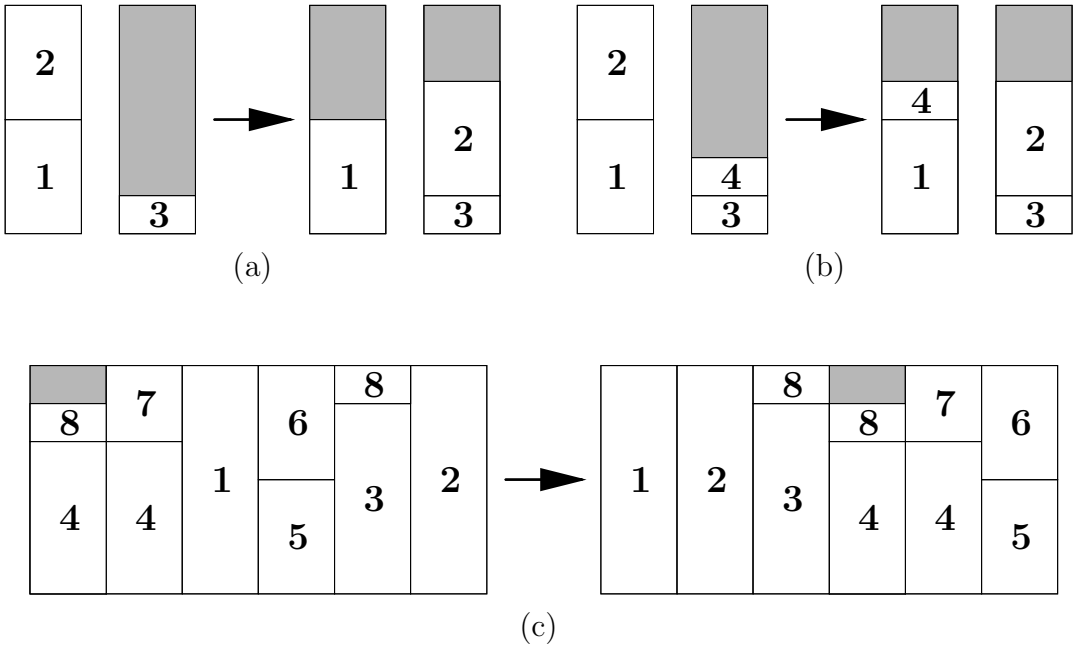


Figure 8.5 – Transformation d'une solution optimale en solution MPD.
(a) étape 1 ; (b) étape 2 ; (c) étape 3.

diffère que par des échanges de bouts d'objets de même hauteur, et a donc même valeur. Ceci prouve l'affirmation (c) (et donc aussi la (a)). ⊗

Il faut faire attention : l'affirmation (c) ne peut pas être réduite à « il existe une solution optimale avec au plus deux objets dans la boîte la plus haute ». (p.ex. l'instance : $m = 2$, $n = 6$, $\forall j : size_j = 1$ et $h_1 = 5, h_2 = 3, h_3 = h_4 = h_5 = h_6 = 2$ est un contre-exemple). De même l'hypothèse (b) n'est pas impliquée par $\sum_{O_j \in \mathbf{O}} size_j \leq 2m$ (p.ex. l'instance : $m = 2$, $n = 4$, $\forall j : size_j = 1$ et $h_1 = 5, h_2 = h_3 = h_4 = 2$ est un contre-exemple). En fait, dans ce dernier cas AMPD n'est pas toujours optimal (p.ex. considérer l'instance : $m = 4$, $n = 8$, $\forall j : size_j = 1$ et $h_1 = h_2 = 8, h_3 = 5, h_4 = 3, h_5 = h_6 = h_7 = h_8 = 2$).

En conjuguant ce dernier résultat avec les résultats de la section précédente, on obtient la garantie pour AMPD :

Théorème 8.4. *AMPD est une* 4/3*-approximation.*

PREUVE — Soit S une solution construite par AMPD. Soit $\mathbf{O}_1 = \{O_j \in \mathbf{O}, h_j > H^*_{\max}/3\}$ et $\mathbf{O}_2 = \{O_j \in \mathbf{O}, h_j \leq H^*_{\max}/3\}$. Par définition, AMPD range d'abord les objets de $\mathbf{O}_1$. D'après le lemme 8.4(a), ces objets sont rangés optimalement. Aussi, d'après le théorème 8.3, on a la garantie que :

$$H^S_{\max} \leq H^*_{\max} + h_{\max}(\mathbf{O}_2) \leq H^*_{\max} + \frac{H^*_{\max}}{3} \leq \frac{4}{3} H^*_{\max}.$$

⊗

L'algorithme LPT, proposé par Graham pour le problème $B|size_j = 1|H_{\max}$ est un cas particulier de AMPD. Des exemples ont été donnés, pour lesquels LPT n'atteint qu'une performance de $4/3 - 1/3m$ [64, 35]. Aussi AMPD est au mieux une $(4/3 - 1/3m)$-approximation. Ce petit écart reste ouvert, et il serait intéressant de le combler :

Question 8.1 : L'algorithme de meilleur placement décroissant (AMPD) est-il plutôt une 4/3- ou une $(4/3 - 1/3m)$-approximation ?

Ceci nous amène à la comparaison de la règle MP avec les algorithmes de listes, dont LPT est un cas particulier.

8.2.4 Ordonnancements par liste et règle MP

Pour conclure cette section sur les algorithmes de meilleur placement, je présente une comparaison des résultats existants sur les algorithmes de liste pour le problème d'Ordonnancement sur machines parallèles et ceux présentés dans cette section.

Les résultats des théorèmes 8.1, 8.3, et 8.4 généralisent ceux bien connus de [64] et [35] pour des objets de largeur unitaire (le cas $B|size_j = 1|H_{\max}$) :

(a) AMP est une $(2 - \frac{1}{m})$-approximation ;

(b) AMPD est une $(\frac{4}{3} - \frac{1}{3m})$-approximation ;

(c) ranger les k plus hauts objets optimalement, puis compléter selon la règle MP est une $(1 + \frac{1-\frac{1}{m}}{1+\lfloor \frac{k}{m} \rfloor})$-approximation ;

(d) AMPD est optimal si $r = 1, 2$ et est une $(1 + \frac{1}{r} - \frac{1}{rm})$-approximation sinon (pour r : nombre d'objets rangés dans la boîte la plus haute).

Les assertions (a), (b) et (c) ont été prouvées par Graham [64], tandis que la partie (d) est due à Coffman et Sethi [35]. L'affirmation (c) implique à la fois (a) et (b), mais ceux-ci restent les résultats les plus connus : (a) est la borne générale pour l'ordonnancement par liste, tandis que (b) est la performance garantie pour LPT.

Pour le problème généralisé $Bm|size_j|H_{\max}$, où les objets sont de largeur quelconque, le (a) reste valide (corollaire 8.1.2).

Le (b) est affaibli : pour des objets de largeur 1, la garantie prouvée par Graham (et atteinte) est de $\frac{4}{3} - \frac{1}{3m}$, tandis que je n'obtiens qu'une garantie de $\frac{4}{3}$ (théorème 8.4). La question reste ouverte quant à cet écart de $\frac{1}{3m}$ (voir question 8.1). Précisons toutefois que pour $m = 2$, on vérifie facilement que AMPD atteint la performance $\frac{4}{3} - \frac{1}{3m}$ et que pour m très grand, les deux garanties sont asymptotiquement équivalentes.

Par rapport à (c), le théorème 8.3 généralise à la fois le problème (objets de largeur quelconque) et les hypothèses (l'ensemble rangé optimalement n'est pas nécessairement formé des plus gros objets). Par cette généralisation, un petit terme

correctif a été perdu, lorsqu'on revient au même problème : la garantie prouvée n'est plus que de $1 + \frac{1}{1+\lfloor k/m \rfloor}$, contre $1 + \frac{1-1/m}{1+\lfloor k/m \rfloor}$ (corolaire 8.3.1).

L'assertion (d) est une extension possible de ces travaux ; la première partie, AMPD optimal pour $r = 1, 2$ est prouvée (lemme 8.4).

Pour finir cette section sur la règle MP, signalons une extension intéressante (notamment en vue du chapitre 9) des théorèmes 8.1, 8.2 et 8.3. Les cas sans pré-affectation, avec pré-affectation quelconque et avec pré-affectation optimale d'un sous-ensemble d'objets ont été étudiés. Qu'en est-il pour une affectation avec des garanties ? ou pour poser formellement la question :

Question 8.2 : Pour une partition $\mathbf{O}_1, \mathbf{O}_2$ de l'ensemble des objets, soit S la solution construite à partir d'une solution $S(\mathbf{O}_1)$ complétée par $\mathbf{O}_2$ selon MP. Si $S(\mathbf{O}_1)$ est une α-approximation pour les seuls objets de $\mathbf{O}_1$, a-t-on une garantie du type : $H^S_{\max} \leq \alpha H^*_{\max} + h_{\max}(\mathbf{O}_2)$?

8.3 Placement dans les boîtes suivantes

Au chapitre 3, nous avons vu un algorithme (l'algorithme 3.1) qui résout optimalement $B|size_j|H_{\max}$ si les objets sont de hauteur unitaire. Que devient cet algorithme pour des hauteurs quelconques ? Cette section répond à cette question.

Le principe de l'algorithme est très simple : un objet est placé immédiatement, dans les boîtes qui suivent celles où l'objet précédent a été rangé. Cette règle de *placement dans les boîtes suivantes* (PBS) donne ainsi, sans aucune difficulté, l'algorithme 8.2 (APBS), qui construit une solution en temps $\mathcal{O}(nm)$.

Algorithme 8.2. APBS

- **entrées :** ensemble d'objets $\mathbf{O}$, nombre de boîtes m -
$i = 1$ — *les indices sont modulo* m *:* $B_i = B_{(i \bmod m)+1}$
pour $O_j \in \mathbf{O}$ faire
  ranger O_j dans $B_i, B_{i+1}, \ldots, B_{i+size_j-1}$
  $i = i + size_j$

Tel quel, cet algorithme peut facilement être dupé : il suffit d'ordonner les objets de manière à ce que les plus grands soient systématiquement mis dans la même boîte pour obtenir une solution exécrable. En fait, on peut obtenir le pire ratio possible, avec une solution m fois trop grande[2] ! Il suffit pour cela de considérer l'instance suivante : m boîtes et $n = 2p$ objets tels que $h_j = K$ et $size_j = 1$ si j est impair, et $h_j = 1$ et $size_j = m - 1$ si j est pair. La solution construite par APBS, pour les

[2]Ce facteur m est bien le pire possible, comme cela a été expliqué au chapitre 3, section 3.2.

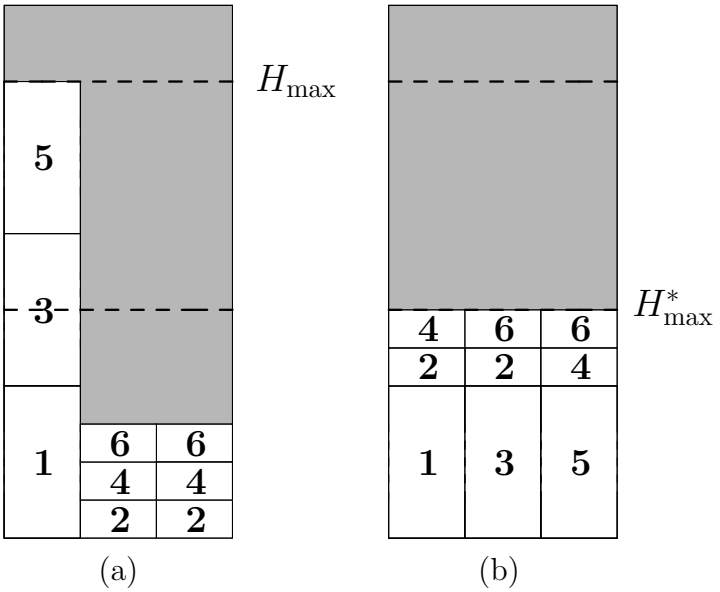


Figure 8.6 – Un exemple d'instance pour lequel ABPS est exécrable.
(a) ABPS selon l'ordre naturel ; (b) une solution optimale.

objets pris dans l'ordre naturel, a pour valeur pK, tandis que la valeur optimale est égale au volume par boîte : $p(K+m-1)/m$, si $p \bmod m = 0$ (voir figure 8.6). Le rapport entre les deux est alors de $mK/(K+m-1)$, ce qui tend vers m lorsque K tend vers l'infini.

Il suffit toutefois d'utiliser un ordre spécial sur les objets pour que l'algorithme devienne compétitif. Assez naturellement, il faut trier les objets par hauteurs décroissantes. Cela donne l'algorithme APBSD (Algorithme de Placement dans les Boîtes Suivantes, Décroissant), qui est une 2-approximation.

Théorème 8.5. *APBSD est une 2-approximation.*

PREUVE — Soit $H_{\max}$ la hauteur de la plus grande boîte (disons B_1), à la fin de l'algorithme, et appelons $O_1, O_2, \ldots, O_k$ les objets que celle-ci contient. Ceux-ci étant rangés par ordre décroissant, la règle PBS implique que, lorsque O_2 a été mis dans B_1, des objets au moins aussi grands avaient été rangés dans toutes les autres boîtes. Ceci est aussi vrai pour $O_3, O_4, \ldots, O_k$. Toutes les boîtes contiennent donc, au minimum, la quantité : $K = \sum_{j=2}^{k} h_k$.

Soit V le volume total des objets ; $V \geq mK + h_1$. Comme la valeur optimale $H^*_{\max}$ vérifie $H^*_{\max} \geq \max(h_{\max}, V/m)$ (proposition 7.1) on déduit pour ces deux valeurs, respectivement :

$$\frac{H_{\max}}{H^*_{\max}} \leq \begin{cases} \dfrac{(K+h_1)}{h_{\max}} & \leq \; 1 + \dfrac{K}{h_{\max}} \\ \dfrac{m(K+h_1)}{mK+h_1} & \leq \; 1 + \dfrac{(m-1)h_1}{mK+h_1} \end{cases}$$

Deux possibilités sont ainsi à envisager. Si $K \leq h_1$, alors il est *a fortiori* plus petit que $h_{\max}$. Dans ce cas, $H_{\max}/H^*_{\max} \leq 1 + K/h_{\max}$ permet de conclure à un rapport inférieur à 2. Si $K \geq h_1$, on utilise l'autre borne, $H_{\max}/H^*_{\max} \leq 1+(m-1)h_1/(mK+h_1)$: comme $K \geq h_1$, $(m-1)h_1/(mK+h_1)$ est plus petit que $(m-1)/(m+1)$, ce qui permet de conclure à un rapport inférieur à 2.

Dans tous les cas le rapport à la solution optimale est donc d'au plus 2. ⊗

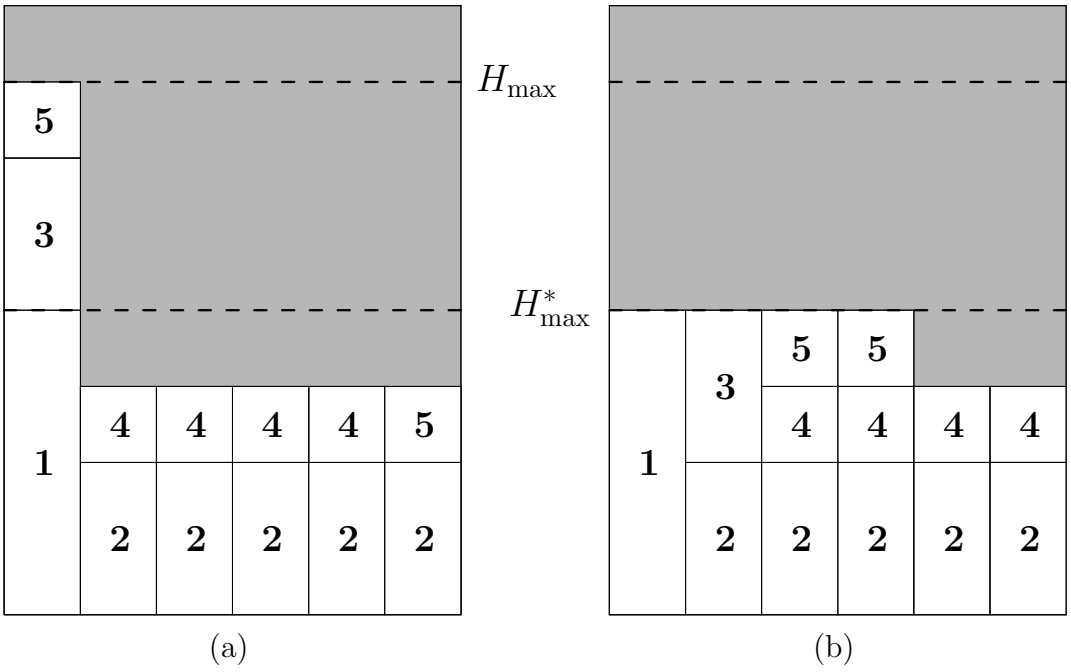


Figure 8.7 – Un exemple d'instance pour lequel ABPSD est mauvais.
(a) ABPS selon l'ordre naturel ; (b) une solution optimale.

Ce rapport de 2 est ce que l'on peut obtenir de mieux pour APBS. Pour illustrer cela, considérons l'instance suivante. Pour deux entiers p et K donnés (avec K puissance de 2 supérieure à 2^{p-1}), il y a $m \geq p+2$ boîtes et $n = 2p-1$ objets. $h_1 = K$ et $size_1 = 1$; pour les autres objets : $h_{2j} = K/2^j$, $size_{2j} = m - j$ et $h_{2j+1} = K/2^j$, $size_{2j+1} = j$ (voir figure 8.7).

Par construction, $size_{2j} + size_{2j+1} = m$: la boîte qui reçoit O_1 contiendra donc aussi tous les autres objets d'indices impairs ; de plus cette boîte sera la plus haute. La valeur $H_{\max}$ de la solution est donc

$$H_{\max} = \sum_{j=0}^{p-1} \frac{K}{2^j} = K\left(2 - \frac{1}{2^{p-1}}\right).$$

Par ailleurs, la valeur optimale $H^*_{\max}$, obtenue par exemple par AMPD, est de K. Donc

$$\frac{H_{\max}}{H^*_{\max}} = \frac{K(2 - 2^{-(p-1)})}{K} = 2 - 2^{-(p-1)},$$

ce qui tend vers 2 lorsque p (donc le nombre d'objets) tend vers l'infini.

8.4 Placement optimiste

Dans cette section, j'aborde une troisième règle de placement des objets, après celles de meilleur placement et de placement dans les boîtes suivantes des sections précédentes.

Tout commence par le constat que la borne inférieure $\lceil V/m \rceil$, où V est le volume total des objets, est très souvent atteinte en pratique ; elle est aussi rarement une mauvaise approximation. L'idée est donc de lui faire confiance et de s'efforcer de ranger les objets sans dépasser cette valeur, tant que cela est possible ; cette règle, qui revient à espérer que tout se passera au mieux pour atteindre sans dépasser la valeur critique, est appelée *placement optimiste* (PO).

Pour réaliser cela, on calcule la valeur de référence $H_{ref} = \left\lceil \sum_{j=1}^{n} size_j h_j / m \right\rceil$ puis, boîte après boîte, on range tous les objets que l'on peut, dans l'ordre où ils se présentent, tant que la valeur H_{ref} n'est pas dépassée. Les quelques objets restant à la fin sont rangés selon la règle de meilleur placement, *i.e.* dans les boîtes les moins pleines.

Algorithme 8.3. APO

- **entrées :** ensemble d'objets $\mathbf{O}$, nombre de boîtes m -

1. calculer $H_{ref} = \left\lceil \sum_{j=1}^{n} size_j h_j / m \right\rceil$
2. pour $i = 1$ à m faire
   pour $O_j \in \mathbf{O}$ faire
   si O_j n'est pas complètement rangé et
   $H_i + h_j \leq H_{ref}$
   alors ranger O_j dans B_i
3. placer chaque bout restant dans une boîte différente

L'algorithme 8.3 (APO) construit une telle solution. Il est doublement optimiste : en plus d'avoir confiance en la valeur H_{ref}, il espère que tout se passera bien à l'étape 3 et qu'aucun bout d'objet ne sera mis dans une boîte contenant déjà cet objet... ce qui n'est pas toujours possible ! Si tout se passe bien, on obtient toutefois :

Propriété 8.4. *Une solution construite par APO vérifie :*

$$H_{\max} \leq H^*_{\max} + h_{\max} - 1$$

et cette borne est serrée. Si les objets considérés à l'étape 3 sont déjà partiellement rangés, elle peut ne pas être réalisable.

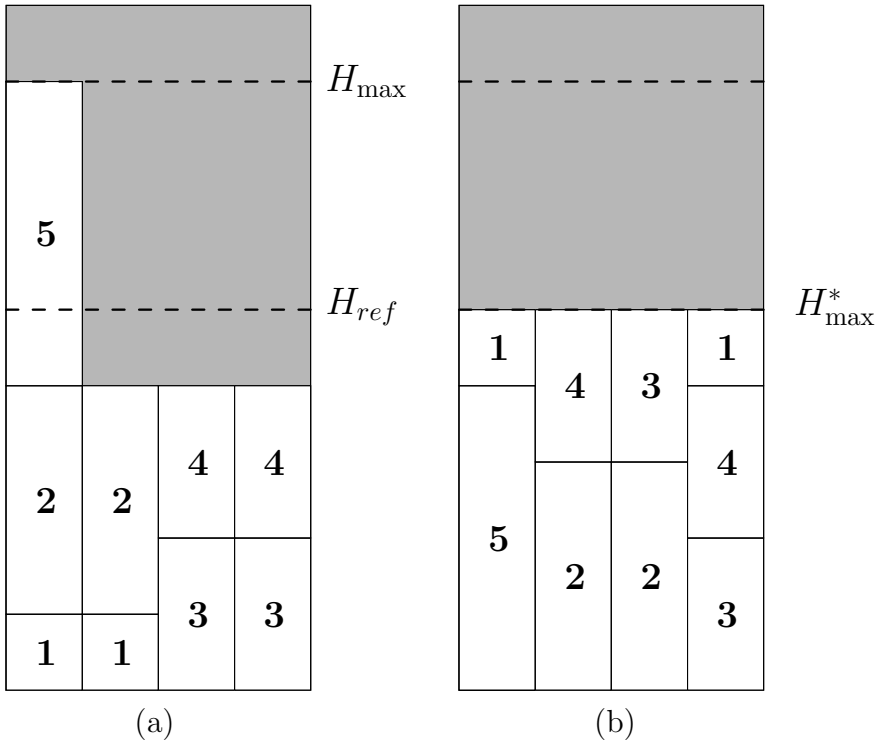


Figure 8.8 – Un exemple d'instance pour lequel APO et APOM sont mauvais.
(a) APO (ou APOM) selon l'ordre naturel ; (b) une solution optimale.

PREUVE — Soit R l'ensemble des objets non rangés à la fin de l'étape 2. Le volume total de R tient dans les « vides » $(H_{ref} - H_i)$ laissés en haut des boîtes, bien qu'aucun n'objet ne tienne dans ces vides : il y a donc moins de bouts d'objets que de boîtes avec un vide. Il est donc possible de mettre chacun dans une boîte différente. Comme toute boîte avec un vide vérifie $H_i < H_{ref}$ avant cet ajout, après cet ajout : $\forall i$: $H_i < H_{ref} + h_{\max}$, et donc $H_{\max} < H^*_{\max} + h_{\max}$.

La figure 8.8 donne un exemple où cette borne est atteinte. ⊗

Une version alternative à l'implémentation de l'algorithme 8.3 est donnée par l'algorithme 8.4 (APO Modifié, APOM). Elle permet d'assurer que la solution construite est réalisable. Cependant, elle ne correspond pas exactement aux hypothèses précédemment utilisées (p.ex. deux objets peuvent être placés dans la même boîte au delà de la hauteur H_{ref}) et le résultat précédent ne lui est donc pas applicable : des objets très larges arrivant en fin de procédure peuvent créer des problèmes.

Il semble cependant que, si les objets sont triés par largeurs décroissantes, l'algorithme obtenu (APOMD) a les mêmes garanties.

Conjecture 8.2. *Une solution constuite par APOM (algorithme 8.4) vérifie :*

$$H_{\max} \leq H^*_{\max} + h_{\max},$$

si les objets sont initialement rangés par largeurs décroissantes.

Algorithme 8.4. APOM

- **entrées :** ensemble d'objets **O**, nombre de boîtes m -

1. calculer $H_{ref} = \left\lceil \sum_{j=1}^{n} size_j h_j / m \right\rceil$
2. pour $O_j \in \mathbf{O}$ faire
   pour $k = 1$ à $size_j$ faire
   soit B_{i_1} la première boîte ne contenant pas O_j
   et telle que $H_i + h_j \leq H_{ref}$
   soit B_{i_2} la plus petite boîte ne contenant pas O_j
   si B_{i_1} existe, y ranger le $k^{ème}$ bout de O_j
   sinon le ranger dans B_{i_2}

En tout cas, on ne peut pas espérer mieux sans hypothèse supplémentaire : l'exemple de la figure 8.8 convient pour illustrer cela. Si les objets sont ordonnés par hauteurs décroissantes (pour des sous-ensembles de même largeur), on peut s'attendre à des gains substantiels. Deux autres améliorations possibles sont :

- APOM variable (APOMV) où H_{ref} est variable : chaque fois qu'un objet dépasse la valeur H_{ref} (il est rangé dans « B_{i_2} »), celle-ci est augmentée à la nouvelle valeur minimum imposée par cet objet (pour O_j mis dans B_{i_2}, on pose $H_{ref} = H_{i_2}$ après insertion de O_j).
- APOM à essais multiples (APOMX) où les $h_{\max}$ valeurs possibles pour $H^*_{\max}$ sont essayées[3].

Aucun résultat théorique n'est pour l'instant disponible pour ces variantes, mais une comparaison empirique est dressée au chapitre 11. APOMX apparaît toutefois très proche du *Multifit*, proposé pour $P||C_{\max}$, qui a de très bonnes garanties et un excellent comportement en pratique [36]. À noter que la version énoncée (essayer les $h_{\max}$ valeurs possibles) est un algorithme seulement pseudo-polynomial, même pour m fixé. Une recherche dichotomique pourrait résoudre ce problème, mais elle n'a pas de sens tant qu'un critère de monotonie sur les solutions produites n'a pas été dégagé. D'un point de vue expérimental, cela n'a cependant qu'une faible incidence.

8.5 Diviser pour régner

Je m'intéresse maintenant à une approche différente pour résoudre $B|size_j|H_{\max}$, tirée du principe de « diviser pour régner ».

8.5.1 L'algorithme et ses performances

La méthode consiste à séparer l'ensemble des objets en deux, résoudre récursivement pour chaque partie, puis rassembler en une solution unique. Ainsi l'algo-

[3]Les bornes L_V et U_V du chapitre 7, section 7.1, garantissent $V/m \leq H^*_{\max} \leq V/m + h_{\max}$, pour V le volume total.

rithme commence par créer n solutions partielles (une par objet), puis les fusionne itérativement par deux, la $k^{ème}$ plus haute boîte de l'une étant mise avec la $k^{ème}$ plus basse boîte de l'autre (voir figure 8.9). Un tel algorithme est appelé ADR (Algorithme de Diviser pour Régner).

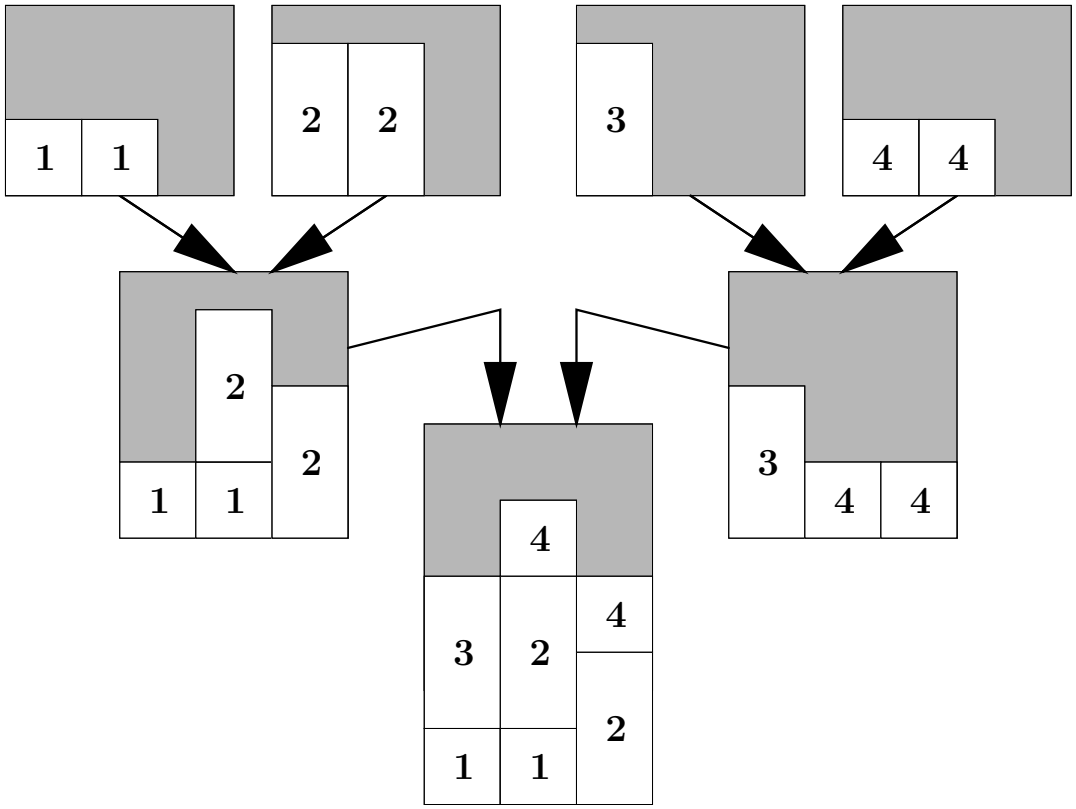


Figure 8.9 – Un exemple d'exécution d'ADR.

Pour une implémentation efficace de la fusion de deux solutions partielles, les boîtes sont systématiquement triées par hauteurs décroissantes, la boîte i d'une solution devant alors être ajoutée à la boîte $m-i$ de l'autre solution. Sur cette idée, l'algorithme 8.5 implémente ADR en temps $\mathcal{O}(nm \log m)$.

Il est possible d'éviter de trier systématiquement toutes les boîtes, ce qui laisse la possibilité d'une meilleure implémentation. Quoi qu'il en soit, ADR, tout comme AMP (section 8.2) est un algorithme polynomial pour $Bm|size_j|H_{\max}$ et pseudo-polynomial si m n'est pas fixé ($B|size_j|H_{\max}$).

Les performances de ADR sont assez similaires à celle de AMP, et elles sont basées sur des propriétés proches :

Propriété 8.5. *Soit S une solution partielle obtenue en fusionnant deux solutions T et U, comme cela est effectué par ADR.*

$$\Delta^S \leq \max(\Delta^U, \Delta^T).$$

Algorithme 8.5. ADR

```
- entrées : ensemble d'objets O, nombre de boîtes  m -
1. pour O_j ∈ O faire                    — n solutions partielles initiales
       ranger O_j dans les size_j premières boîtes de S_j.
2. k = 1
   tant que (k < n) faire                — fusion des solutions partielles
     pour j = 1 à n par 2k faire
       si (j + k ≤ n) alors
         pour i = 1 à m faire            — fusion de S_j et S_{j+k}
           ajouter B_{m-i} de S_{j+k} à B_i de S_j
         trier les boîtes de S_j par hauteurs décroissantes
       détruire S_{j+k}.
     k = k * 2
3. renvoyer S_1
```

PREUVE — Soit B_{i_1} (respectivement B_{i_2}) la boîte la plus haute (basse) de S :

$$\Delta^S = |H^S_{i_1} - H^S_{i_2}| = |H^U_{i_1} - H^U_{i_2} + H^T_{i_1} - H^T_{i_2}|.$$

Les boîtes de U et T étant triées dans des sens opposés, $H^U_{i_1} - H^U_{i_2}$ est positif si, et seulement si, $H^T_{i_1} - H^T_{i_2}$ est négatif. Cela implique $\Delta \leq \max(\Delta^U, \Delta^T)$. ⊗

Cette propriété implique qu'une solution construite par ADR a une différence de hauteur d'au plus $h_{\max}$. Tout comme pour AMP, il s'ensuit donc :

Théorème 8.6. *Pour une solution calculée par ADR :*

$$H_{\max} \leq H^*_{\max} + \frac{m-1}{m}\Delta \qquad \text{avec} \qquad \Delta \leq h_{\max}$$

et cette borne est atteinte.

PREUVE — Il ne reste qu'à montrer que la borne est atteinte. On peut, par exemple, considérer la même instance que pour AMP : m boîtes et m^2+1 objets de largeur 1 dont les hauteurs sont $h_1 = h_2 = \ldots = h_{m^2} = 1$ et $h_{m^2+1} = m$. La solution optimale a pour valeur $m+1$. Si m^2 est une puissance de 2 alors la solution produite par ADR a pour valeur $H_{\max} = 2m$, soit $H_{\max} = H^*_{\max} + \frac{m-1}{m}h_{\max}$ (voir figure 8.10). ⊗

Les conséquences pour ADR du théorème précédent sont naturellement les mêmes que celles du théorème 8.1 pour AMP :

Corollaire 8.6.1. *ADR est asymptotiquement optimal pour les instances uniformément bornées de $Bm|size_j|H_{\max}$ (*i.e. *il existe des bornes uniformes $h_{\max} \geq h_{\min} > 0$ telles que tout objet O_j vérifie : $h_{\max} \geq h_j \geq h_{\min}$).*

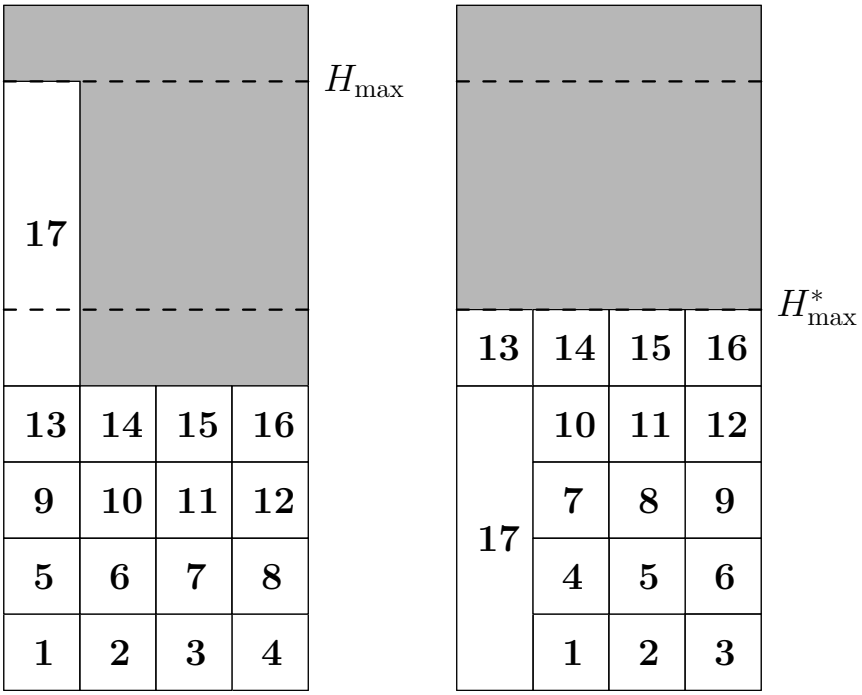


Figure 8.10 – Un exemple d'instance pour lequel ADR est mauvais.
(a) ADR selon l'ordre naturel ; (b) une solution optimale.

Corollaire 8.6.2. *Une solution construite par ADR vérifie :*

$$H_{\max} \leq (2 - \frac{1}{m})H^*_{\max}.$$

Corollaire 8.6.3. *ADR est optimal pour* $B|size_j, h_j = 1|H_{\max}$.

8.5.2 ADR et la « méthode par différences »

L'algorithme ADR ressemble énormément à un algorithme proposé par Karmarkar et Karp pour le problème d'ordonnancement $P||C_{\max}$: la « méthode par différences »[4] [77], qui donne d'excellents résultats en pratique.

Plusieurs implémentations en ont été proposées, mais toutes se basent sur la fusion de solutions partielles, de manière similaire à celle proposée pour ADR. Les différences résident dans la sélection des solutions à fusionner.

Pour deux machines, l'alternative consistant à fusionner les deux solutions les plus grandes donne une 7/6-approximation [57]. Cette performance est aussi atteinte, pour deux boîtes, par ADR. L'extension à plus de deux machines de ce

[4]Connue dans la littérature sous le terme de *Differencing Method*.

choix, proposée dans [106], consiste à fusionner les deux solutions partielles ayant les plus grandes différences de hauteurs. Les auteurs prouvent alors une performance de $4/3 - 1/3m$. À la vue de ces résultats, la question qui se pose est bien entendu :

Question 8.3 : Existe-t-il une règle de fusion des solutions partielles telle que ADR soit une 4/3-approximation ?

8.6 Déplacements d'objets

Je présente dans cette section une approche très classique d'amélioration de solutions par déplacement d'objets. Cette méthode donne un algorithme de résolution mais elle permet surtout d'améliorer une solution déjà existante, par exemple calculée par l'un des algorithmes précédents. Toutefois, les garanties ne sont pas meilleures.

Le *déplacement d'objets* (DO) consiste à sélectionner un objet O_j qui se trouve dans une boîte B_{i_1} mais pas dans une boîte B_{i_2} plus petite, puis à enlever O_j de B_{i_1} pour le mettre dans B_{i_2}.

Si l'on effectue une telle opération tant qu'elle est possible, il est clair que la différence de hauteur entre deux boîtes, à la fin de l'algorithme, est au plus de $h_{\max}$. L'algorithme 8.6 (ADO), basé sur ce principe vérifie donc, en vertu du lemme 8.1 :

Théorème 8.7. *Une solution construite par ADO vérifie :*

$$H_{\max} \leq H^*_{\max} + \frac{m-1}{m} h_{\max}.$$

Corollaire 8.7.1. *ADO est une 2-approximation.*

Algorithme 8.6. ADO

- **entrées :** ensemble d'objets $\mathbf{O}$, nombre de boîtes m -
1. construire une solution réalisable S
2. tant qu'il existe i_1 et i_2 tels que $H_{i_1} > H_{i_2} + h_{\max}$ faire
   choisir $O_j \in B_{i_1} \backslash B_{i_2}$
   supprimer O_j de B_{i_1}
   ranger O_j dans B_{i_2}
3. renvoyer S

Si les performances de ADO sont évidentes, son temps d'exécution l'est beaucoup moins. En fait, il n'a même pas encore été justifié qu'il s'arrête toujours !

Heureusement il le fait... le mauvais point est qu'il peut ne terminer qu'en temps pseudo-polynomial, même pour m fixé :

Proposition 8.1. *L'algorithme 8.6 s'arrête après au plus* $m \sum_j h_j/2$ *itérations.*

PREUVE — À chaque itération, la mesure $f = \sum_{i<j} |H_i - H_j|$ décroît d'au moins 2. De plus, elle est bornée par 0 et $(m-1)\sum_j h_j$: il y a donc au plus $(m-1)\sum_j h_j/2$ itérations. ⊗

Une implémentation particulière permet cependant d'obtenir un algorithme polynomial pour m fixé. Il suffit de ne bouger les objets que de la gauche vers la droite. Ceci est réalisé par l'algorithme 8.7 (ADO Modifié, ADOM), en temps $\mathcal{O}(nm + m\ln(m))$.

Algorithme 8.7. ADOM

```
- entrées : ensemble d'objets O, nombre de boîtes m -
 1. construire une solution réalisable S
 2. trier les boîtes par hauteurs décroissantes
 3. i1 = 1, i2 = 2
 4. tant que H_{i1} − H_{i2} > h_max faire
       choisir O_j ∈ B_{i1} \ B_{i2}
       supprimer O_j de B_{i1}
       ranger O_j dans B_{i2}
 5. i2 = i2 + 1
    si i2 > m alors
       i1 = i1 + 1
       i2 = i1 + 1
    si i1 ≥ m aller à 6
    sinon aller à 4
 6. renvoyer S
```

Proposition 8.2. *L'algorithme 8.7 a un temps d'exécution* $\mathcal{O}(nm + m\ln(m))$.

PREUVE — Les étapes 1 et 2 prennent respectivement un temps $\mathcal{O}(nm)$ et $\mathcal{O}(m\ln(m))$. Pour les étapes 4-5, un objet n'a pas à être considéré plus d'une fois par boîte : elles s'exécutent donc en temps $\mathcal{O}(nm)$. Le temps global est donc de $\mathcal{O}(nm + m\ln(m))$. ⊗

En tant qu'algorithme de construction de solution à part entière, ADO (ou même ADOM) se satisfait de peu pour s'arrêter et ne permet pas d'obtenir de meilleures garanties que la plupart des algorithmes précédents. En fait, cette approche n'a essentiellement de sens qu'en vue d'améliorer des solutions déjà existantes. Dans ce

cas-là, les critères d'arrêt ne seront plus des différences de hauteurs plus petites que $h_{\max}$. Il faudra au moins, pour tout couple de boîtes (B_{i_1}, B_{i_2}) et tout objet O_j rangé dans B_{i_1} mais pas dans B_{i_2}, que $H_{i_1} - h_j \geq H_{i_2}$.

8.7 Quelques précisions et un récapitulatif

Les algorithmes précédents sont tous au minimum des 2-approximations. C'est en fait le moins qu'on doit leur demander : ce coefficient de 2 s'obtient simplement avec des arguments de volume, sans même essayer de construire une solution (voir chapitre 7, section 7.1). Un algorithme ne garantissant pas cela serait pathétique, à moins d'être extraordinairement performant sur toutes les instances à l'exception d'un minuscule sous-ensemble problématique.

Quelques améliorations dans les garanties peuvent sans doute être obtenues, par exemple en prenant en compte d'autres paramètres. Ainsi le lemme 8.1, base des garanties des algorithmes AMP, ADR et ADO, peut être amélioré :

Propriété 8.6. *Soit une solution S et une solution optimale S^* :*

$$H^S_{\max} \leq H^*_{\max} + \frac{(m-1)\Delta^S - \Delta^{S^*}}{m}.$$

PREUVE — La preuve est similaire à celle du lemme 8.1. En ce qui concerne S :

$$\sum_{j=1}^{n} h_j size_j \geq m H^S_{\max} - (m-1)\Delta^S.$$

Pour S^* il y a au plus $m-1$ boîtes de hauteur $H^*_{\max}$, et celle restante est de hauteur $H^*_{\max} - \Delta^{S^*}$:

$$(m-1)H^*_{\max} + (H^*_{\max} - \Delta^{S^*}) \geq \sum_{j=1}^{n} h_j size_j.$$

Il en découle : $mH^*_{\max} - \Delta^{S^*} \geq mH^S_{\max} - (m-1)\Delta^S$, c'est-à-dire :

$$H^*_{\max} + \frac{(m-1)\Delta^S - \Delta^{S^*}}{m} \geq H^S_{\max}.$$

⊗

Cette propriété pourrait se révéler très utile pour certains types d'instances pour lesquelles une bonne borne sur Δ^{S^*} peut être donnée (un exemple simple : si $\sum_{i=1}^{n} h_j size_j \neq 0 \pmod m$ alors $\Delta^{S^*} \geq 1$). De plus, il suffit de borner Δ^{S^*} pour une solution optimale.

Au chapitre 11, une confrontation expérimentale de ces algorithmes est effectuée. La comparaison théorique est récapitulée dans la table 8.1.

Table 8.1 – Comparaison théorique des heuristiques du chapitre 8.
Le « pire cas » est la meilleure garantie connue ; le « ratio » est la meilleur garantie multiplicative connue.

Algorithme	**Pire cas**	**Ratio**	**Complexité**	
AMP	$H^*_{\max} + \frac{m-1}{m} h_{\max}$ (•)	$2 - \frac{2}{m}$ (•)	$\mathcal{O}(nm)$ (◇)	(1)
AMPD	$H^*_{\max} + \frac{m-1}{m}(h_{\max} - 1)$	$\frac{4}{3}$ (○)	$\mathcal{O}(nm + n \ln n)$ (◇)	
APBS		m (•)	$\mathcal{O}(nm)$ (◇)	(1)
APBSD		2 (•)	$\mathcal{O}(nm + n \ln n)$ (◇)	
APOMD	$H^*_{\max} + h_{\max}$ (†)(•)		$\mathcal{O}(nm + n \ln n)$ (◇)	
ADR	$H^*_{\max} + \frac{m-1}{m} h_{\max}$ (•)	$2 - \frac{2}{m}$ (•)	$\mathcal{O}(nm \ln m)$	
ADOM	$H^*_{\max} + \frac{m-1}{m} h_{\max}$	$2 - \frac{2}{m}$	$\mathcal{O}(nm)$ (◇)	

(•) borne atteinte
(○) pires exemples connus : $4/3 - 1/3m$
(†) conjecture
(◇) complexité optimale
(1) algorithme en-ligne

Il est amusant de remarquer que toutes les instances données pour montrer les limites des algorithmes autres que AMPD... sont résolues optimalement par ce dernier. Cela laisse présager de très bons résultats quant à la meilleure solution donnée par l'ensemble de ces algorithmes.

Le travail théorique sur ces algorithmes mérite encore quelques efforts. Ainsi, AMPD est-elle un peu mieux qu'une 4/3-approximation (question 8.1) ? Quelle est la performance réelle de APOMD (conjecture 8.2) ? existe-t-il une implémentation de ADR qui soit une 4/3-approximation (question 8.3) ? Voici les principales questions restées en suspens.

Une autre approche d'analyse permettrait également de mieux appréhender les qualités de ces algorithmes. Je pense notamment à l'analyse différentielle où la performance d'une heuristique est recadrée dans le spectre de l'ensemble des valeurs des solutions en adoptant le rapport de performance : $\alpha = (W - A)/(W - OPT)$, où A est la valeur de l'algorithme, OPT la meilleure valeur et W la pire valeur possible pour une solution réalisable.

Un schéma d'approximation en temps polynomial

Si tu n'es pas trop long, je t'attendrai ici toute ma vie.
— Oscar Wilde

Dans ce chapitre, je décris un schéma d'approximation en temps polynomial (*Polynomial-Time Approximation Scheme*, PTAS) pour $Bm|size_j|H_{\max}$, c'est-à-dire un algorithme A_ε qui, pour tout ε fixé, est une $(1+\varepsilon)$-approximation avec un temps d'exécution polynomial (pour, rappelons-le, ε et m, fixés). Ce PTAS est basé sur l'algorithme de meilleur placement décroissant (AMPD, voir chapitre 8, section 8.2).

9.1 Schéma d'approximation et conséquences

Une approche classique pour la conception de PTAS est de séparer les objets (au sens large, pas nécessairement des objets multiboîtes), entre les « grands » et les « petits ». Ensuite, on résout optimalement pour l'ensemble des grands objets (par exemple par une recherche exhaustive, possible s'il n'y a pas trop de grands objets), puis on complète la solution avec les petits objets, grâce à une heuristique rapide et performante (ce qui est suffisamment bon si les objets sont suffisamment petits). La clé est de trouver un seuil qui soit un bon compromis pour qu'il n'y ait ni trop de grands objets, ni de petits objets trop grands. Une telle méthode a été appliquée avec succès sur des problèmes liés au rangement d'objets multiboîtes (voir par exemple [79, 33]).

Pour une instance I de $Bm|size_j|H_{\max}$, considérons un seuil β et séparons l'ensemble $\mathbf{O}$ des objets de I de la manière suivante : $\mathbf{O}_{grand} = \{O_j \in \mathbf{O}, h_j > \beta\}$ et $\mathbf{O}_{petit} = \{O_j \in \mathbf{O}, h_j \leq \beta\}$. Si l'on range optimalement les objets de $\mathbf{O}_{grand}$ puis que l'on complète la solution avec l'algorithme AMPD appliqué sur $\mathbf{O}_{petit}$, alors l'algorithme ainsi spécifié (l'algorithme 9.1) est un PTAS, sous réserve d'un choix correct de β.

Théorème 9.1. *L'algorithme 9.1 est un PTAS pour $Bm|size_j|H_{\max}$ dont le temps d'exécution est $\mathcal{O}(n \ln n + nm + 2^{\frac{4m^2}{3\varepsilon}})$.*

Une version préliminaire de ce chapitre a été publiée dans [95] et présentée en diverses occasions [93, 95].

Algorithme 9.1. Un PTAS pour $Bm|size_j|H_{\max}$

- **entrées :** ensemble d'objets $\mathbf{O}$, nombre de boîtes m, réel ε -
1. calculer $H_{approx} = \mathrm{AMPD}(\mathbf{O}, m)$
2. calculer $\beta = \varepsilon\frac{3}{4}H_{approx}$
3. partitionner l'ensemble d'objets $\mathbf{O}$ en :
   $\mathbf{O}_{grand} = \{O_j \in \mathbf{O}, h_j > \beta\}$
   $\mathbf{O}_{petit} = \{O_j \in \mathbf{O}, h_j \leq \beta\}$
4. ranger optimalement $\mathbf{O}_{grand}$
5. ranger $\mathbf{O}_{petit}$ avec AMPD

PREUVE — Calculons une solution selon l'algorithme 9.1. Lors de l'étape 1, H_{approx} est calculé par AMPD ; d'après le théorème 8.4 :

$$H_{approx} \leq \frac{4}{3}H^*_{\max},$$

donc la plus grande hauteur d'un petit objet vérifie :

$$h_{\max}(\mathbf{O}_{petit}) \leq \beta \leq \varepsilon\frac{3}{4}H_{approx} \leq \varepsilon H^*_{\max}.$$

Le théorème 8.3 permet alors d'affirmer :

$$H_{\max} \leq H^*_{\max} + h_{\max}(\mathbf{O}_{petit}) \leq (1+\varepsilon)H^*_{\max},$$

et l'algorithme 9.1 est une $(1+\varepsilon)$-approximation.

Les temps de calcul sont de $\mathcal{O}(n\ln n + nm)$ pour l'étape 1 ; $\mathcal{O}(1)$ pour l'étape 2 ; $\mathcal{O}(n)$ pour l'étape 3 et au pire $\mathcal{O}(n\ln n + nm)$ pour l'étape 5. Pour l'étape 4, un rangement optimal peut être calculé par recherche exhaustive. Soit K le nombre de rangements possibles de $\mathbf{O}_{grand}$:

$$K \leq \prod_{O_j \in \mathbf{O}_{grand}} \mathrm{C}_m^{size_j} \leq \prod_{O_j \in \mathbf{O}_{grand}} 2^m \leq 2^{m|\mathbf{O}_{grand}|}.$$

Or :

$$|\mathbf{O}_{grand}| \leq \sum_{O_j \in \mathbf{O}_{grand}} size_j \leq \frac{mH^*_{\max}}{\beta} \leq \frac{4m}{3\varepsilon}$$

(la première inégalité vient du fait que les largeurs sont toutes au moins de un ; la deuxième est impliquée par le volume total de $\mathbf{O}_{grand}$ qui doit être plus petit que $mH^*_{\max}$; la troisième vient de la définition de β).

En combinant les deux dernières inéquations :

$$K \leq 2^{\frac{4m^2}{3\varepsilon}}.$$

Une recherche exhaustive sur $\mathbf{O}_{grand}$ peut donc être faite en temps $\mathcal{O}(2^{\frac{4m^2}{3\varepsilon}})$.

Finalement, le temps global d'exécution est $\mathcal{O}(n\ln(n) + nm + 2^{\frac{4m^2}{3\varepsilon}})$. ⊗

L'existence de ce PTAS, basé sur AMPD, a des conséquences sur les garanties asymptotiques de cette heuristique. Ainsi, AMPD est une $(1+\varepsilon)$-approximation pour tout ε tel que $\mathbf{O}_{grand}$ est vide, c'est-à-dire pour ε tel que :

$$h_{\max} \leq \beta = \varepsilon \frac{3H_{approx}}{4} \iff \varepsilon \geq \frac{4h_{\max}}{3H_{approx}}.$$

Or H_{approx} est la valeur de la solution calculée par AMPD. Aussi la garantie suivante est valide :

Corollaire 9.1.1. *Une solution calculée par AMPD vérifie :*

$$H_{\max} \leq \left(1 + \frac{4h_{\max}}{3H_{\max}}\right) H^*_{\max}.$$

AMPD est donc asymptotiquement un schéma d'approximation en temps fortement polynomial (*Fully Polynomial-Time Approximation Scheme*, FPTAS) pour les instances où $h_{\max}$ est bornée par une constante : pour tout ε, il existe une borne H telle que $H^{AMPD}_{\max} \leq (1+\varepsilon)H^*_{\max}$ si $H^*_{\max} \geq H$, et AMPD a un temps d'exécution polynomial en la taille de l'instance et ε.

Ceci est similaire au corollaire 8.1.1 et a la même conséquence pratique : AMPD est très performant, spécialement sur les instances avec un grand nombre d'objets.

9.2 Compléments

Un PTAS peut aussi être obtenu en se basant, non pas sur les théorèmes 8.4 et 8.3, mais sur le corollaire 8.1.2 et le théorème 8.2 ; les bases étant plus faibles, le résultat est cependant moins bon : chaque rangement possible pour les grands objets doit être considéré et complété par un algorithme AMP, ce qui mène à un algorithme en temps $\mathcal{O}(2^{\frac{2m(m-1)^2}{\varepsilon}} nm)$ [96]. Les principes restent toutefois les mêmes.

De plus un résultat meilleur, au moins sur le plan théorique, que ce PTAS a déjà été prouvé au chapitre 5 : il existe un FPTAS pour $Bm|size_j|H_{\max}$. Le temps d'exécution de ce dernier est $\mathcal{O}(2^m n(\frac{n^2}{\varepsilon})^m)$, ce qui, en définitive, le rend tout aussi inexploitable en pratique.

Sur des cas particuliers, comme Bi-Partition ou l'Ordonnancement sur machines parallèles, les performances du PTAS peuvent être calculées plus finement et ainsi améliorées (voir annexe B, section B.1). Il ne rivalise toutefois pas avec les PTAS conçus spécialement pour ces problèmes. Ainsi, dans [68], une approche basée sur l'approximation duale permet d'obtenir un PTAS pour l'Ordonnancement sur machines parallèles dont le temps d'exécution est $\mathcal{O}((n/\varepsilon)^{1/\varepsilon^2})$ (donc, même pour un nombre de machines non fixé). Pour plus de détails sur les schémas d'approximation existants pour ces cas particuliers, le lecteur est renvoyé à [10] et [67].

Il serait intéressant, dans la continuité de ce chapitre, d'avoir une réponse à la conjecture 8.1 : s'il existe toujours un ordre pour lequel AMP est optimal, l'étape 4 de l'algorithme 9.1 pourrait être accélérée. Une autre alternative à la recherche exhaustive pour la résolution optimale de $\mathbf{O}_{grand}$ est la méthode pseudo-polynomiale du chapitre 5. L'algorithme qui en découle a un temps d'exécution de $\mathcal{O}(A^m/\varepsilon + n\ln n)$, pour $A = \sum h_j$ et m fixé, ce qui n'est que pseudo-polynomial, mais beaucoup plus prometteur pour une résolution réelle, surtout pour des petits objets[1] (c'est en fait un FPTAS si la hauteur des objets est bornée par une constante indépendante de l'instance).

Une autre extension serait d'utiliser le FPTAS du chapitre 5. Ainsi, résoudre $\mathbf{O}_{grand}$ grâce au FPTAS du chapitre 5 puis compléter la solution selon la règle MP, donne un algorithme dont le temps d'exécution, pour m fixé, est de $\mathcal{O}(n\ln n + 1/\varepsilon^{3m+1})$; le problème est alors :

Question 9.1 : Quelle garantie a-t-on si $\mathbf{O}_{grand}$ est résolu à ε près, puis complété avec $\mathbf{O}_{petit}$ par AMPD ?

et plus généralement :

Question 9.2 : Peut-on construire un FPTAS pour $Bm|size_j|H_{\max}$, utilisable en pratique, par exemple en conjuguant le FPTAS du chapitre 5 et le PTAS du chapitre 9 ?

[1] Il reste toutefois un facteur multiplicatif, une « constante » de l'ordre de 2^m...

Une approche évolutionniste

La Vie est une chose beaucoup trop importante
pour en parler sérieusement.
— Oscar Wilde

Les heuristiques du chapitre 8 sont efficaces. En des temps très courts, elles permettent de résoudre très bien, souvent même optimalement, la plupart des instances de $B|size_j|H_{\max}$. Il reste cependant certains types d'instances assez mal résolus. Pour ces instances-là, je me suis tourné vers une méta-heuristique : les algorithmes génétiques.

Mon choix a d'abord été guidé par une curiosité personnelle envers ces méthodes ; la littérature m'a rapidement convaincu que l'approche était prometteuse, d'excellents résultats sur des problèmes assez proches étant rapportés.

L'algorithme proposé est hybride, construit à partir des heuristiques AMP et ADR (voir chapitre 8, sections 8.2 et 8.5), afin de profiter des excellentes propriétés de celles-ci. Les résultats obtenus sont tout à fait satisfaisants.

Dans la suite de ce chapitre, je commence par présenter les algorithmes génétiques ; les experts du domaine peuvent donc passer la première section. Les deux autres sections décrivent l'algorithme proposé avec ses différentes variantes, et son comportement selon la manière de le configurer.

10.1 Les algorithmes génétiques

L'histoire des algorithmes génétiques commence véritablement dans les années soixante-dix, avec les travaux de Holland [69], puis de De Jong [41]. Pour Holland, il s'agissait de copier Dame Nature dans son processus d'optimisation, où chaque espèce doit maximiser sa capacité de survie — ou disparaître.

Holland résuma ce processus à quelques règles simples inspirées de la sélection naturelle darwinienne et de découvertes en génétique[1]. Tout d'abord, d'un point de vue génétique, un individu est entièrement codé et déterminé par ses gènes ; de plus les individus se reproduisent par copie, échange et mutation de leur matériel génétique (ces points sont détaillés plus bas). Ensuite entre en jeu la sélection natu-

[1]Ce qui suit, y compris dans les sections suivantes, n'est heureusement que la traduction informatique de phénomènes biologiques, voire sociaux. Il ne faudra donc rien interpréter hors du champ des algorithmes (génétiques).

relle : un individu aura d'autant plus de chance de survivre et/ou de se reproduire qu'il est bien adapté, c'est-à-dire de bonne qualité.

Dans les algorithmes génétiques initiaux de Holland, cela est réalisé de la manière suivante. La première gageure est de trouver un codage binaire pour les solutions du problème à résoudre : ce codage est l'unique chromosome des individus considérés, et chaque *bit* est un gène. À partir d'une population, on crée la population suivante par reproduction. Un rejeton est engendré par une paire de parents sélectionnés au hasard en tenant compte de leurs qualités ; chaque rejeton est une copie d'un parent, à laquelle on fait subir des échanges de matériel génétique avec l'autre parent (phénomène de croisement) et des altérations de ses gènes (phénomène de mutation).

Cette méthode d'optimisation est très générale : elle est en effet complètement aveugle, et le problème à optimiser n'intervient que dans la fonction d'évaluation d'un individu. Elle est aussi, théoriquement, très performante : les principes simples ci-dessus permettent une émergence très rapide des sous-séquences de gènes caractéristiques des bons individus (théorème des schémas de Holland — voir [104, chapitre 3]). Toutefois deux défauts majeurs apparaissent. En premier lieu la généralité de l'approche ne la rend pas pour autant adaptée à tout problème, et il faut parfois une gymnastique ardue pour réussir à coder un problème en une suite de 0 et 1 et un surcoût de calcul prohibitif pour maintenir une population saine, c'est-à-dire sans solution irréalisable. Secondement, ses qualités théoriques sont limitées par les réalisations pratiques et les contraintes supplémentaires qu'elles imposent : ainsi les algorithmes génétiques se retrouvent fréquemment piégés dans des *extrema* locaux ou à converger très lentement — phénomènes hautement dépendants des réglages adoptés, par exemple le taux de mutations. Ces travers ne sont cependant pas propres aux algorithmes génétiques et on les trouve également chez les autres méta-heuristiques, vis-à-vis desquelles ils sont très compétitifs.

Pour pallier ces défauts, les algorithmes génétiques se diversifièrent — une mise en application de ces mêmes principes qu'ils mettent en œuvre... À partir du milieu des années quatre-vingt, De Jong préconisa d'abandonner le modèle de chromosomes binaires, lorsque ceux-ci s'adaptent mal à un problème, pour se tourner vers d'autres codages, spécifiques aux cas traités (par exemple une permutation de n éléments pour un problème de voyageur de commerce entre n villes[2]). Plus généralement, l'idée d'incorporer des connaissances du problème à traiter dans les algorithmes génétiques se fit de plus en plus présente[3]. Ainsi Davis [40] propose l'utilisation d'algorithmes hybrides : des algorithmes génétiques basés sur les heuristiques de résolution du

[2]Pour reprendre les mots de De Jong sur le problème du voyageur de commerce (cité dans [104, page 6]) : « Using the standard crossover and mutation operators, a genetic algorithm will explore the space of all *combinations* of city names when, in fact, it is the space of all *permutations* which is of interest. The obvious problem is that as n [the number of cities] increases, the space of permutations is a vanishingly small subset of the space of combinations, and the powerful genetic algorithm sampling heuristic has been rendered impotent by a poor choice of representation. »

[3]Davis (cité dans [104, page 8]) : « It has seemed true to me for some time that we cannot handle most real-world problems with binary representations. [...] One reason for that is that nearly every real-world domain has associated domain knowledge that is of use when one is considering a transformation of a solution in the domain. » — voir aussi [40].

problème disponibles. C'est un algorithme de ce type que j'ai développé.

Ces quelques paragraphes n'ont fait qu'effleurer le champs des algorithmes génétiques, et plus généralement des algorithmes évolutionnistes[4]. Pour plus de détails je renvoie le lecteur aux excellents ouvrages de Davis [40] et Michalewicz [104]. Plus particulièrement, je recommande la première partie de [104] pour répondre aux questions : les algorithmes génétiques, que sont-ils ? comment fonctionnent-ils ? pourquoi fonctionnent-ils ? Le programmeur intéressé par développer ses propres algorithmes génétiques trouvera les précisions nécessaires dans [40].

10.2 Un algorithme hybride pour $B|size_j|H_{\max}$

J'ai développé un algorithme génétique hybride, selon la terminologie de Davis [40], pour résoudre le problème $B|size_j|H_{\max}$. Dans cette section, je commence par expliquer en quoi et pourquoi cet algorithme est hybride. Je décris ensuite les opérateurs de mutations et de croisement utilisés, puis la procédure de reproduction.

10.2.1 Un algorithme hybride

Pourquoi un algorithme hybride ? Il est vrai qu'un codage binaire peut être réalisé facilement pour coder le rangement d'objets dans des boîtes : par exemple par le biais d'une matrice binaire, un élément $a_{i,j}$ valant 1 si et seulement si l'objet O_j est rangé dans la boîte B_i. Malheureusement, si on veut utiliser un tel codage pour des objets multiboîtes, on se heurte à deux problèmes. Tout d'abord, il devient coûteux de vérifier — et de rectifier le cas échéant — qu'une solution est réalisable, c'est-à-dire qu'un objet O_j est rangé exactement $size_j$ fois. Ensuite, et surtout, un objet O_j a $\mathrm{C}_m^{size_j}$ possibilités d'être rangé : or, beaucoup des rangements possibles sont équivalents (le problème possède de nombreuses symétries), ce qui aboutit à un espace de recherche disproportionné par rapport à l'espace des solutions « utiles ». J'ai donc préféré imposer une restriction sur les solutions, afin d'obtenir un algorithme génétique fonctionnel.

En quoi l'algorithme est-il hybride ? J'ai suivi les recommandations de Davis [40, chapitre 4] : pour le problème qui nous occupe, je disposais de plusieurs algorithmes très rapides et efficaces (voir chapitre 8) ; j'ai donc décidé de les utiliser comme briques de base. Parmi ceux-ci, ma préférence est allée à deux algorithmes de listes : AMP (chapitre 8, section 8.2) et ADR (chapitre 8, section 8.5).

J'ai ainsi opté pour la représentation suivante : une solution, pour l'algorithme génétique, est une permutation de l'ensemble des objets ; sa valeur est la valeur atteinte par AMP (ou ADR) appliqué à cette permutation.

[4]Selon la définition de Michalewicz [104], est évolutionniste tout algorithme basé sur la répétition de trois phases : sélection d'une population à partir de la population précédente, altération de cette population, évaluation de cette population. Les algorithmes génétiques sont de tels systèmes, mais plusieurs autres existent également.

10.2.2 Altération d'un individu

L'algorithme étant basé sur des permutations, j'ai adopté des opérateurs de mutation et de croisement développés pour ce genre de codage et recommandés par Davis [40]. Ils ont déjà été appliqués avec succès, notamment sur le problème du voyageur de commerce et des problèmes d'ordonnancement.

Mutation

L'opérateur de mutation est le « *scramble list mutation* ». Pour celui-ci, on choisit une sous-séquence qu'on remplace par l'une de ses permutations. La figure 10.1 donne un exemple d'une telle mutation.

```
solution initiale :   1 2 3 4 5 6 7 8 9
une sous-séquence :   . . 3 4 5 6 . . .
      permutation :   . . 4 6 5 3 . . .
  solution mutée :    1 2 4 6 5 3 7 5 9
```

Figure 10.1 – Un exemple de mutation.

Croisement

L'opération de croisement est le « *uniform order-based crossover* ». On commence par sélectionner un certain nombre de positions : pour celles-ci, on recopie les éléments du parent 1 dans ces positions. Les places vacantes de l'enfant sont remplies avec les éléments restants, dans l'ordre où ils apparaissent dans le deuxième parent. La figure 10.2 donne un exemple d'un tel croisement.

```
parent 1 :  1 2 3 4 5 6 7 8  →  . 2 3 . 5 6 . .
parent 2 :  8 6 4 2 7 5 3 1  →  8,4,7,1
  enfant :  8 2 3 4 5 6 7 1
```

Figure 10.2 – Un exemple de croisement.

10.2.3 Gestion de la population

Un enfant est calculé à partir de deux parents, de manière dissymétrique mais habituelle[5]. Le premier parent est copié exactement pour former l'enfant. Sont ensuite, éventuellement, appliqués une mutation et un croisement avec le deuxième parent. Le nouvel individu ainsi créé est alors intégré dans la population globale.

Que ce soit pour la sélection des parents ou l'intégration dans la population, différentes méthodes issues de la littérature [40, 104] ont été implémentées.

[5] Habituelle pour les algorithmes génétiques...

Sélection des parents

La sélection des parents est réalisée selon le principe de la roulette : ils sont choisis au hasard parmi l'ensemble de la population, avec une probabilité d'autant plus grande qu'ils sont bien adaptés. Le nom de « roulette » vient de l'analogie que l'on peut faire avec un tirage au sort sur une roue divisée en secteurs proportionnés à l'adaptation des individus.

L'adaptation d'un individu reflète directement la valeur de la solution qu'il représente. La méthode de calcul la plus simple est appelée *fenêtrage* : à un individu de valeur v, on associe l'adaptation $1 + v_\infty - v$, où v_∞ est la valeur de la pire solution dans la population actuelle.

Un algorithme génétique se doit d'être eugéniste. La méthode de *fenêtrage*, et son reflet trop fidèle de la valeur intrinsèque de chaque individu, ne correspond donc pas toujours à cette nécessité d'agir en faveur des plus favorisés pour faire ressortir une élite. L'alternative proposée est celle de l'*adaptation linéaire*. Pour la calculer, deux paramètres sont nécessaires : l'adaptation maximale MAX et le coefficient de diminution $DECR$. Ceux-ci étant définis, on attribue l'adaptation MAX aux individus de valeur maximale, puis $MAX - DECR$ aux individus juste inférieurs, et on continue ainsi en diminuant de $DECR$ pour chaque nouveau palier, jusqu'à attribuer une adaptation nulle aux éventuels individus restants.

Pour un bon choix de $DECR$, cette approche permet de creuser les écarts entre des individus proches et donc favorise bien l'élite. Les paramètres $DECR$ et MAX sont cependant critiques et doivent être choisis avec soin, en fonction du problème, afin de ne pas engendrer une domination trop forte qui entraînerait la perte de la diversité génétique, essentielle au bon déroulement de l'algorithme.

Intégration dans la population

Les enfants ayant été engendrés, il faut les intégrer à la population globale. La manière la plus simple de le faire est de repartir à zéro avec une population flambant neuve : à partir de p parents, on engendre p enfants, puis tous les géniteurs sont supprimés et remplacés par leurs rejetons.

Une deuxième approche vise à favoriser dès l'intégration à la population les individus méritants. Pour cela on ne génère qu'un enfant à la fois, que l'on insère dans la population à la place du pire individu qu'elle contient (ce peut être un parent, un oncle ou même un frère). Pour préserver la diversité dans cette méthode par remplacement, un individu ayant un jumeau dans la population n'est pas inséré : à la place un nouvel enfant est calculé[6].

Pour l'une comme pour l'autre des méthodes, l'algorithme offre la possibilité d'élitisme : si cette option est activée, le meilleur individu d'une population est systématiquement copié et reporté dans la population suivante.

[6]En pratique, l'algorithme accepte un doublon si les x précédents enfants créés avaient aussi un jumeau ; ceci afin de préserver des temps de calcul corrects, même face à des populations dégénérées n'engendrant que peu de diversité (x est un paramètre de l'algorithme).

Population initiale

La population initiale de l'algorithme n'a rien de particulier : chaque individu est une permutation aléatoire, déterminée indépendamment des autres. Le seul paramètre est la taille de la population, qui restera constant tout au long de l'algorithme.

10.3 Configuration de l'algorithme

L'algorithme décrit ci-dessus possède différents paramètres, dont les valeurs influent fortement sur sa qualité. Cette section présente une étude empirique du réglage de ces paramètres afin d'obtenir de bonnes performances, ainsi que le comportement général de l'algorithme qui en résulte.

10.3.1 Réglages des paramètres

Les réglages des paramètres proposés ici sont issus d'une étude empirique « à la main ». Le but est essentiellement de dégager les forces et les faiblesses de l'algorithme génétique, et d'appréhender son potentiel.

Dans le cas d'une application industrielle, où aucun gain n'est négligeable, une analyse plus approfondie serait nécessaire. L'utilisation d'un algorithme génétique pour régler les paramètres de celui-ci pourrait, par exemple, être faite.

Choix des données

La plupart des instances de $B|size_j|H_{\max}$ se résolvent très bien. Pour les petites instances, le peu d'objets permet d'en avoir vite exploré les possibilités ; pour les grandes instances, quelques échanges d'objets permettent d'obtenir une solution quasi-optimale, ne laissant pas beaucoup de place à l'amélioration.

Pour pouvoir véritablement mettre à l'épreuve l'algorithme génétique, j'avais besoin d'instances résistantes, pour lesquelles une bonne marge de progression existe. Dans la batterie de tests, mon choix s'est porté sur deux instances particulières, que je nommerai A et B, et dont les principales caractéristiques sont données dans la table 10.1. Comme la plupart de leurs consœurs posant problèmes, ces deux instances ont la particularité de n'avoir que de grands objets, avec peu de différence de hauteur entre eux.

Table 10.1 – Caractéristiques des instances pour le réglage de l'algorithme génétique.

	données						**valeurs**	
	m	n	$h_{\min}$	$h_{\max}$	$size_{\min}$	$size_{\max}$	L_V	AMPD
A	18	726	48	50	12	18	29646	29664
B	20	486	37	40	14	20	15813	15836

Valeurs des paramètres

Les très nombreux paramètres et leurs fortes interactions rendent impossible un réglage « parfait », optimal pour toutes les instances. Certains choix sont cependant clairement mauvais ; d'autres robustes. Voici les conclusions auxquelles différents tests m'ont amené pour cet algorithme génétique.

La taille de la population est un facteur clé pour lequel un mauvais choix ruine complètement toutes les possibilités de l'algorithme. L'essentiel est de ne pas imposer une taille trop réduite afin d'avoir une population assez variée. D'un autre côté, plus une population est grande, plus le nombre de générations nécessaires pour en tirer parti croît, ce qui mène à des temps de calcul très longs.

Aussi n'y-a-t-il pas d'optimum identifiable : selon l'instance, mais aussi la méthode utilisée (AMP ou ADR) et le nombre total d'individus autorisé, la bonne valeur pour la taille de la population est variable. Des populations inférieures à 20 (et surtout à 10) sont généralement trop peu diversifiées et leurs améliorations sont beaucoup trop lentes. À l'opposé, une population de 40 n'apporte un gain qu'après un total de plusieurs milliers d'individus générés : un tel choix sera donc pertinent lorsqu'on dispose de tout son temps, mais il vaudra mieux se limiter à des valeurs un peu moindres pour avoir un algorithme rapidement efficace.

Une valeur de 35 individus par génération m'a semblé un choix de compromis robuste, qui permet des gains assez importants dès le début de la recherche, et qui ne s'essouffle pas si l'on veut poursuivre sur un grand nombre de générations. Pour la suite, je me limite à 3500 individus générés au total, soit 100 générations.

L'évolution naturelle donne à penser que la reproduction sexuée est plus à même d'engendrer des individus évolués que la simple parthénogénèse[7]. Il ne sera donc pas étonnant si, pour de bons résultats, la probabilité de croisement doit être relativement élevée.

Il s'avère ainsi que, pour l'un des taux de croisement ou de mutation nul, la convergence de l'algorithme est globalement d'autant meilleure que l'autre taux est grand — avec tout de même un seuil limite à ne pas dépasser, une probabilité de mutation ou de croisement de l'ordre de 0,9.

Lorsqu'on conjugue les effets des deux opérateurs, les performances sont accrues, mais il faut alors se limiter à des valeurs moindres. Un couple de valeurs $(0,6; 0,5)$ pour mutation et croisement m'a paru adéquate.

Pour le calcul de l'adaptation d'un individu, la méthode par fenêtrage est nettement dominée par l'adaptation linéaire, si les paramètres de cette dernière sont bien choisis. Pour une adaptation maximale $MAX = 100$, un facteur de décroissance $DECR$ de 36 donne de très bons résultats.

[7]Des arguments mathématiques viennent aussi appuyer le fait que le croisement est une opération essentielle, qui permet une meilleure convergence que la simple mutation, grâce à une propagation plus rapide des bonnes caractéristiques au sein de la population [104].

Pour la méthode d'intégration dans la population, aucun des différents choix n'est véritablement dominant, ni dominé. Ma préférence va donc au remplacement total, sans élitisme, qui présente l'avantage de la simplicité et de la rapidité. Remarquons toutefois que pour un remplacement individuel, sans jumeau, l'élitisme a tendance à être bénéfique, tandis qu'il dégrade légèrement les performances dans le cas d'un remplacement complet de la population.

Pour finir sur les paramètres, signalons que l'algorithme AMP semble plus adapté que ADR pour l'algorithme génétique. Les résultats sont généralement meilleurs avec une convergence plus franche, c'est-à-dire à la fois plus rapide et vers une meilleure valeur.

10.3.2 Comportement de l'algorithme

Pour un choix de paramètres « standard », tel que décrit ci-dessus, et même pour un assez large éventail de réglages, le comportement de l'algorithme est satisfaisant, et ce dernier remplit bien le rôle pour lequel il a été proposé. Pour une analyse expérimentale des gains réels procurés, le lecteur se reportera au chapitre 11.

Comme pour à peu près n'importe quelle méta-heuristique, l'essentiel des gains est obtenu en début de recherche ; des améliorations sont toutefois encore gagnées même après un très grand nombre de générations.

La taille de la population (35) est habituelle pour un algorithme génétique (généralement 30 à 50). De même que le taux de croisement, lorsqu'il n'y a pas de mutation (0,8-0,9, pour un standard de 0,7 à 0,95).

Dans la combinaison mutation-croisement, les taux donnant de bons résultats sont par contre très surprenants, avec une probabilité de mutation très grande, plus grande même que celle de croisement (0,6 contre 0,5). Des taux classiques seraient d'au plus 0,05 contre au moins 0,7 !

Grossièrement, le rôle du croisement est de rassembler et propager les bonnes caractéristiques des meilleurs individus au sein de l'ensemble de la population, tandis que la mutation est censée permettre d'atteindre de nouvelles régions inexplorées. Les taux classiques sont donc assez intuitifs : faire peu de changements à la fois, et en tirer parti rapidement.

Un taux de mutation aussi élevé (0,6) donne à penser que l'algorithme génétique proposé est encore par trop aléatoire : cela ouvre des perspectives de nettes améliorations. La cause en est peut-être dans le choix des opérateurs, et le croisement choisi n'est sans doute pas aussi adapté au problème et à son codage que ne l'est la mutation.

10.4 Extensions

Comme cela a été signalé, le taux de mutation anormalement élevé laisse à penser qu'un travail sur les opérateurs de croisement et de mutation est nécessaire.

Cependant, pour améliorer encore les performances, ma piste préférée est de permettre une population de taille variable. En effet, cette taille de population est un facteur critique et déterminant : il fait la balance entre diversité et sélectivité, entre ne pas perdre de temps à explorer un espace inutilement grand et ne pas se laisser piéger dans un recoin.

Michalewicz propose une solution élégante aux résultats étonnants [104] : introduire une notion d'âge. Un individu n'est alors jamais remplacé par un autre : il meurt simplement une fois son temps révolu ; son espérance de vie reflète son adaptation. Cette technique est radicalement opposée à la méthode de remplacement global de la population à chaque itération. Sur quelques exemples, Michalewicz [104] montre que l'algorithme adapte parfaitement la taille de la population au besoin : celle-ci commence par croître, pour une exploration conséquente de l'espace de recherche, puis diminue lorsqu'il devient nécessaire de se concentrer sur les qualités de l'élite pour produire la meilleure solution.

Comparaison expérimentale des algorithmes

Il y a si loin de la façon dont on vit à celle dont on devrait vivre,
que celui qui laisse ce qui se fait pour ce qui se devrait faire
apprend plutôt à se détruire qu'à se préserver.
— *Niccolò Machiavel*

Ce chapitre fait office à la fois de complément essentiel et de conclusion à cette partie sur le résolution de $B|size_j|H_{\max}$: il s'agit de confronter les algorithmes proposés précédemment à de véritables instances, et non plus simplement de chercher des performances théoriques.

Ne disposant pas d'instances réelles, toutes les données ont été générées aléatoirement. Comment cela a été fait est le sujet de la première section. Les sections suivantes analysent, sur la base de tests expérimentaux, respectivement les bornes inférieures, les heuristiques et l'algorithme génétique. Pour terminer ce chapitre, et cette partie, une petite section de conclusion met l'accent sur les points pour lesquels la résolution a le plus besoin d'être améliorée.

Tous les algorithmes étudiés ici sont très rapides, aussi ce chapitre se concentre sur la qualité des solutions produites. Pour avoir une idée des temps de calcul, le lecteur se reportera à l'annexe B, section B.2.

11.1 Les données

11.1.1 Méthodes de génération des données

Les données utilisées pour mener à bien les tests sont toutes générées automatiquement, selon différentes méthodes. Voici leurs descriptions.

Génération aléatoire uniforme

Pour deux paires d'entiers $(h_a < h_b)$ et $(s_a < s_b)$, les hauteurs des objets suivent une loi uniforme dans l'intervalle $\{h_a, \ldots, h_b\}$, et les largeurs une loi uniforme dans l'intervalle $\{s_a, \ldots, s_b\}$.

Génération aléatoire gaussienne

Pour un triplet d'entiers $(h_a < h_b, v_h)$, les hauteurs des objets suivent une loi gaussienne de moyenne $(h_a + h_b)/2$, de variance v_h, tronquée dans l'intervalle $\{h_a, \ldots, h_b\}$.

Similairement, pour un triplet d'entiers $(s_a < s_b, v_s)$, les largeurs des objets suivent une loi gaussienne de moyenne $(s_a + s_b)/2$, de variance v_s, tronquée dans l'intervalle $\{s_a, \ldots, s_b\}$.

Largeur fixée

On attribue une même largeur s donnée à tous les objets, tandis que les hauteurs sont choisies uniformément dans un intervalle $\{h_a, \ldots, h_b\}$.

Escaliers

Trois variantes existent. Pour la première, la hauteur des objets commence à une valeur donnée h_a et croît de un en un jusqu'à une valeur seuil h_b fixée (puis repart à h_a). Les largeurs sont tirées au hasard, uniformément, dans un intervalle $\{s_a, \ldots, s_b\}$.

La deuxième variante échange les rôles des hauteurs et des largeurs. Pour la troisième, les deux dimensions sont sujettes à la croissance.

Objets complémentaires

Les objets sont créés par paires (s'il y en a un nombre pair) : la largeur $size_1$ du premier objet est choisie aléatoirement, uniformément dans l'intervalle $\{1, \ldots, m-1\}$ (m étant le nombre de boîtes), et la largeur du deuxième objet est complémentaire : $size_2 = m - size_1$.

Pour les hauteurs, deux variantes existent : soit les deux hauteurs sont identiques, soit elles sont indépendamment tirées uniformément dans un intervalle $\{h_a, \ldots, h_b\}$ donné.

11.1.2 Les données utilisées

Quatre groupes d'instances ont été générés, correspondant chacun à des tailles d'instances différentes : de 5 à 25 objets, de 25 à 60, de 60 à 100 et de 100 à 500.

Pour chaque groupe, 500 instances ont été construites aléatoirement, parmi lesquelles 205 correspondent à la génération aléatoire uniforme et forment l'ensemble U ; 217 correspondent à la génération aléatoire gaussienne et forment l'ensemble G ; 78 correspondent aux autres méthodes et forment l'ensemble X.

Les différents paramètres des méthodes ont été choisis aléatoirement et uniformément dans l'intervalle $\{1, \ldots, 50\}$; le nombre de boîtes a été tiré uniformément dans l'intervalle $\{2, \ldots, 20\}$.

À travers les diverses expériences préliminaires que j'ai eu menées, il m'est apparu que le nombre d'objets est un paramètre beaucoup plus influent que le nombre de boîtes, pour ce qui est de la difficulté de résolution. C'est pourquoi ce paramètre est mis en avant. Similairement, la hauteur des objets est beaucoup plus critique que leur largeur ; par exemple, les instances où la plus petite hauteur est grande et peu différente de la plus haute, se montrent particulièrement difficiles. Ce deuxième paramètre critique n'a pas été explicitement privilégié dans le choix des instances générées, toutefois celui-ci est suffisamment varié pour faire ressortir les qualités et les limites des algorithmes testés. Dans l'analyse des résultats qui suit, je reviendrai sur les types d'instances posant particulièrement problème.

11.2 Les bornes inférieures

Commençons par l'analyse des bornes inférieures pour $B|size_j|H_{\max}$.

11.2.1 Bornes testées

En plus des bornes décrites au chapitre 7, d'autres bornes, expérimentales, ont été utilisées. En effet, les garanties des algorithmes du chapitre 8 permettent, à partir d'une solution, de déduire une borne inférieure sur la valeur optimale ; et plus une solution est mauvaise, plus la borne déduite est bonne !

À partir de cette constatation, des bornes ont été calculées à partir de solutions construites par les algorithmes AMP, AMPD et ADR, notamment en prenant les pires solutions parmi 200 engendrées par AMP et ADR pour des ordres aléatoires des objets.

Les différentes bornes inférieures testées sont ainsi :

- L_V : la borne de volume décrite chapitre 7, section 7.1. À noter que la version testée tient compte du seul volume, et aucunement de la plus grande hauteur.
- $L_{n_{\min}}$: la borne de « raisonnement énergétique », présentée au chapitre 7, section 7.2, telle qu'elle est décrite par l'algorithme 7.1.
- $L_{\max}$: la meilleure de L_V et $L_{n_{\min}}$.
- L_x : la meilleure des bornes données par la plus grande hauteur d'un objet et les heuristiques AMP, ADR et AMPD.

11.2.2 Résultats

Les résultats sont résumés dans la table 11.1 et la figure 11.1. La discussion qui suit apporte les indispensables et nombreux compléments et précisions.

Ces résultats démontrent avant tout l'excellent comportement de L_V, même pour de petites l'instances. Comme l'annonçait déjà la théorie, lorsque la taille grandit, cette borne ne laisse qu'une faible marge de progression pour de futures améliorations. Cette marge est d'autant plus réduite que les pourcentages de valeurs optimales et les écarts sont basés sur la meilleure valeur connue, qui peut ne pas être optimale.

Table 11.1 – Comparaison des bornes inférieures pour $B|size_j|H_{\max}$.

type - n	**meilleure** (%)				**optimale** (%)				**écart max** (%)				**écart moyen** (%)			
	L_V	$L_{n_{\min}}$	$L_{\max}$	L_x	L_V	$L_{n_{\min}}$	$L_{\max}$	L_x	L_V	$L_{n_{\min}}$	$L_{\max}$	L_x	L_V	$L_{n_{\min}}$	$L_{\max}$	L_x
U - 5..25	83,41	84,39	100	0	70,73	78,05	81,95	0	18,99	42,70	11,39	25	1,17	1,95	0,38	9,00
G - 5..25	87,56	82,94	100	0	75,12	78,34	82,95	0	14,91	41,46	11,97	25	0,87	1,69	0,24	9,13
X - 5..25	89,74	56,41	100	0	66,67	56,41	76,92	0	16,22	36,47	2,78	25	0,88	5,27	0,17	9,27
U - 25..60	95,12	72,68	100	0,00	86,34	69,76	88,78	0,00	2,85	45,24	0,98	7,10	0,09	3,05	0,04	2,67
G - 25..60	94,93	76,50	100	0,00	84,33	71,43	85,25	0,00	3,64	58,16	0,92	6,32	0,11	2,45	0,05	2,67
X - 25..60	94,87	51,28	100	1,28	83,33	48,72	85,90	1,28	4,87	37,55	0,71	6,89	0,18	7,82	0,04	2,53
U - 60..100	98,72	67,95	100	0	89,74	67,95	91,03	0	0,75	50,31	0,22	2,52	0,02	4,20	0,01	1,30
G - 60..100	98,62	72,35	100	0	86,64	71,89	87,56	0	0,83	55,14	0,59	3,09	0,02	2,67	0,02	1,32
X - 60..100	100	47,44	100	0	94,87	47,44	94,87	0	0,56	36,63	0,56	2,88	0,01	8,36	0,01	1,26
U - 100..500	98,05	68,78	100	0	89,27	68,78	91,22	0	1,59	50,56	0,10	1,80	0,02	3,59	0	0,42
G - 100..500	99,54	70,97	100	0	91,24	70,97	91,71	0	0,19	62,27	0,05	1,78	0,00	2,93	0	0,41
X - 100..500	98,72	47,44	100	0	92,31	47,44	93,59	0	0,90	49,83	0,03	1,29	0,01	9,08	0	0,40

« meilleure » : pourcentage des cas où une borne est la meilleure parmi les bornes testées ;
« optimale » : pourcentage des cas où une borne est optimale (la comparaison est faite avec la meilleure solution connue) ;
« écart max » et « écart moyen » : écarts maximum et moyen d'une borne, par rapport à la meilleure solution connue.

L_V est la borne de volume seul ($L_V = \left\lceil \sum_j size_j h_j / m \right\rceil$) ;
$L_{n_{\min}}$ est la borne décrite chapitre 7, section 7.2 ;
$L_{\max}$ est la meilleure de L_V et $L_{n_{\min}}$;
L_x est la meilleure des autres bornes évoquées dans le texte.

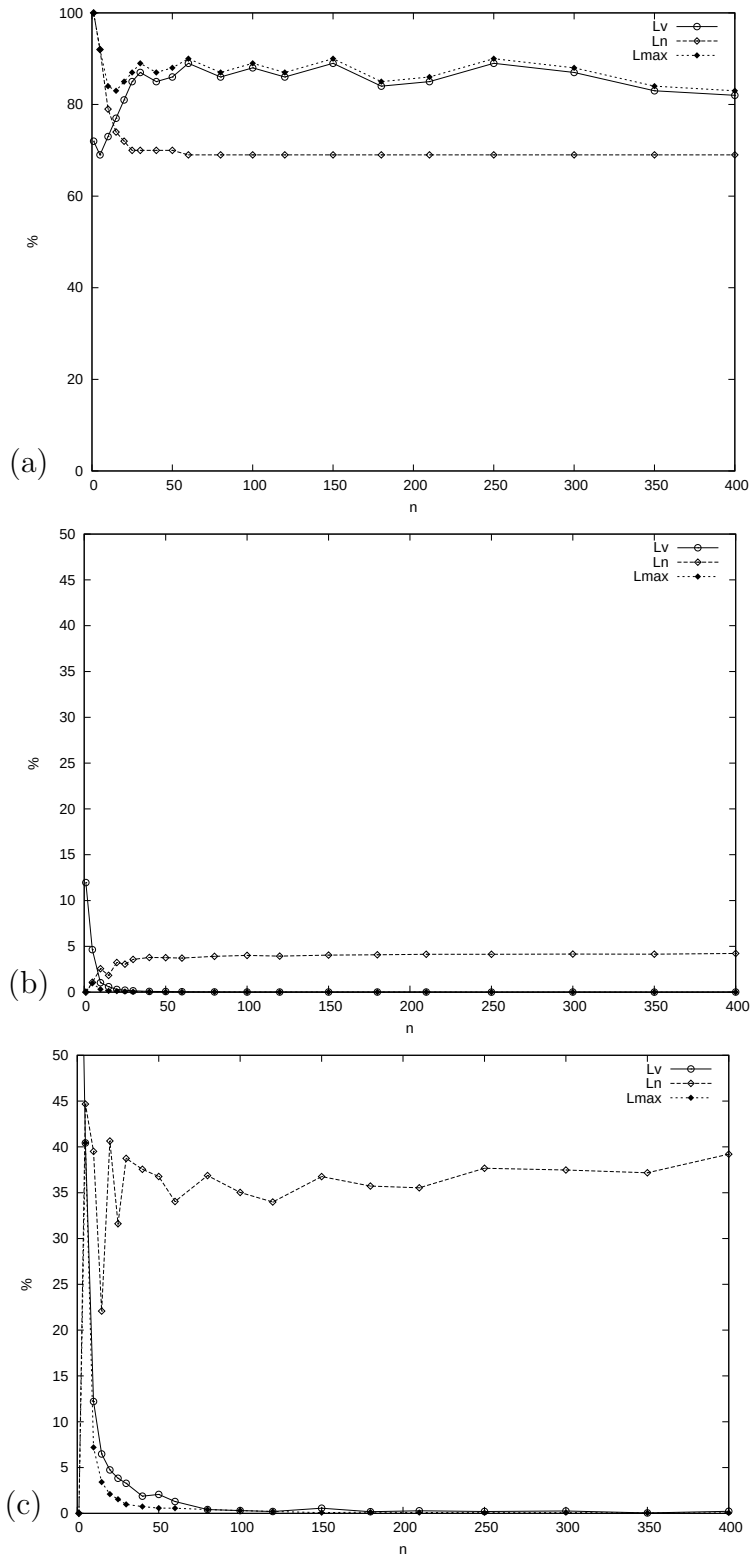


Figure 11.1 – Performances des bornes L_V, $L_{n_{\min}}$ et $L_{\max}$ en fonction du nombre d'objets. (a) Pourcentage de valeurs optimales ; écarts (b) moyen et (c) maximum de l'optimum (moyennes pour 100 instances, « l'optimum » étant en fait la meilleure valeur connue).

La borne L_V est très polyvalente : elle donne de très bons résultats sur n'importe quel type d'instances. Ses performances les plus faibles sont, comme prévu, les petites instances avec de grands objets (plus particulièrement quand les petits objets sont eux aussi grands).

À noter que l'amélioration $L_+^{L_V}$, proposée au chapitre 7, section 7.3, n'a permis de gain sur aucune instance.

La borne $L_{n_{\min}}$ fait mieux que ce que l'on pouvait espérer d'elle. Non seulement elle s'avère effectivement complémentaire de L_V, résolvant parfaitement les instances les plus problématiques pour L_V et permettant ainsi d'augmenter significativement les performances par le biais de la borne $L_{\max}$, mais elle se montre aussi relativement résistante face à de grandes instances. Alors qu'elle a été développée pour de petites instances, ses performances sont tout à fait correctes même pour un grand nombre d'objets, bien qu'elle soit alors assez nettement dominée par L_V. Son principal défaut est une grosse perte de contrôle : sur certaines instances l'erreur dépasse les 50% ! et le phénomène ne diminue pas avec le nombre d'objets (voir figure 11.1(c)).

La version utilisée de $L_{n_{\min}}$ (algorithme 7.1) apporte un gain réel par rapport à la version basique (proposition 7.4). Ainsi, la suppression des petits objets « inutiles » pour le calcul de la borne permet d'améliorer de 5% à 15% des valeurs, selon la taille de l'instance (le nombre d'améliorations grandissant avec le nombre d'objets). Ceci est particulièrement intéressant sur certaines des petites instances (2% des instances avec moins de 25 objets) puisque c'est cela qui permet d'obtenir la meilleure valeur (toutes bornes confondues), généralement optimale. Pour les grandes instances, toutefois, le gain n'est jamais suffisant pour rivaliser avec L_V.

Pour de futures améliorations de cette borne, il est bon de savoir que les instances les plus problématiques pour $L_{n_{\min}}$ sont celles de type « objets complémentaires ». Ce sont pour ces cas que les horribles déviations de plus de 40% sont enregistrées.

Les autres bornes ont été essayées sans *a priori* sur leurs éventuelles qualités. Il ressort qu'elles n'apportent rien.

Sur absolument toutes les instances des tests, L_x est dominée par $L_{\max}$. Sa seule qualité serait, par rapport à $L_{n_{\min}}$, d'avoir des garanties sur les valeurs données qui la rendent asymptotiquement optimale. Une caractéristique que l'on retrouve sur les grandes instances, avec de bien meilleurs écarts avec l'optimum, tant maximum que moyen, en faveur de L_x par rapport à $L_{n_{\min}}$. L_x est cependant très loin, notamment sur ce point, d'être compétitive avec L_V.

11.3 Les heuristiques

Venons-en maintenant à l'analyse des heuristiques pour $B|size_j|H_{\max}$.

11.3.1 Heuristiques testées

Les différentes heuristiques du chapitre 8 peuvent chacune se décliner en plusieurs versions, notamment en fonction d'un éventuel ordre imposé aux objets. Les

variantes des heuristiques retenues sont ainsi :

- AMP (algorithme de meilleur placement) tel que décrit par l'algorithme 8.1.
- AMPD (AMP décroissant) : AMP appliqué sur les objets triés par hauteurs décroissantes (voir section 8.2.3).
- ADR (algorithme de diviser pour régner) tel que décrit par l'algorithme 8.5.
- APBS (algorithme de placement dans les boîtes suivantes) tel que décrit par l'algorithme 8.2.
- APBSD (APBS décroissant) : APBS appliqué sur les objets triés par hauteurs décroissantes (voir section 8.3).
- APOM (algorithme de placement optimiste modifié) tel que décrit par l'algorithme 8.4.
- APOMD (APOM décroissant) : APOM appliqué sur les objets triés par largeurs décroissantes, puis hauteurs décroissantes (voir section 8.4).
- APOM-V et APOMD-V : variantes de APOM et APOMD pour lesquelles la valeur de référence augmente lorsqu'un objet la dépasse (voir section 8.4).
- APOM-X et APOMD-X : variantes de APOM et APOMD pour lesquelles toutes les valeurs possibles pour la valeur de référence sont utilisées (voir section 8.4).

Quelques autres versions ont aussi été essayées, mais n'ont pas été retenues. Ainsi, pour AMPD, la variante consistant à trier, pour une même hauteur, les objets par largeurs décroissantes n'apporte aucun gain. Pour APBS, trier les objets par largeurs décroissantes détériore l'algorithme. Pour les trois procédures APOMD, APOMD-V et APOMD-X, ne pas trier, pour une même largeur, par hauteurs décroissantes permet quelques très rares améliorations ; toutefois, la version choisie est très généralement préférable.

Aucune version « décroissante » de ADR n'a été retenue. Cela vient notamment de la sélection des solutions partielles à fusionner, pour laquelle un tri selon les hauteurs ou les largeurs n'a pas vraiment de sens. Une implémentation plus proche de la « méthode par différences » pourrait, par contre, tirer profit de certains ordres. Je reviendrai sur ce point dans la suite. Pour l'instant, il faut déjà noter deux variantes de ADR : Alea-ADR-50 et Alea-ADR-200, qui renvoient la meilleure de 50 ou 200, respectivement, solutions données par ADR pour autant d'ordres aléatoires des objets.

Similairement, Alea-AMP-50 et Alea-AMP-200 renvoient la meilleure de 50 ou 200, respectivement, solutions données par AMP.

11.3.2 Résultats

Les résultats sont résumés dans les tables 11.2 à 11.5. La discussion qui suit apporte les indispensables et nombreux compléments et précisions.

Commençons par ce qui, a première vue, est le plus surprenant : l'extraordinairement bonne qualité de APBS et APBSD, qui résolvent optimalement deux instances sur trois ! En fait, cela ne doit pas être attribué aux mérites de ces heuristiques, mais au fait que beaucoup d'instances (les deux tiers, donc), sont en fait très simples. Pour

Table 11.2 – Comparaison des heuristiques pour $B|size_j|H_{\max}$ (domination).
Pourcentage des cas où une heuristique est la meilleure parmi les heuristiques testées.

type - n	AMP	AMPD	ADR	APBS	APBSD	APOM	APOMD	APOM-X	APOMD-X
U - 5..25	78,54	92,20	85,37	76,10	76,10	72,68	73,66	73,17	79,02
G - 5..25	79,72	93,55	83,41	75,12	75,58	73,73	75,12	74,65	77,88
X - 5..25	56,41	82,05	61,54	55,13	55,13	47,44	56,41	61,54	69,23
U - 25..60	71,71	90,73	80,00	69,27	69,27	66,83	67,32	66,83	68,29
G - 25..60	74,19	89,86	84,79	71,43	71,43	70,97	70,97	70,97	71,43
X - 25..60	66,67	75,64	70,51	60,26	60,26	47,44	66,67	47,44	75,64
U - 60..100	74,15	86,34	77,56	68,78	68,78	66,83	66,83	66,83	68,29
G - 60..100	75,58	86,64	87,10	70,97	70,97	70,97	70,51	70,97	70,51
X - 60..100	58,97	83,33	61,54	56,41	56,41	46,15	60,26	46,15	71,79
U - 100..500	71,71	85,85	82,44	68,78	68,78	66,83	66,83	66,83	67,32
G - 100..500	73,73	84,33	87,10	70,97	71,43	70,51	70,51	70,51	71,43
X - 100..500	66,67	84,62	74,36	65,38	65,38	46,15	65,38	46,15	70,51
(moyenne)	70,67	86,26	77,98	67,38	67,46	62,21	67,54	63,50	71,78

Table 11.3 – Comparaison des heuristiques pour $B|size_j|H_{\max}$ (optimalité).
Pourcentage des cas où une heuristique est optimale (la comparaison est faite avec la meilleure borne connue).

type - n	AMP	AMPD	ADR	APBS	APBSD	APOM	APOMD	APOM-X	APOMD-X
U - 5..25	76,59	78,05	77,56	75,61	75,61	72,68	73,66	73,17	76,10
G - 5..25	76,04	77,88	76,04	74,65	73,73	73,73	74,65	74,65	76,04
X - 5..25	53,85	56,41	53,85	53,85	53,85	47,44	53,85	51,28	55,13
U - 25..60	69,27	74,15	69,27	68,78	68,78	66,83	67,32	66,83	67,32
G - 25..60	71,43	73,73	71,89	71,43	71,43	70,97	70,97	70,97	70,97
X - 25..60	60,26	66,67	60,26	60,26	60,26	47,44	60,26	47,44	60,26
U - 60..100	68,78	73,17	69,27	68,78	68,78	66,83	66,83	66,83	67,32
G - 60..100	70,97	74,65	71,89	70,97	70,97	70,51	70,51	70,51	70,51
X - 60..100	56,41	71,79	58,97	56,41	56,41	46,15	56,41	46,15	57,69
U - 100..500	68,78	73,17	69,27	68,78	68,78	66,83	66,83	66,83	67,32
G - 100..500	70,97	78,34	74,65	70,97	70,97	70,51	70,51	70,51	71,43
X - 100..500	65,38	82,05	69,23	65,38	65,38	46,15	61,54	46,15	61,54
(moyenne)	67,39	73,34	68,51	67,16	67,08	62,17	66,11	62,61	66,80

Table 11.4 – Comparaison des heuristiques pour $B|size_j|H_{\max}$ (écarts moyens).
Écart moyen (%) par rapport à la meilleure borne connue.

type - n	AMP	AMPD	ADR	APBS	APBSD	APOM	APOMD	APOM-X	APOMD-X
U - 5..25	1,05	0,50	0,80	1,87	1,28	4,58	2,70	3,33	1,49
G - 5..25	0,81	0,40	0,66	1,83	1,10	4,22	2,30	3,19	1,75
X - 5..25	2,45	1,36	2,24	3,04	2,75	7,65	2,98	3,92	2,00
U - 25..60	0,52	0,22	0,31	1,30	0,67	4,82	2,67	4,17	2,14
G - 25..60	0,43	0,24	0,27	1,11	0,57	4,29	2,36	3,78	1,97
X - 25..60	0,97	0,58	0,81	1,80	1,21	7,75	1,52	5,79	1,22
U - 60..100	0,28	0,14	0,20	0,90	0,36	4,59	2,45	4,28	2,17
G - 60..100	0,26	0,15	0,18	0,85	0,32	4,26	2,24	3,98	2,02
X - 60..100	0,46	0,20	0,40	0,93	0,55	7,60	1,17	6,65	0,98
U - 100..500	0,09	0,05	0,05	0,46	0,12	4,63	2,36	4,51	2,27
G - 100..500	0,06	0,04	0,04	0,37	0,10	4,08	2,18	4,00	2,11
X - 100..500	0,11	0,04	0,06	0,36	0,11	7,48	0,96	7,20	0,90
(moyenne)	0,62	0,33	0,50	1,23	0,76	5,50	2,16	4,57	1,75

Table 11.5 – Comparaison des heuristiques pour $B|size_j|H_{\max}$ (écarts maximums).

Écart maximum (%) par rapport à la meilleure borne connue.

type - n	AMP	AMPD	ADR	APBS	APBSD	APOM	APOMD	APOM-X	APOMD-X
U - 5..25	16,09	6,93	15,48	23,40	16,83	39,22	19,72	25,00	19,72
G - 5..25	13,01	5,88	10,57	48,36	13,11	31,07	20,00	24,77	15,70
X - 5..25	12,73	7,56	12,73	16,81	13,45	33,33	16,86	19,92	11,90
U - 25..60	6,35	4,02	3,88	12,69	9,52	23,55	24,03	21,66	15,16
G - 25..60	7,98	2,56	3,17	23,31	6,71	27,38	15,49	23,02	14,01
X - 25..60	8,31	5,34	8,31	14,53	8,31	21,60	16,02	18,52	12,73
U - 60..100	3,43	2,00	3,86	16,35	4,15	22,65	20,82	21,25	18,24
G - 60..100	4,57	1,63	3,71	28,29	4,00	22,14	15,40	21,47	14,28
X - 60..100	2,83	1,66	2,83	6,51	2,83	20,90	16,14	18,39	15,08
U - 100..500	1,57	0,77	1,12	9,21	2,47	21,26	21,06	20,67	20,64
G - 100..500	0,88	0,74	0,57	12,16	1,31	22,81	15,95	22,52	15,47
X - 100..500	0,74	0,61	0,61	3,71	0,82	20,69	18,53	20,11	18,12
(moyenne)	6,54	3,31	5,57	17,95	6,96	25,55	18,33	21,44	15,92

toutes les instances où APBS et APBSD donnent le meilleur résultat, qui est alors en général aussi l'optimum, les autres heuristiques font aussi bien (seules APOM, APOM-V et APOM-X font significativement souvent moins bien). En fait, il n'y a qu'une seule instance pour laquelle seule AMPD fait aussi bien que APBS et APBSD.

Lorsque l'on considère, en plus, les écarts à l'optimum, on constate qu'ils sont moins bons que pour d'autres heuristiques, même si la version APBSD se défend bien. En définitive, en regardant plus en détails, on arrive à la conclusion sans surprise que APBS, et même APBSD, n'apportent rien : AMPD et ADR, ou AMP si l'on veut résoudre en-ligne, seront toujours préférables.

Une heuristique très décevante est APOM. La version en-ligne n'arrive même pas à résoudre optimalement toutes les instances très simples, et il est excessivement rare qu'elle résolve une instance mieux que la plupart des autres méthodes. La variante APOM-V ne change pas le comportement général, et a même plutôt tendance à détériorer les performances (du fait de leur similitude, les résultats pour APOM-V et APOMD-V ne sont pas reportés dans les tables).

Trier les objets par largeurs puis hauteurs décroissantes améliore ces deux procédures, mais cela ne suffit pas à les rendre attractives par rapport aux autres méthodes.

Les versions APOM-X et APOMD-X sont, par définition, meilleures que APOM et APOMD. En moyenne, leur comportement est tout à fait honnête, bien qu'elles résolvent relativement mal certaines des instances très simples. En fait, elles restent des méthodes dangereuses, dans le sens où leurs erreurs peuvent être importantes.

Elles possèdent toutefois une propriété très intéressante : pour beaucoup des instances du groupe X où AMPD n'est pas le meilleur algorithme, c'est APOM-X et encore plus souvent APOMD-X qui lui ravissent le titre. Ainsi, leur comportement est très bon sur les instances à « objets complémentaires », qui posent problème à AMPD.

De manière générale, AMPD domine nettement les autres heuristiques, sur tous les critères et pour tous les types d'instances. Elle permet ainsi une excellente résolution de toutes les instances, optimale près de 3 fois sur 4, et avec un très bon contrôle : les erreurs, moyenne comme maximale, sont faibles.

Si les objets doivent être rangés en-ligne, et donc qu'aucun tri n'est possible, le prix à payer est significatif : AMP a des performances moindres que AMPD, mais tout de même nettement supérieures que les autres algorithmes en-ligne.

ADR est incontestablement la meilleure heuristique après AMPD. Sur les instances des groupes U et G, lorsque cette dernière est battue, c'est généralement par ADR.

11.3.3 Ordres aléatoires

Venons-en maintenant à Alea-AMP et Alea-ADR. Tester ces approches est une démarche naturelle. Pour AMP, c'est en raison de la conjecture 8.1, qui affirme

qu'il existe toujours un ordre des objets pour lequel la solution donnée par AMP est optimale. Pour ADR, de nombreux critères différents peuvent être choisis pour sélectionner les solutions partielles à fusionner, et il a été mentionné que certains choix pourraient mener à de meilleures garanties que celles obtenues (voir question 8.3).

Les résultats sont résumés dans les tables 11.6 à 11.9.

Les gains apportés par Alea-AMP-50 et Alea-AMP-200 sont décevants. Bien souvent, l'ordre utilisé par AMPD est tout simplement le meilleur ordre considéré... ainsi, AMPD est aussi bon que Alea-AMP-50 ! Augmenter le nombre d'essais à 200 apporte un meilleur contrôle des solutions, par rapport à AMPD, mais échoue plus souvent à trouver l'optimum.

Si les résultats sont médiocres du point de vue de Alea-AMP, ils permettent toutefois de confirmer que l'ordre utilisé par AMPD est véritablement judicieux, à la fois performant, robuste, et polyvalent.

Beaucoup plus appréciables sont les performances de Alea-ADR. Les gains par rapport à ADR sont très conséquents, et Alea-ADR-50 est meilleur à la fois que AMPD et Alea-AMP-200. Avec près de 80% d'instances résolues optimalement, Alea-ADR-200 a une réussite impressionnante, pour des temps de calculs, certes 200 fois plus élevés que pour une simple exécution, mais toujours très rapides !

Ces excellentes performances de Alea-ADR soulignent l'intérêt de chercher une implémentation particulière de ADR, avec un choix judicieux dans la sélection des solutions partielles à fusionner.

Dans les deux cas, rechercher un bon ordre pour les objets se révèle efficace ; aussi peut-on déjà s'attendre à de bons résultats lorsque cette recherche est menée de manière astucieuse, en particulier pour l'algorithme génétique (une comparaison entre la recherche aléatoire d'un bon ordre et l'algorithme génétique est effectuée dans la section suivante).

11.3.4 Remarques conclusives

Pour résumer l'analyse expérimentale de ces heuristiques, il faut tout d'abord noter que l'heuristique AMPD est la meilleure, mais que ADR est déjà très bonne et que des améliorations prometteuses doivent pouvoir lui être faites.

Je veux ensuite insister sur la complémentarité du trio AMPD, ADR et APOMD-X : sur quasiment toutes les instances, l'une au moins de ces trois heuristiques domine toutes les autres et leur conjonction assure ainsi d'excellents résultats.

Au niveau des instances, la plupart, et principalement celles des ensembles U et G, sont faciles à résoudre optimalement. De manière générale, les plus problématiques sont celles à « objets complémentaires », ainsi que celles pour lesquelles la plus petite hauteur est grande et proche de la plus haute.

Table 11.6 – Tests de AMP et ADR sur des ordres aléatoires (domination).
Pourcentage des cas où une heuristique est la meilleure parmi les heuristiques testées.

type - n	AMP	AMPD	AMP-50	AMP-200	ADR	ADR-50	ADR-200
U - 5..25	77,07	85,37	85,37	92,2	81,95	91,22	97,56
G - 5..25	76,5	82,95	86,18	90,32	77,88	92,63	96,31
X - 5..25	55,13	64,1	74,36	80,77	57,69	76,92	96,15
U - 25..60	70,24	80,00	71,71	75,61	70,24	86,83	94,15
G - 25..60	71,43	76,96	74,19	78,80	73,27	84,79	98,16
X - 25..60	60,26	70,51	62,82	69,23	61,54	75,64	93,59
U - 60..100	68,78	76,10	70,73	76,10	69,27	83,90	93,66
G - 60..100	70,97	76,96	74,65	77,88	71,89	82,49	94,01
X - 60..100	56,41	75,64	61,54	66,67	58,97	79,49	92,31
U - 100..500	68,78	74,15	70,24	72,68	69,27	89,76	97,56
G - 100..500	70,97	78,34	72,35	76,04	74,65	89,86	95,39
X - 100..500	65,38	82,05	69,23	71,79	71,79	85,90	94,87
(moyenne)	67,66	76,93	72,78	77,34	69,87	84,95	95,31

Table 11.7 – Tests de AMP et ADR sur des ordres aléatoires (optimalité).
Pourcentage des cas où une heuristique est optimale
(la comparaison est faite avec la meilleure borne connue).

type - n	AMP	AMPD	AMP-50	AMP-200	ADR	ADR-50	ADR-200
U - 5..25	76,59	78,05	79,02	80,98	77,56	79,02	81,46
G - 5..25	76,04	77,88	78,80	79,26	76,04	79,72	80,18
X - 5..25	53,85	56,41	62,82	62,82	53,85	62,82	67,95
U - 25..60	69,27	74,15	69,27	69,27	69,27	73,66	75,12
G - 25..60	71,43	73,73	71,89	73,27	71,89	76,04	79,26
X - 25..60	60,26	66,67	61,54	61,54	60,26	64,10	70,51
U - 60..100	68,78	73,17	69,76	71,71	69,27	75,61	81,46
G - 60..100	70,97	74,65	72,35	72,81	71,89	74,65	79,72
X - 60..100	56,41	71,79	60,26	61,54	58,97	71,79	79,49
U - 100..500	68,78	73,17	70,24	69,27	69,27	84,39	88,29
G - 100..500	70,97	78,34	71,89	72,81	74,65	85,25	88,48
X - 100..500	65,38	82,05	67,95	67,95	69,23	79,49	85,90
(moyenne)	67,39	73,34	69,65	70,27	68,51	75,55	79,82

Table 11.8 – Tests de AMP et ADR sur des ordres aléatoires (écarts moyens).
Écart moyen (%) par rapport à la meilleure borne connue.

type - n	AMP	AMPD	AMP-50	AMP-200	ADR	ADR-50	ADR-200
U - 5..25	1,05	0,50	0,35	0,26	0,80	0,26	0,23
G - 5..25	0,81	0,40	0,31	0,24	0,66	0,26	0,23
X - 5..25	2,45	1,36	0,62	0,52	2,24	0,52	0,42
U - 25..60	0,52	0,22	0,20	0,16	0,31	0,11	0,09
G - 25..60	0,43	0,24	0,19	0,15	0,27	0,11	0,09
X - 25..60	0,97	0,58	0,31	0,23	0,81	0,20	0,14
U - 60..100	0,28	0,14	0,11	0,08	0,20	0,06	0,05
G - 60..100	0,26	0,15	0,11	0,08	0,18	0,07	0,06
X - 60..100	0,46	0,20	0,15	0,11	0,40	0,06	0,05
U - 100..500	0,09	0,05	0,03	0,02	0,05	0,01	0,01
G - 100..500	0,06	0,04	0,03	0,02	0,04	0,01	0,01
X - 100..500	0,11	0,04	0,04	0,03	0,06	0,02	0,01
(moyenne)	0,62	0,33	0,21	0,16	0,50	0,14	0,12

Table 11.9 – Tests de AMP et ADR sur des ordres aléatoires (écarts maximums).
Écart maximum (%) par rapport à la meilleure borne connue.

type - n	AMP	AMPD	AMP-50	AMP-200	ADR	ADR-50	ADR-200
U - 5..25	16,09	6,93	4,76	4,64	15,48	4,64	4,64
G - 5..25	13,01	5,88	4,92	3,00	10,57	3,00	3,00
X - 5..25	12,73	7,56	4,37	3,57	12,73	4,37	3,97
U - 25..60	6,35	4,02	2,09	1,66	3,88	1,45	1,94
G - 25..60	7,98	2,56	2,10	1,46	3,17	1,67	1,46
X - 25..60	8,31	5,34	1,78	1,78	8,31	1,78	1,60
U - 60..100	3,43	2,00	1,31	1,06	3,86	1,72	1,57
G - 60..100	4,57	1,63	0,98	0,98	3,71	1,14	0,98
X - 60..100	2,83	1,66	1,05	1,05	2,83	1,12	1,05
U - 100..500	1,57	0,77	0,35	0,23	1,12	0,24	0,21
G - 100..500	0,88	0,74	0,45	0,38	0,57	0,50	0,45
X - 100..500	0,74	0,61	0,37	0,31	0,61	0,44	0,42
(moyenne)	6,54	3,31	2,04	1,68	5,57	1,84	1,77

11.4 Apport de l'algorithme génétique

Pour tester l'apport de l'algorithme génétique, je me suis basé sur le *gain potentiel* que je définis, pour une instance donnée, comme la différence entre la meilleure valeur donnée par l'une des heuristiques du chapitre 8, et la meilleure valeur donnée par l'une des bornes inférieures du chapitre 7[1]. L'idée est de tester quel pourcentage de ce gain l'algorithme génétique arrive effectivement à gagner.

J'ai sélectionné pour chaque catégorie d'instances (*i.e.* pour chacun des trois types U, G, X et chacune des tailles 0..25, 25..60, 60..100, 100..500) les trois instances les moins bien résolues, c'est-à-dire pour lesquelles le gain potentiel est le plus grand (en cas d'égalité, j'ai choisi l'instance de plus petite valeur, le gain potentiel étant alors, relativement, plus élevé).

L'évaluation des algorithmes est faite en moyenne, pour 5 exécutions pour chacune des 36 instances. Dans la suite, Gen-AMP désigne l'algorithme génétique basé sur AMP, Gen-ADR celui basé sur ADR. Les paramètres utilisés sont ceux déduits au chapitre précédent[2].

Les résultats sont résumés dans les tableaux 11.10 et 11.11, ainsi que par les figures 11.2 à 11.5.

La première bonne nouvelle, pour ce qui est de l'intérêt de l'algorithme génétique, est que parmi les 36 instances, 15, principalement du type X, ont été résolues optimalement pour au moins l'une des exécutions (5 fois par Gen-AMP, 3 fois par Gen-ADR et 7 fois par les deux). Ces proportions ne sont certes pas impressionnantes, mais il faut se rappeler qu'en guise de valeurs optimales, on ne dispose en fait que de bornes inférieures.

Ainsi, toutes les mesures sont très pessimistes. Ceci est particulièrement important pour les petites instances. L'allure très cassée des courbes pour des instances de taille 5..25 et leur convergence vers une même valeur pour Gen-AMP et Gen-ADR (figure 11.4) me font attribuer l'écart de 30% restant à combler non pas à l'algorithme génétique, mais aux bornes inférieures.

De manière générale, les deux versions de l'algorithme génétique permettent des gains importants, très rapidement (tables 11.10 et 11.11, figure 11.2). Ces gains sont très nettement supérieurs à ceux d'une recherche hasardeuse, et ce dès les premières générations (figure 11.3). L'approche est donc adaptée lorsqu'on veut consacrer du temps pour trouver une excellente solution, mais aussi lorsqu'on désire améliorer significativement une solution en peu de temps.

Pour ces 36 instances « difficiles », il ne semble pas y avoir de véritable corrélation entre la taille (n) et la difficulté. Le type des objets est par contre essentiel : l'algorithme est doué pour le type X mais a des difficultés avec le type gaussien. Il faut d'ailleurs remarquer que Gen-ADR est plus sensible au type, avec des écarts plus grands, tandis que Gen-AMP est plus robuste.

[1] Ce gain potentiel n'est donc, en fait, qu'une borne supérieure du gain réellement possible.

[2] Pour repréciser ces paramètres : une population de 35 individus ; un taux de mutation de 0,6 ; un taux de croisement de 0,5 ; une adaptation linéaire, avec pour maximum 100 et comme facteur de décroissance 36 ; un remplacement total de la population, sans élitisme.

Table 11.10 – Gains apportés par Gen-AMP selon le nombre de générations. Pourcentages du gain potentiel.

type - n	**générations (individus)**						
	5 (175)	10 (350)	50 (1750)	100 (3500)	250 (8750)	500 (17500)	1000 (35000)
U - 5..25	65,33	69,66	74,00	75,00	77,33	79,33	80,00
G - 5..25	42,00	44,66	51,33	54,66	55,33	57,00	57,33
X - 5..25	59,66	66,00	74,33	76,00	79,66	81,00	82,00
U - 25..60	30,00	37,33	51,00	54,66	58,00	60,66	61,66
G - 25..60	36,33	43,33	56,00	57,33	59,66	60,00	60,66
X - 25..60	50,00	56,00	72,00	75,00	78,33	80,00	81,33
U - 60..100	42,33	50,33	72,00	77,66	81,00	83,66	86,00
G - 60..100	46,33	55,00	77,33	82,00	86,66	89,00	90,00
X - 60..100	45,66	52,33	68,00	73,33	79,66	81,33	82,66
U - 100..500	52,00	60,33	79,66	84,33	91,00	93,66	95,66
G - 100..500	47,00	56,66	79,66	87,00	91,00	93,33	94,00
X - 100..500	67,33	72,00	81,00	84,66	86,00	88,33	89,33

Table 11.11 – Gains apportés par Gen-ADR selon le nombre de générations. Pourcentages du gain potentiel.

type - n	**générations (individus)**						
	5 (175)	10 (350)	50 (1750)	100 (3500)	250 (8750)	500 (17500)	1000 (35000)
U - 5..25	68,33	72,00	77,33	78,33	78,66	80,33	80,33
G - 5..25	45,33	51,00	53,66	55,00	56,00	56,33	56,66
X - 5..25	57,33	65,00	72,33	73,33	75,33	77,00	79,00
U - 25..60	21,66	26,66	35,33	41,33	44,00	46,66	49,33
G - 25..60	29,33	32,00	39,00	43,33	45,00	47,66	48,33
X - 25..60	57,33	61,66	70,66	73,33	75,00	76,33	76,66
U - 60..100	28,33	32,00	44,00	50,33	54,66	57,33	60,66
G - 60..100	22,33	28,00	48,00	56,00	65,33	70,00	73,66
X - 60..100	43,33	46,33	55,66	60,00	63,00	67,00	69,66
U - 100..500	45,00	52,00	77,66	83,33	88,33	91,33	94,00
G - 100..500	12,00	18,00	38,33	51,66	61,33	66,66	73,66
X - 100..500	71,33	74,66	83,66	88,00	92,00	96,33	98,00

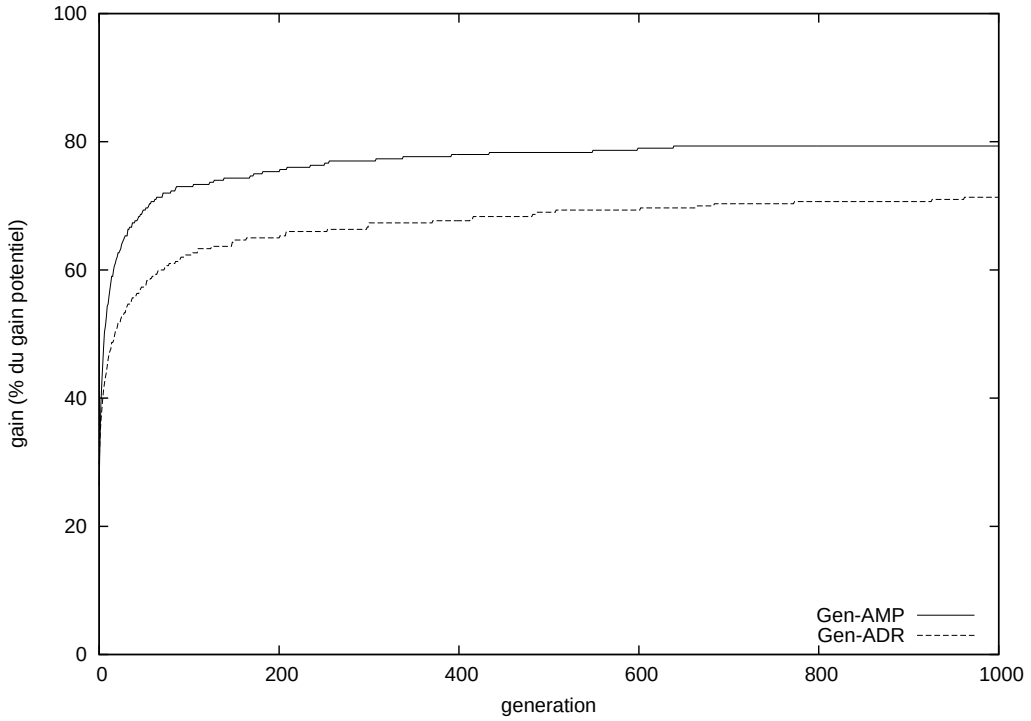


Figure 11.2 – Gains apportés par Gen-AMP et Gen-ADR en fonction du nombre de générations.

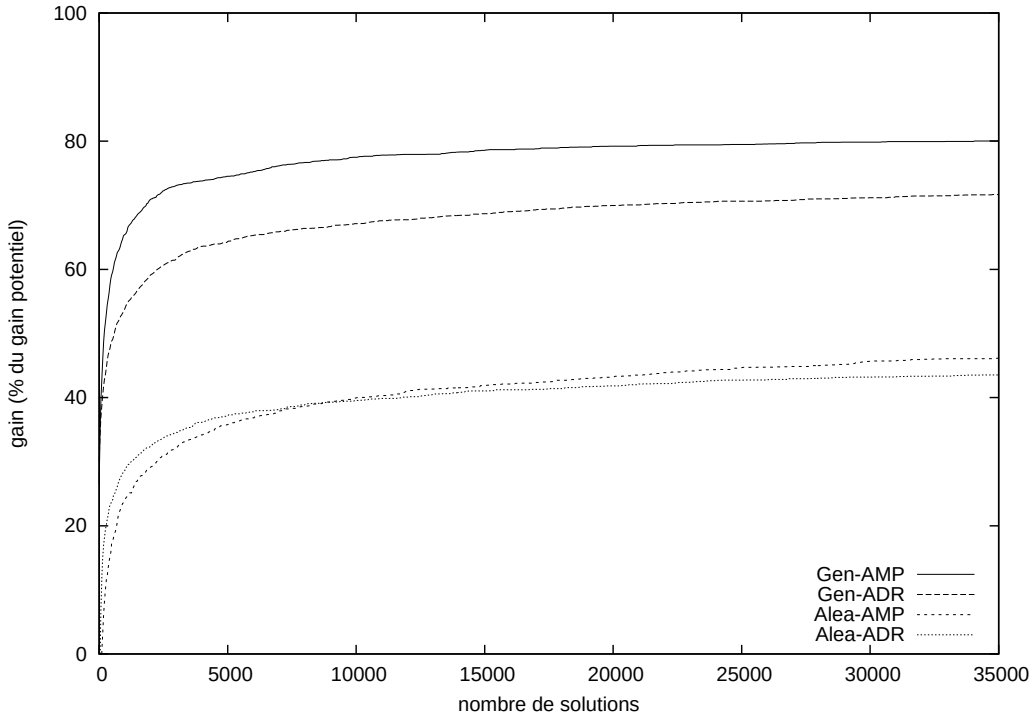
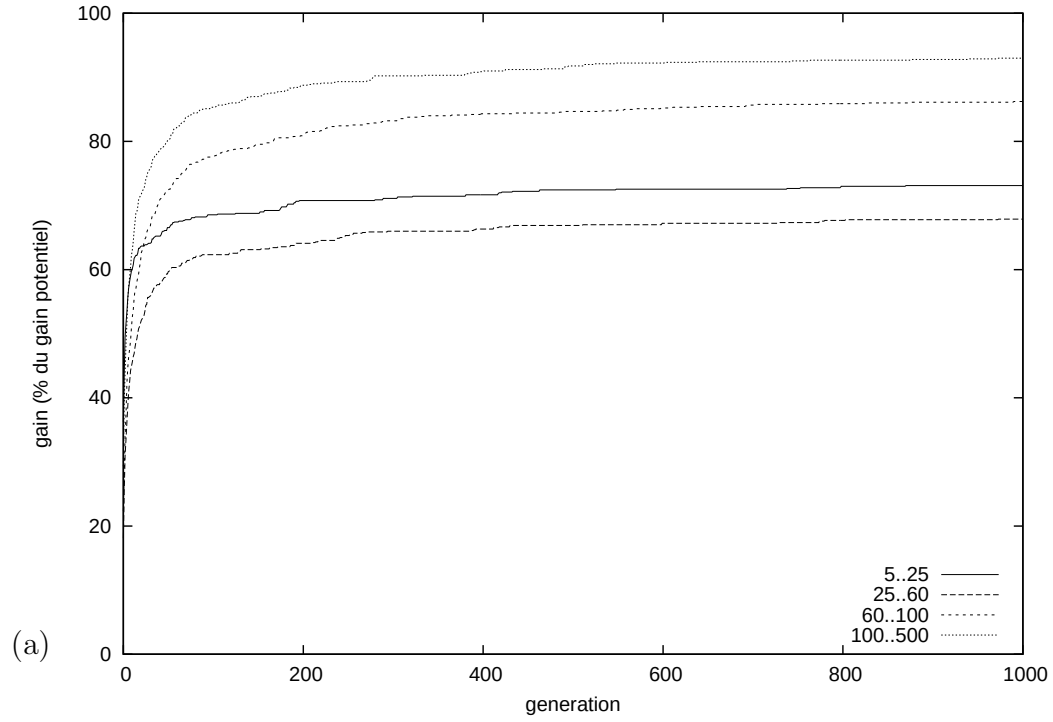


Figure 11.3 – Gains apportés par Gen-AMP et Gen-ADR par rapport à Alea-AMP et Alea-ADR en fonction du nombre de solutions générées.



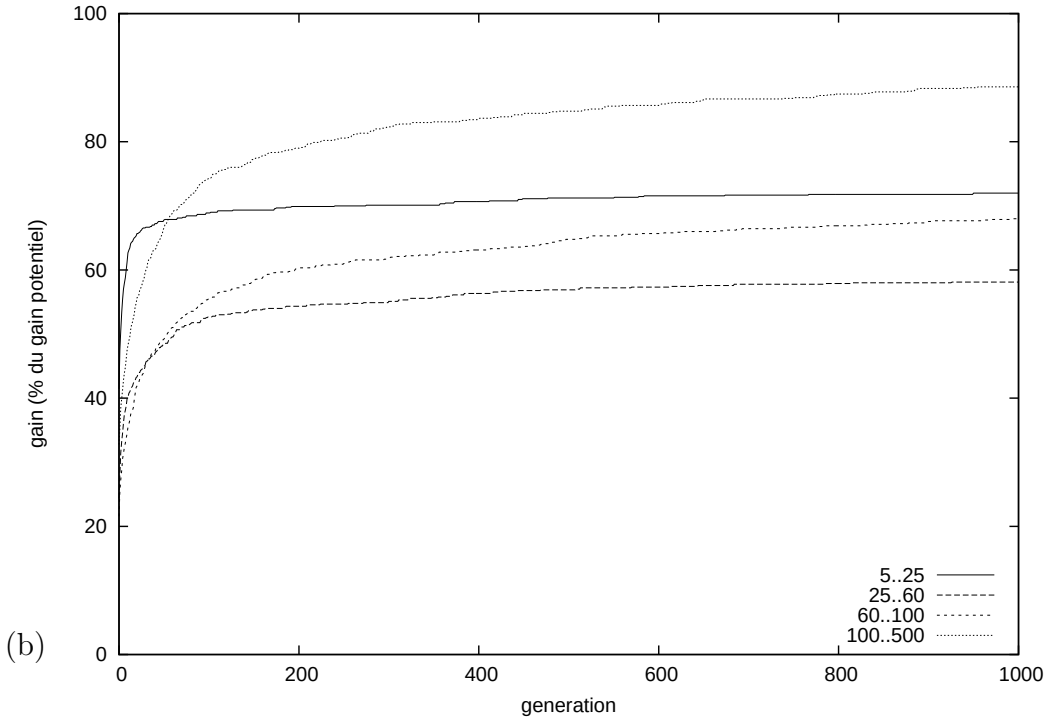
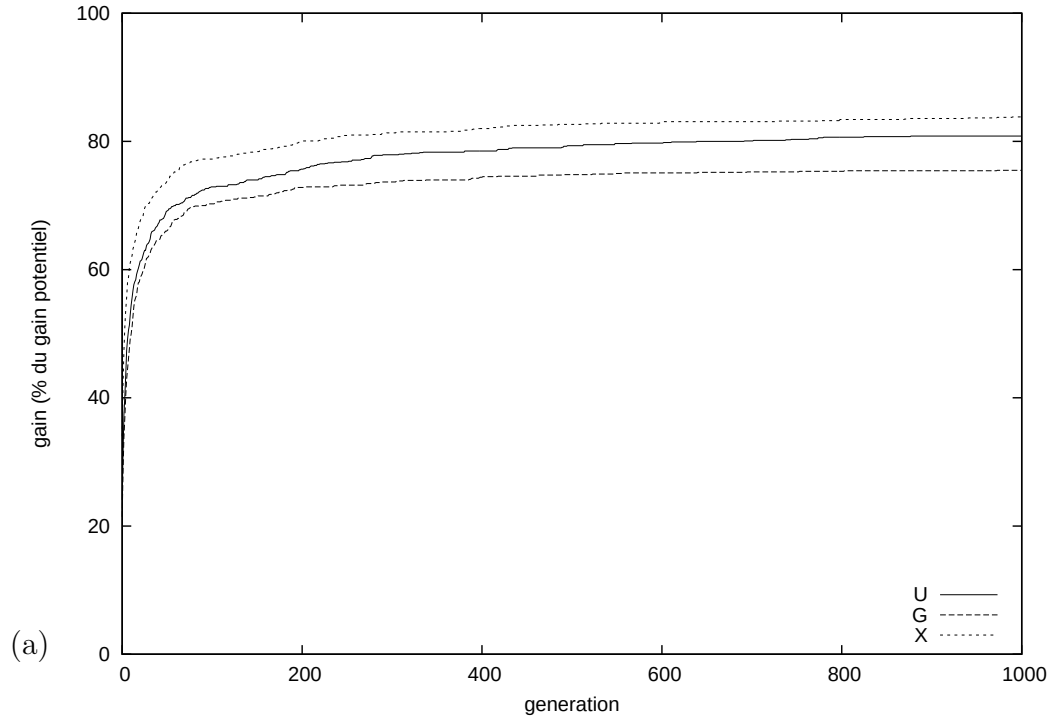
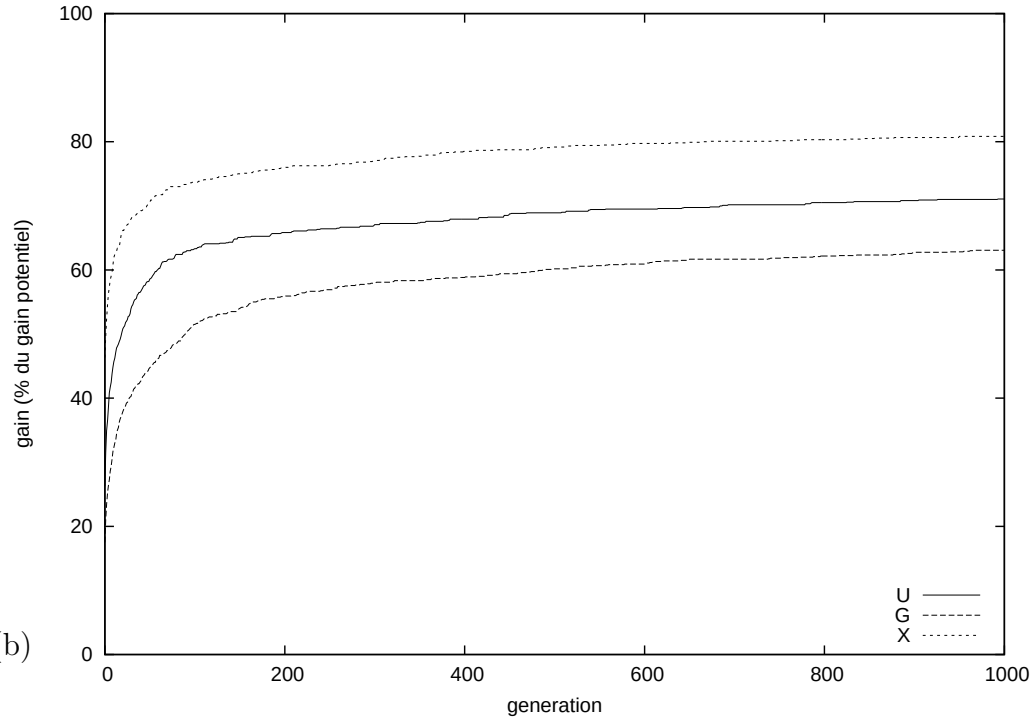


Figure 11.4 – Gains apportés par Gen-AMP et Gen-ADR en fonction du nombre de générations et selon la taille des instances.
(a) Gen-AMP ; (b) Gen-ADR.



(a)



(b)

Figure 11.5 – Gains apportés par Gen-AMP et Gen-ADR en fonction du nombre de générations et selon le type des instances.
(a) Gen-AMP ; (b) Gen-ADR.

En définitive, l'algorithme génétique remplit bien son rôle. La version Gen-AMP est particulièrement polyvalente et permet, quel que soit le type d'instance ou sa taille, des gains importants rapidement. Gen-ADR a des performances moindres mais cependant tout à fait honorables. L'une des raisons pour lesquelles Gen-ADR est moins bon est sans doute que les opérateurs de mutation et de croisement, qui affectent localement l'ordre des objets, sont moins adaptés à cet algorithme.

11.5 Où il reste du travail

La grande majorité des instances est résolue optimalement par les heuristiques ou par l'algorithme génétique. Il reste malheureusement souvent une difficulté : bien que des solutions améliorées soient trouvées, il n'est pas aisé de prouver leur optimalité. C'est donc avant tout sur les bornes inférieures que le travail futur devrait se concentrer.

En ce qui concerne les instances problématiques, pour les bornes comme pour les heuristiques, ce sont les instances avec une plus petite hauteur d'objet $h_{\min}$ grande et relativement peu différente de $h_{\max}$ qui posent le plus de problèmes et qui mériteraient un effort particulier.

Pour finir ?

On ne termine pas un poème, on l'abandonne.
— *Paul Valéry*

Prologue à une suite

Toute œuvre scientifique « achevée » n'a d'autre sens
que celui de faire naître de nouvelles « questions » :
elle demande donc à être « dépassée » et à vieillir.
Dans les sciences, non seulement notre destin,
mais encore notre but à tous
est de nous voir un jour dépassés.
— *Max Webber*

Voici venu le moment de mettre un terme à ce document. Mais pas à ce travail, inachevé comme il se doit. Conclusions, oui ; mais avant tout perspectives. Le champ ouvert par le concept d'« objet multiboîte » est énorme ; il étend plusieurs domaines intensément étudiés depuis plus de trente ans. Autant de cas à considérer, à adapter...

Le concept d'« objet multiboîte » a été clairement posé et placé dans son contexte, c'est-à-dire par rapport aux nombreux problèmes de la littérature dont il se rapproche, mais aussi vis-à-vis des nombreuses applications potentielles dont il peut rendre compte (chapitre 1).

C'est avant tout un problème de partition d'un ensemble (les objets) en sous-ensembles (le contenu des boîtes). Comme tout problème de rangement, il est très proche des problèmes d'ordonnancements, en particulier des ordonnancements de tâches multiprocesseurs, dont la classification a même été reprise pour construire une taxinomie permettant de baliser l'immense ensemble de problèmes que l'on peut réunir sous la terminologie de « rangements multiboîtes » (chapitre 1, section 1.2).

Par rapport aux rangements « classiques » (p.ex. le Bin-Packing), la principale caractéristique des rangements multiboîtes est que les objets sont composés de plusieurs bouts distincts devant être placés dans des boîtes différentes. Cette généralisation n'est absolument pas identifiable à une augmentation de la dimension (p.ex. le Bin-Packing à deux dimensions, où boîtes et objets sont des rectangles et non plus des segments) : les différents bouts d'un même objet n'ont pas à être rangés à la même hauteur dans les différentes boîtes ; ils n'ont même pas, non plus, à être rangés dans des boîtes adjacentes.

De manière générale, les problèmes de rangement d'objets multiboîtes sont $\mathcal{NP}$-difficiles, au sens fort si le nombre de boîtes m est arbitraire (chapitre 3). Quelques cas particuliers, toutefois, peuvent être résolus en temps polynomial. C'est notamment le cas pour des objets de hauteur unitaire et trois objectifs très naturels :

minimiser le nombre de boîtes utilisées, maximiser le nombre d'objets rangés et minimiser la taille des boîtes (chapitre 3, section 3.3). Pour ce dernier cas, une résolution polynomiale est possible même lorsque des incompatibilités entre objets et boîtes sont considérées (chapitre 6).

Lorsque le nombre de boîtes est fixé, une résolution théoriquement assez efficace (pseudo-polynomiale) est possible lorsque l'on veut maximiser le nombre (pondéré) d'objets rangés ou minimiser la hauteur des boîtes (chapitre 5). Dans ce deuxième cas, une excellente approximation, un FPTAS, existe même (chapitre 5, section 5.4). Toutefois, ces bonnes nouvelles ne resteront qu'encre sur papier ; elles sont, comme souvent pour de telles approches, complètement inadaptées pour une résolution pratique et réelle dès qu'il y a plus de 3 ou 4 boîtes...

Cet écart entre les réalités théoriques, établies dans la première partie, et pratiques est partiellement comblé par la deuxième partie, dont la raison d'être est la résolution efficace du cas d'objets de largeur constante, avec pour objectif de minimiser la hauteur des boîtes. L'étude se concentre sur la recherche d'algorithmes rapides et aucune méthode exacte n'est proposée.

Essentiellement, deux bornes inférieures sont proposées, très complémentaires et basées sur deux principes différents (chapitre 7). La première n'envisage les objets qu'en terme de volume total, et est donc optimiste : elle donne la valeur « idéale », celle que l'on obtient s'il est possible de répartir équitablement les objets entre les boîtes. La deuxième borne, au contraire, se concentre sur la particularité essentielle des objets : leur largeur. Dans les deux cas, les résultats pratiques sont excellents sur la grande majorité des cas, mais des améliorations seraient nécessaires pour certaines instances n'ayant que de grands objets (chapitre 11, section 11.2).

À la suite de ces bornes inférieures, plusieurs heuristiques très rapides et avec des garanties de performances ont été proposées (chapitre 8). Ces différents algorithmes laissent très peu d'instances mal résolues, les divers principes mis en œuvre permettant une assez bonne complémentarité des méthodes. Parmi celles-ci l'approche du « meilleur placement », qui consiste à mettre un objet dans les boîtes les plus vides au moment où l'objet est considéré, donne les meilleurs résultats et est facilement déclinable en de nombreuses variations. En particulier elle s'adapte très bien au cas où certains objets ont une pré-affectation imposée ; elle permet ainsi l'élaboration d'un PTAS qui, bien que théoriquement moins bon que le FPTAS évoqué précédemment, est beaucoup plus prometteur (chapitre 9).

Pour terminer la résolution de $B|size_j|H_{\max}$ sans laisser de cas mal résolus, un algorithme génétique est proposé, en complément des heuristiques (chapitre 10). Cet algorithme, hybride, a un très bon comportement pour un assez large éventail de réglages de ses paramètres, ce qui en fait une méthode robuste. Il permet très rapidement des gains très significatifs, et si l'on dispose de plus de temps, atteint d'excellentes solutions, souvent optimales.

* * *

Le travail effectué permet déjà de bien appréhender les problèmes de rangements

multiboîtes et résout très bien théoriquement, voire pratiquement, plusieurs cas importants, essentiellement lorsque les objets sont de largeur constante et qu'il faut minimiser la hauteur des boîtes. Le chemin a cependant été jonché de nombreuses questions non résolues (rappelées dans le chapitre suivant), et plusieurs directions de recherche se dégagent.

Commençons par le cas le plus traité : $B|size_j|H_{\max}$. La plus grande lacune, à mon sens, dans la résolution de ce problème est l'absence totale de méthodes exactes. Le principal écueil pour en réaliser une est qu'il y a relativement peu de différence de valeur entre une solution optimale et n'importe quelle autre solution construite un tant soit peu intelligemment (typiquement, par l'une des heuristiques du chapitre 8). Il est donc difficile d'obtenir de bonnes bornes inférieures sur des solutions partielles qui permettraient, par exemple, d'effectuer une recherche arborescente de type « brancher, borner, couper ».

Aussi, le travail futur devrait se concentrer avant tout sur la recherche de telles bornes. L'utilisation de fonctions redondantes, esquissée au chapitre 7, section 7.3, devrait ainsi être développée. Je pense surtout qu'il faudrait intégrer au maximum la largeur des objets dans le calcul, et donc étendre l'approche du raisonnement énergétique (chapitre 7, section 7.2). Pour l'instant l'aspect « multiboîte » n'est encore exploité que de manière globale ; des considérations plus fines permettraient, par exemple, de conclure que certaines paires, certains triplets *etc.* d'objets sont obligatoirement présents ou absents de certaines boîtes.

Ensuite, toujours dans l'optique d'une résolution pratique de $B|size_j|H_{\max}$, plusieurs interrogations importantes restent en suspens quant aux diverses heuristiques proposées (chapitre 8), notamment la performance exacte de APOM (conjecture 8.2). De plus, l'utilisation de solutions particulières (typiquement celles « par blocs ») pourrait permettre, sous réserve d'un contrôle de la dégradation ainsi introduite, de proposer des méthodes beaucoup plus rapides que celles utilisées ici.

Mais c'est surtout ADR qui mériterait des développements. En s'inspirant davantage de la « méthode par différences », une implémentation avec de meilleures garanties est sûrement possible (voir question 8.3) ; une réalisation plus rapide serait aussi appréciable, notamment dans le cadre de l'algorithme génétique.

Pour l'ensemble des heuristiques, et même des bornes, des études probabilistes pourraient également être menées pour étudier le comportement en moyenne, et non pas seulement pour le pire cas ; des garanties pour des ratios de performances relatifs pourrait également être obtenues.

L'algorithme génétique mériterait, comme toujours pour ce type de méthode, d'être développé. La première chose à faire serait de comprendre véritablement pourquoi le taux de mutations doit être aussi anormalement élevé. Ensuite, un réglage plus fin des paramètres, en fonction de l'instance, le rendrait beaucoup plus exploitable. Enfin, plusieurs améliorations sont à envisager, notamment la possibilité d'une population de taille variable.

Venons-en maintenant aux progrès théoriques. L'un d'entre eux aurait des répercutions pratiques : savoir s'il existe toujours un ordre des objets tel que la solution

donnée par AMP est optimale (conjecture 8.1). Cela donnerait plus de poids à l'algorithme génétique qui serait alors toujours capable, en principe, de trouver une solution optimale, et ouvrirait de nouvelles perspectives pour des méthodes exactes.

L'existence de PTAS, FPTAS et autres algorithmes pseudo-polynomiaux ouvre de nombreuses possibilités pour l'élaboration de bons schémas d'approximation. Il est ainsi sûrement possible de faire beaucoup mieux qu'aux chapitres 5 et 9, notamment en conjugant les points forts de chaque méthode.

Quittons le cas particulier $B|size_j|H_{\max}$. Pour le concept de rangement multiboîte de manière générale, certains points sont aussi à préciser. Du point de vue de la complexité (chapitre 3), tout d'abord, les cas particuliers n'ont pas été abordés. Je pense plus particulièrement au modèle set_j et ses multiples et diverses facettes.

Et à parler de cas particuliers, que devient $Bm|allow_j|H_{\max}$ (chapitre 6) lorsque les hauteurs ne sont pas unitaires ? Le problème devient $\mathcal{NP}$-difficile, mais peut-être peut-on étendre les algorithmes proposés tout en conservant certaines garanties.

Des pans entiers du concept d'objet multiboîte n'ont été qu'à peine effleurés, et sont encore à étudier. Ainsi, pour les objectifs W, N et surtout m, trop peu de méthodes sont encore proposées, ou pour des cas trop particuliers. La très riche littérature sur les problèmes de rangement, notamment le Bin-Packing et ses variantes Bin-Packing Dual et Bin-Packing de cardinalité maximum, est truffée d'idées et d'algorithmes qu'il faudrait adapter aux cas d'objets multiboîtes.

Il reste ainsi, encore, beaucoup de labeur et de plaisir, de déconvenues et de trouvailles dans le monde des objets multiboîtes. Ces trois années m'ont donné envie de continuer la visite de celui-ci ; j'espère que ces pages ont eu, lecteur, le même effet.

Perspectives

Je pense souvent qu'il devrait y avoir un signe typographique spécial
pour désigner un sourire — une sorte de marque concave,
une parenthèse renversée sur le dos,
signe que j'aimerais pouvoir utiliser
en réponse à votre question.
— *Vladimir Nabokov*

L'essentiel des directions de recherche que je veux dégager de ce travail a été présenté au chapitre précédent. Je n'entends ici qu'effectuer une brève synthèse des points essentiels.

Les perspectives, en bref

Voici la « liste des choses à faire », chapitre par chapitre (lorsque cela a du sens). Mes goûts personnels donnent un poids particulièrement important aux extensions des chapitres 7 et 8.

Ranger des objets multiboîtes (chapitre 1)

- étendre à des boîtes non identiques.
- ne pas se contenter d'énumérer des applications, mais exploiter les résultats de ce travail sur des données réelles.

Modèles linéaires (chapitre 2)

- déterminer des modèles alternatifs et/ou des coupes pour rendre l'approche utilisable.
- étudier le passage d'une solution fractionnaire, pour un modèle bien choisi, à une solution entière réalisable.

Complexité, approximabilité (chapitre 3)

- déterminer la complexité exacte de $B|size_j|H_{\max}$.
- étudier la complexité de cas particuliers, notamment de set_j.

Le cas $Bm|any_j|\cdot$ (chapitre 5)
- étudier l'existence d'un FPTAS pour $Bm|any_j|W$.

Le cas $Bm|allow_j|H_{\max}$ (chapitre 6)
- améliorer les temps d'exécution des algorithmes.
- étendre à des hauteurs quelconques.

Bornes inférieures (chapitre 7)
- poursuivre l'étude des fonctions redondantes et de leur utilisation pour $B|size_j|H_{\max}$.
- déterminer de nouvelles bornes, notamment en se basant sur des raisonnements énergétiques locaux.
- en déduire des bornes inférieures de bonne qualité, même pour des solutions partielles, en vue d'une résolution exacte.

Heuristiques à performances garanties (chapitre 8)
- poursuivre l'étude théorique de AMP, notamment savoir s'il existe toujours un ordre optimal, et quelles seraient les garanties avec une pré-affectation de certains objets sous-optimale, mais avec des garanties.
- déterminer la performance de APOM.
- améliorer la performance de ADR.
- étudier des solutions « par blocs ».
- proposer des heuristiques pour d'autres objectifs et d'autres modèles d'objets.

Un schéma d'approximation en temps polynomial (chapitre 9)
- étudier les algorithmes obtenus en combinant le PTAS de ce chapitre avec la résolution pseudo-polynomiale ou le FPTAS du chapitre 5.
- étudier l'existence de PTAS pour d'autres objectifs et d'autres modèles d'objets.

Une approche évolutionniste (chapitre 10)
- affiner le réglage des paramètres de l'algorithme.
- comprendre précisément l'étrange prévalence de la mutation sur le croisement.
- envisager une population de taille variable.

Questions ouvertes

Voici, sobrement, la liste des questions ouvertes posées au fil du texte avec, en premier lieu, les conjectures. Ces questions sont des redites, plus précises, de certaines des perspectives rappelées ci-dessus.

* * *

Conjecture 8.1 : *[page 98]* Pour toute instance I de $B|size_j|H_{\max}$, il existe un ordre des objets $<_I$ tel que AMP appliqué pour cet ordre donne une solution optimale.

Conjecture 8.2 : *[page 114]* Une solution construite par APOM (algorithme 8.4) vérifie : $H_{\max} \leq H^*_{\max} + h_{\max}$, si les objets sont initialement rangés par largeurs décroissantes.

* * *

Question 3.1 : *[page 38]* Le problème $B|size_j|\cdot$ appartient-il à $\mathcal{NP}$?

Question 3.2 : *[page 45]* Que vaut la meilleure solution « par blocs » par rapport à une solution optimale ? et ainsi : quelles garanties peut-on avoir grâce à des solutions « par blocs » ?

Question 5.1 : *[page 63]* Peut-on obtenir un FPTAS pour $Bm|any_j|W$?

Question 7.1 : *[page 88]* Est-il difficile de trouver le sous-ensemble d'objets telle que $L_{n_{\min}}$ est la meilleure possible ? Quelle serait alors la performance de cette borne ?

Question 8.1 : *[page 109]* L'algorithme de meilleur placement décroissant (AMPD) est-il plutôt une 4/3- ou une $(4/3 - 1/3m)$-approximation ?

Question 8.2 : *[page 110]* Pour une partition $\mathbf{O}_1, \mathbf{O}_2$ de l'ensemble des objets, soit S la solution construite à partir d'une solution $S(\mathbf{O}_1)$ complétée par $\mathbf{O}_2$ selon MP. Si $S(\mathbf{O}_1)$ est une α-approximation pour les seuls objets de $\mathbf{O}_1$, a-t-on une garantie du type : $H^S_{\max} \leq \alpha H^*_{\max} + h_{\max}(\mathbf{O}_2)$?

Question 8.3 : *[page 119]* Existe-t-il une règle de fusion des solutions partielles telle que ADR soit une 4/3-approximation ?

Question 9.1 : *[page 128]* Quelle garantie a-t-on si $\mathbf{O}_{grand}$ est résolu à ε près, puis complété avec $\mathbf{O}_{petit}$ par AMPD ?

Question 9.2 : *[page 128]* Peut-on construire un FPTAS pour $Bm|size_j|H_{\max}$, utilisable en pratique, par exemple en conjuguant le FPTAS du chapitre 5 et le PTAS du chapitre 9 ?

Annexes

Complexité et approximabilité (synthèse)

On doit dire du bien le bien.
— François Villon

Ce chapitre présente une synthèse de la complexité et de l'approximabilité des problèmes de rangements multiboîtes. Ces problèmes sont regroupés par objectif (table A.1 : $H_{\max}$; table A.2 : m ; table A.3 : N ; table A.4 : W). Les notations sont données dans l'annexe D.

Table A.1 – Complexité des problèmes de rangement pour l'objectif $H_{\max}$.

Problème	**Complexité**	**Référence**
$B\|size_j, h_j = 1\|H_{\max}$	$\mathcal{O}(n)$	proposition 3.5
$B\|size_j = 1\|H_{\max}$	$\mathcal{NP}$-f	$(P\|\|C_{\max})$
$B\|size_j\|H_{\max}$	au moins $\mathcal{NP}$-f	théorème 3.1
$Bm\|size_j = 1\|H_{\max}$	$\mathcal{NP}$-d, p$\mathcal{P}$	$(Pm\|\|C_{\max})$
$Bm\|size_j\|H_{\max}$	$\mathcal{NP}$-d, p$\mathcal{P}$	théo. 3.1, théo. 5.3
	FPTAS	corollaire 5.5.1
$B\|set_j, compl\|H_{\max}$	$\mathcal{NP}$-d	(Ring Loading)
$Bm\|set_j, h_j = 1\|H_{\max}$	polynomial	théorème 5.3
$Bm\|set_j\|H_{\max}$	$\mathcal{NP}$-d, p$\mathcal{P}$	prop. 3.1, théo. 5.3
	FPTAS	corollaire 5.5.1
$B\|allow_j, h_j = 1\|H_{\max}$	polynomial	théorème 6.1
$B\|any_j\|H_{\max}$	$\mathcal{NP}$-f	théorème 3.2
$Bm\|any_j\|H_{\max}$	$\mathcal{NP}$-d, p$\mathcal{P}$	théo. 3.2, théo. 5.1
	FPTAS	théorème 5.5

Table A.2 – Complexité des problèmes de rangement pour l'objectif m.

Problème	**Complexité**	**Référence**
$B\|size_j, h_j = 1\|m$	$\mathcal{O}(n)$	proposition 3.6
$B\|size_j = 1\|m$	$\mathcal{NP}$-f	(Bin Packing)
$B\|size_j\|m$	au moins $\mathcal{NP}$-f	théorème 3.1
$B^H\|size_j\|m$	(ouvert)	
$B\|any_j\|m$	$\mathcal{NP}$-f	théorème 3.2
$B^H\|any_j\|m$	(ouvert)	

Table A.3 – Complexité des problèmes de rangement pour l'objectif N.

Problème	**Complexité**	**Référence**
$B\|size_j, h_j = 1\|N$	$\mathcal{O}(n \ln(n))$	proposition 3.7
$B\|size_j = 1\|N$	$\mathcal{NP}$-f	(Max. Card. BP)
$B\|size_j\|N$	au moins $\mathcal{NP}$-f	théorème 3.1
$Bm\|size_j\|N$	$\mathcal{NP}$-d, p$\mathcal{P}$	théo. 3.1, théo. 5.4
$B^H\|size_j\|N$	(ouvert)	
$Bm\|set_j, h_j = 1\|N$	polynomial	théorème 5.4
$Bm\|set_j\|N$	$\mathcal{NP}$-d, p$\mathcal{P}$	théorème 5.4
$B\|any_j\|N$	$\mathcal{NP}$-f	théorème 3.2
$Bm\|any_j\|N$	$\mathcal{NP}$-d, p$\mathcal{P}$	théo. 3.2, théo. 5.4
$B^H\|any_j\|N$	(ouvert)	

Table A.4 – Complexité des problèmes de rangement pour l'objectif W.

Problème	**Complexité**	**Référence**
$B^1\|size_j\|W$	$\mathcal{NP}$-d, p$\mathcal{P}$	(Sac-à-dos)
$B\|size_j, h_j = 1\|W$	$\mathcal{NP}$-d ou $\mathcal{NP}$-f	
$B1\|\|W$	$\mathcal{NP}$-d, p$\mathcal{P}$	(Sac-à-dos)
$Bm\|size_j\|W$	$\mathcal{NP}$-d, p$\mathcal{P}$	théorème 3.1
$B^H\|size_j\|W$	(ouvert)	
$B\|size_j\|W$	$\mathcal{NP}$-f	théorème 3.1
$Bm\|set_j, h_j = 1\|W$	polynomial	théorème 5.4
$Bm\|set_j\|W$	$\mathcal{NP}$-d, p$\mathcal{P}$	théo. 3.2, théo. 5.4
$B\|any_j\|W$	$\mathcal{NP}$-f	théorème 3.2
$Bm\|any_j\|W$	$\mathcal{NP}$-d, p$\mathcal{P}$	théo. 3.2, théo. 5.2
$B^H\|any_j\|W$	(ouvert)	

Compléments

Science immense et monotone.
— Henri Michaux

Cette annexe présente deux compléments aux pages précédentes. Le premier est un affinage des calculs pour le PTAS du chapitre 9 afin de déterminer les performances de cet algorithme pour le cas d'objets de largeur unitaire. Le second présente quelques courbes de temps d'exécution afin de se faire une idée plus précise du coût réel des heuristiques et de l'algorithme génétique.

Dans les deux cas, le passage en annexe permet de renforcer la cohérence des chapitres complétés autour de leur noyau. Pour le PTAS, au chapitre 9, il s'agissait de se concentrer sur l'aspect multiboîte : le cas $size_j = 1$ a déjà été traité, avec de bien meilleurs résultats, dans la littérature. Pour l'évaluation des performances des algorithmes, chapitre 11, le but était de mettre en valeur les qualités des solutions, tous les algorithmes étant très rapides.

B.1 Performances du PTAS du chapitre 9 lorsque $size_j = 1$

Dans le chapitre 9, il a été évoqué que le PTAS pour $Bm|size_j|H_{\max}$ (algorithme 9.1) est aussi un PTAS valide pour des cas particuliers comme Bi-Partition ou l'Ordonnancement sur machines parallèles. Pour ces deux cas, les calculs peuvent-être ajustés pour obtenir de meilleures garanties en tenant compte du fait que les objets sont de largeur unitaire[1].

Je reprends les notations et les conventions utilisées pour démontrer le cas général (théorème 9.1), et suppose donc que le lecteur les a en tête.

En premier lieu, pour Bi-Partition comme pour l'Ordonnancement sur machines parallèles, le choix de β peut être affiné. Soit B_1 la boîte la plus haute de la solution S obtenue à la fin de l'algorithme. Lors de l'étape 5 (les objets de $\mathbf{O}_{petit}$ sont rangés selon la règle MP par dessus une solution optimale pour $\mathbf{O}_{grand}$) deux cas sont possibles : ou bien aucun objet n'est mis dans B_1 et alors S est optimale ; ou bien un objet y est mis et alors $\Delta^S \leq h_{\max}(\mathbf{O}_{petit})$. Dans les deux cas (lemme 8.1)

[1]Ce qui suit est adapté de [94], où ces mêmes conséquences étaient évoquées pour une version antérieure, et moins performante, du PTAS.

$H_{\max} \leq H^*_{\max} + \frac{m-1}{m} h_{\max}(\mathbf{O}_{petit})$. On peut donc choisir

$$\beta = \varepsilon \frac{3m}{4(m-1)} H_{approx}$$

et l'on a toujours une $(1+\varepsilon)$-approximation.

Ensuite, c'est l'étape de résolution optimale pour l'ensemble des grands objets qui est grandement accélérée en tenant compte du fait que les objets sont tous de largeur unitaire. Le nombre de rangements possibles est maintenant $K = m^{|\mathbf{O}_{grand}|}$ et $|\mathbf{O}_{grand}| \leq \frac{4(m-1)}{3\varepsilon}$, en tenant compte de l'ajustement de β. La recherche exhaustive d'une solution optimale se fait donc en temps $\mathcal{O}(m^{\frac{4(m-1)}{3\varepsilon}})$ (soit $\mathcal{O}(2^{\frac{4}{3\varepsilon}})$ pour Bi-Partition, où $m = 2$).

Finalement, on obtient :

Proposition B.1. *L'algorithme 9.1 est un PTAS pour l'*Ordonnancement sur machines parallèles *($Pm||C_{\max}$) et* Bi-Partition *; les temps d'exécution sont respectivement de $\mathcal{O}(n \ln n + m^{\frac{4(m-1)}{3\varepsilon}})$ et $\mathcal{O}(n \ln n + 2^{\frac{4}{3\varepsilon}})$, si l'on pose $\beta = \varepsilon \frac{3m}{4(m-1)} H_{approx}$ à l'étape 2.*

Rappelons toutefois que, pour les deux cas particuliers traités, d'autres PTAS existent, spécialement conçus pour ces problèmes et ainsi plus performants [10, 67].

B.2 Temps d'exécution des algorithmes testés au chapitre 11

Pour avoir une idée des durées réelles d'exécution des algorithmes, cette section présente quelques courbes temporelles. Toutes les valeurs sont des temps CPU, calculés en moyenne pour 50 exécutions. Les tests ont été effectués sur un Pentium III 1133MHz avec 256Mo de mémoire.

La figure B.1 présente les temps d'exécution des principales heuristiques (AMPD, AMP, ADR et APOMD-X) en fonction du nombre d'objets n, pour un nombre de boîtes fixé $m = 10$. La figure B.2 présente ces temps en fonction du nombre de boîtes m, pour un nombre d'objets fixé $n = 100$.

Pour les quatre heuristiques, les temps de calcul sont très courts (moins d'une demi-seconde pour ranger 4000 objets dans 10 boîtes), en particulier ceux de AMPD en regard de la qualité des solutions fournies. ADR et APOM-X sont beaucoup moins rapides, surtout pour un grand nombre de boîtes, mais les temps de calcul sont encore tout à fait corrects. De plus, dans ces deux cas, de substantiels gains sont à envisager.

La figure B.3 présente les temps d'exécution de l'algorithme génétique en fonction du nombre maximum d'individus, pour les deux versions Gen-AMP et Gen-ADR, et des populations de taille 20 et 40. La figure B.4 présente ces temps en fonction

du nombre total d'individus. Dans tous les cas les instances ont $m = 10$ boîtes et $n = 100$ objets.

La différence entre le « nombre maximum d'individus » et le « nombre total d'individus » est que, dans le premier cas, la recherche s'arrête dès qu'une solution optimale est trouvée (celle-ci est détectée par comparaison avec la meilleure borne inférieure connue).

On constate ainsi (figure B.3) que, pour des instances choisies aléatoirement, et donc majoritairement faciles, ce phénomène d'arrêt prématuré est dominant et fait qu'en moyenne les temps d'exécution sont faibles et croissent lentement. Par contre, la variance de ces temps augmente assez rapidement, en raison des instances pour lesquelles aucune solution optimale confirmée n'a été trouvée.

Ainsi, pour des données simples (typiquement, si les largeurs et surtout les hauteurs correspondant à des distributions uniformes ou gaussiennes sur des intervalles permettent l'existence de petits objets — un cas naturel pour de nombreuses applications) l'algorithme génétique trouvera très rapidement une solution optimale et s'arrêtera.

Lorsque les données sont plus problématiques, aucune solution optimale confirmée n'est trouvée et la recherche continue jusqu'au nombre d'individus maximal autorisé (figure B.4). Les temps d'exécution restent faibles et dépendent très peu de la taille de population ; on retrouve, par contre, la différence de vitesse entre AMP et ADR.

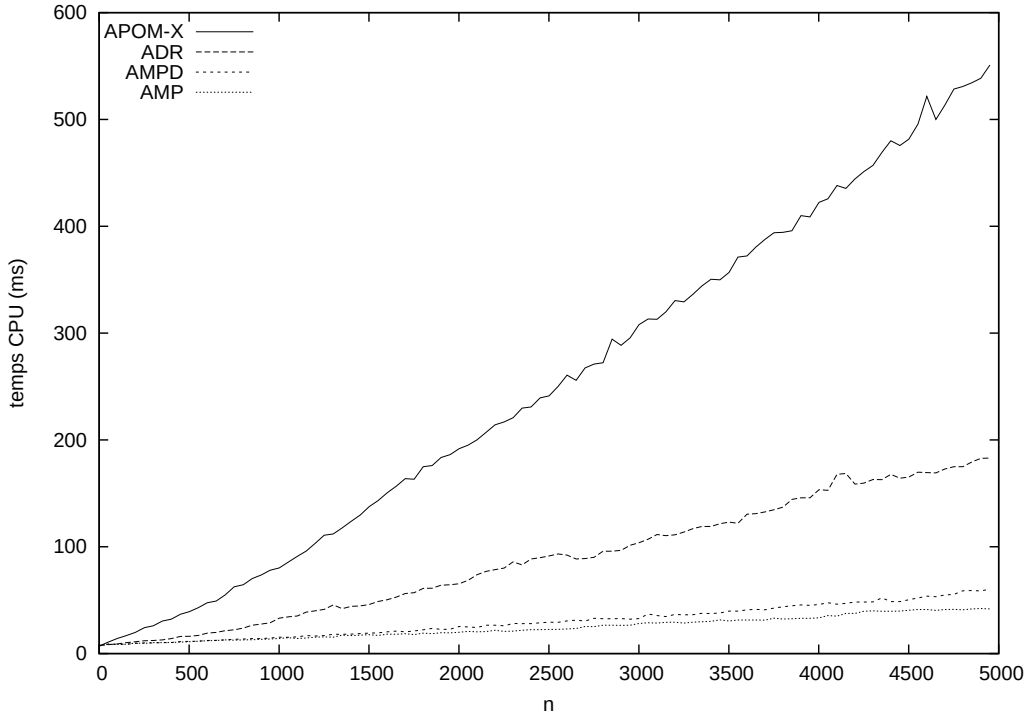


Figure B.1 – Temps d'exécution des principales heuristiques selon le nombre d'objets.

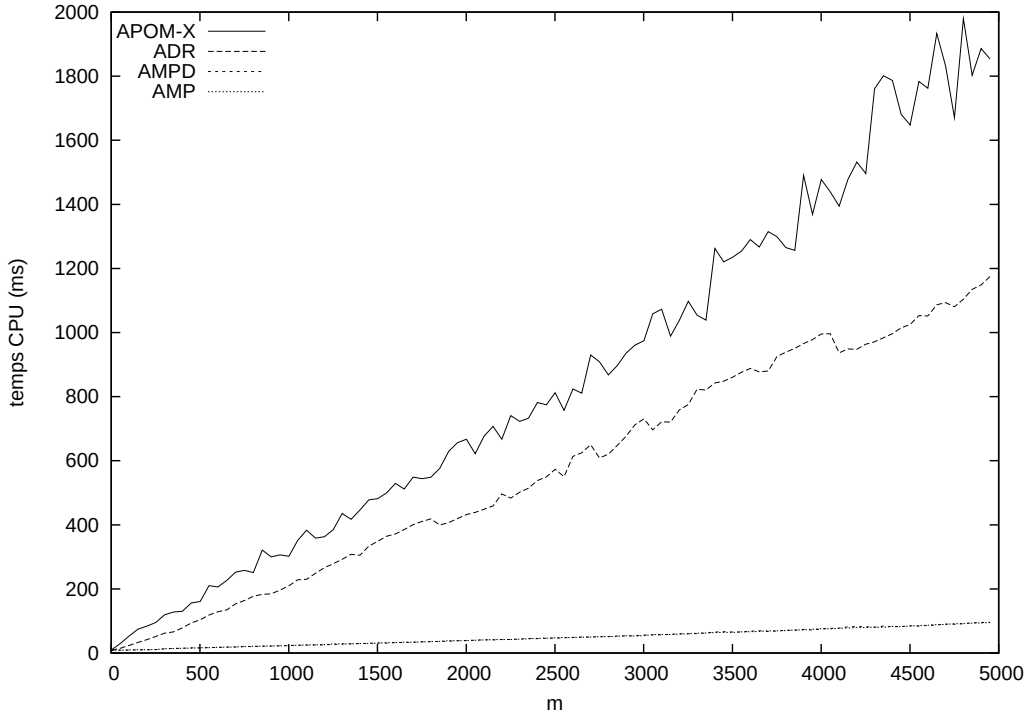


Figure B.2 – Temps d'exécution des principales heuristiques selon le nombre de boîtes.

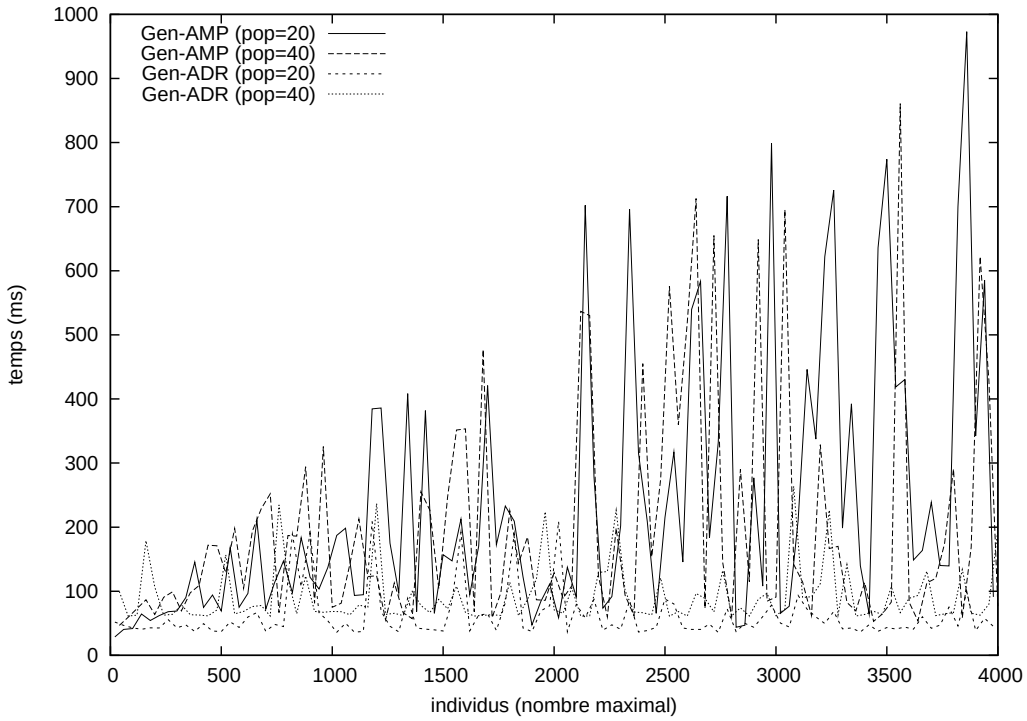


Figure B.3 – Temps d'exécution de l'algorithme génétique selon le nombre maximum d'individus.

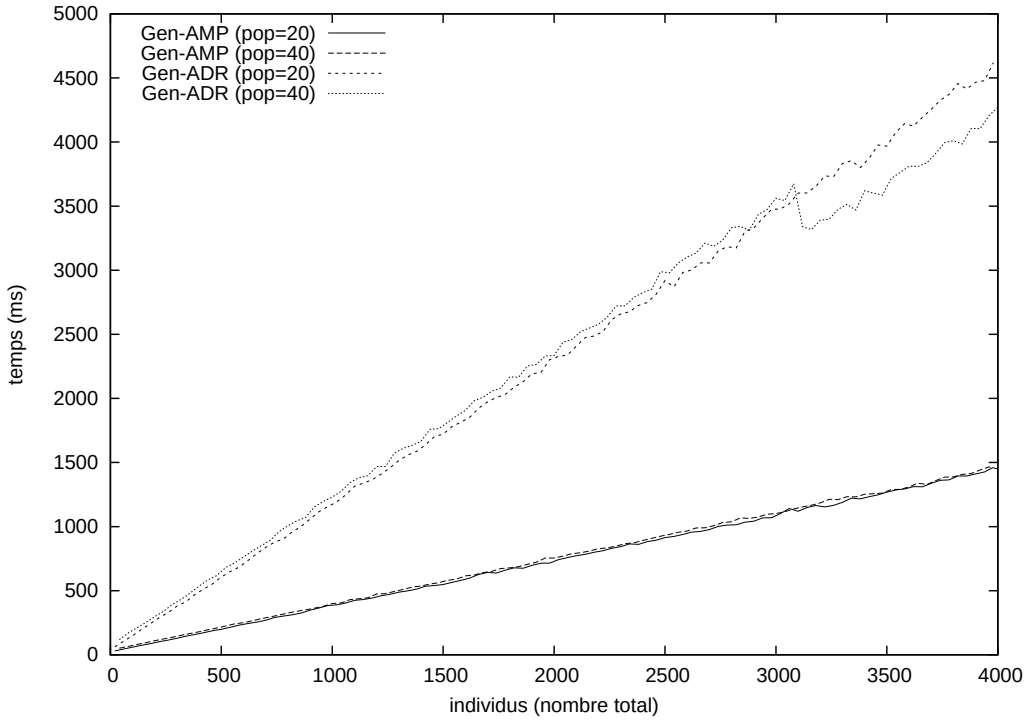


Figure B.4 – Temps d'exécution de l'algorithme génétique selon le nombre total d'individus.

Glossaire

L'éducation n'est pas une préparation à la vie ;
l'éducation est la vie elle-même.
— *John Dewey*

Ce glossaire me permet de définir les termes au sens éventuellement abscons ou technique, disséminés dans les pages précédentes. Je me suis limité aux mots dont les définitions usuelles (pour laquelle mon ouvrage de référence est le *Petit Robert* [112]) ne conviennent pas dans le contexte de ce document, ou qui tout simplement n'existent pas. J'espère que l'on me pardonnera ces néologismes, qui permettent une expression beaucoup plus claire et directe.

Pour les notions de complexité d'un problème, classe de complexité, approximation *etc.* les définitions données ne sont guère plus qu'un aide mémoire, et ne suffisent pas à véritablement assimiler les concepts. Le lecteur intéressé trouvera de nombreuses précisions et de précieux compléments dans les ouvrages d'Ausiello *et al.* [10] et de Garey et Johnson [60]. Les formulations et résultats des problèmes de références, s'il n'y a pas d'autres renvois explicites, proviennent de ces ouvrages (généralement [60] pour la complexité et [10] pour l'approximabilité). Des compléments, essentiellement sur l'approximabilité, sont disponibles *via* internet sur le *Compendium of NP optimization problems* de Crescenzi et Kann [39].

Pour une lecture à plusieurs niveaux de ce glossaire, des symboles indiquent dans la marge, lorsqu'il y a lieu, le type de définition :

- ◇ un problème de référence (problèmes de décision) ;
- ⋆ une définition technique, généralement d'un terme auquel le novice n'aura pas à attacher trop d'importance (plus il y a d'étoiles, plus c'est technique) ;
- † un néologisme.

Lorsque cela est pertinent, des renvois spécifient les autres mots du glossaire pouvant apporter des précisions (« voir aussi ... »), ainsi que les apparitions remarquables du terme dans le document (« pages ... », « chapitre ... »).

*

† Approximabilité — caractère d'un problème qui précise la possibilité de calculer des valeurs proches de l'optimum. On parle généralement de α-approximabilité pour un problème admettant une α-approximation. — voir aussi : Approximation. — chapitre 3.

*

Approximation (algorithme) — pour un problème donné, un algorithme produisant une solution réalisable pour toutes les instances. Une α-approximation est une approximation qui produit en temps polynomial des solutions dont la valeur est au plus (pour un problème de minimisation) de α fois la valeur optimale. — voir aussi : PTAS, FPTAS. — chapitres 3 et 8.

*

† Approximer — rechercher une valeur approchée.

*

⋆ Asymptotiquement optimal (algorithme) — se dit d'un algorithme d'approximation (ou d'une borne inférieure) pour lequel le rapport entre la valeur produite et la valeur optimale tend vers 1 lorsque la valeur optimale tend vers l'infini (il faut se méfier, puisque la différence entre les deux valeurs peut, elle, tendre vers l'infini). — chapitres 7 et 8.

*

◇ Bi-Partition (problème de décision)

Instance. Un ensemble fini A et une taille $s(a) \in \mathbb{N}$ pour tout élément $a \in A$.

Question. Existe-t-il un sous-ensemble $A' \subset A$ tel que $\sum_{a \in A'} s(a) = \sum_{a \in A - A'} s(a)$?

Commentaire. Complexité : $\mathcal{NP}$-complet mais se résout en temps pseudo-polynomial. — Approximabilité : admet un FPTAS. — Garey & Johnson : [SP12] Partition. — pages 21, 38, 61.

*

Bin-Packing (problème) — problème consistant à ranger un ensemble d'objets dans un minimum de boîtes identiques.

⋆ Version du Bin-Packing (problème de décision) pour laquelle on veut minimiser la borne m. — pages 20–21, 38, 41, 89–90, 96, 174.

*

◇ Bin-Packing (problème de décision)

Instance. Un ensemble fini U d'objets ; une taille $s(u) \in \mathbb{N}$ pour chaque $u \in U$; une hauteur de boîte $H \in \mathbb{N}$ et un nombre de boîtes $m \in \mathbb{N}$.

Question. Existe-il une partition de U en m ensembles disjoints $U_1, U_2, \ldots, U_m$ tels que la somme des tailles des objets dans chaque U_i est au plus H ?

Commentaire. Complexité : $\mathcal{NP}$-complet au sens fort ; $\mathcal{NP}$-complet mais se résout en temps pseudo-polynomial pour chaque $m \geq 2$; se résout polynomialement par recherche exhaustive pour tout H fixé. — Approximabilité : approximable à $\frac{3}{2}$ [117] et à $\frac{71}{60} + \frac{78}{71OPT}$ [61, 124] ; n'est pas approximable à moins de $\frac{3}{2}$ [60] ; admet un FPTAS asymptotique [76]. — Garey & Johnson : [SR1] Bin-Packing. — pages 20–21, 38, 41, 89–90, 96, 174.

*

Bin-Packing de cardinalité maximum (problème) — problème consistant à ranger le plus grand nombre d'objets d'un ensemble donné dans un nombre donné de boîtes de taille fixée.

⋆ Version du Bin-Packing (problème de décision) pour laquelle on veut maximiser la taille de l'ensemble U, pris comme sous-ensemble de tous les objets possibles. — pages 22, 174.

*

Bin-Packing dual (problème) — problème consistant à ranger un ensemble d'objets dans un maximum de boîtes identiques, chacune devant être remplie au moins jusqu'à un seuil donné. — page 21.

*

Bin-Packing ouvert (problème) — variante du Bin-Packing pour laquelle le dernier objet mis dans une boîte peut dépasser de celle-ci. — page 21.

*

CuSP (problème) — *Cumulative Scheduling Problem* : problème d'ordonnancement pour lequel les tâches partagent une ressource limitée. — voir aussi : Ordonnancement de tâches multiprocesseurs, RCPSP. — page 22.

*

Diagramme de Gantt — illustration graphique pour les ordonnancements, due à Henry L. Gantt (1861–1919). L'axe horizontal représente le temps, tandis que les ordonnées, discrètes, correspondent aux machines ; une tâche exécutée sur la machine i entre les temps t_1 et t_2 est alors dessinée comme un rectangle sur l'ordonnée i, entre les abscisses t_1 et t_2. — chapitres 1 et 4.

*

En-ligne (algorithme) — se dit d'un algorithme de rangement qui place les objets immédiatement, au fur et à mesure qu'ils lui sont donnés et sans rien connaître de ceux qui suivront, et qui ne revient jamais sur ses choix précédents. Un algorithme est *partiellement en-ligne* si certaines caractéristiques de l'instance (typiquement, le

nombre d'objets, le volume total, *etc.*) sont connues initialement. — voir aussi : Hors-ligne. — chapitre 8.

*

FPTAS (algorithme) — *Fully Polynomial-Time Approximation Scheme* (schéma d'approximation en temps fortement polynomial) : PTAS dont le temps d'exécution est polynomial en la taille de l'instance et en $1/\varepsilon$. — voir aussi : FPTAS (classe), PTAS, Approximation. — chapitres 3 et 5.

*

FPTAS (classe) — ensemble des problèmes pour lesquels il existe un FPTAS. — voir aussi : FPTAS (algorithme), PTAS, Approximation.

*

Hors-ligne (algorithme) — se dit d'un algorithme de rangement qui connaît complètement l'instance à ranger avant de prendre la moindre décision (et peut donc effectuer des pré-traitements). — voir aussi : En-ligne. — chapitre 8.

*

† Implémentable — qui peut être implémenté.

*

† Implémentation — réalisation en un certain langage informatique d'un algorithme.

*

† Implémenter — réaliser une implémentation.

*

Instance — affectation particulière des données d'un problème. Par exemple, la donnée des tailles $\{1, 4, 5, 5, 7\}$ est une instance du problème Bi-Partition.

*

⋆ $\mathcal{NP}$ (classe) — ensemble des problèmes de décision (*i.e.* dont la réponse est « oui » ou « non ») pour lesquels il existe une bonne caractérisation du « oui », c'est-à-dire pour lesquels il existe un certificat permettant de prouver rapidement (en temps polynomial en la taille de l'instance) que la réponse, pour une instance, est bien « oui ».

*

⋆ $\mathcal{NP}$-COMPLET (classe) — ensemble des problèmes de décision auxquels n'importe quel problème de la classe $\mathcal{NP}$ peut se réduire polynomialement ; c'est-à-dire les problèmes les plus durs de $\mathcal{NP}$.

*

⋆ $\mathcal{NP}$-DIFFICILE (problème) — pour lequel il n'existe *a priori* pas d'algorithme rapide (polynomial) pour le résoudre optimalement. Un problème est dit $\mathcal{NP}$-difficile *au sens fort* s'il n'existe *a priori* pas, non plus, d'algorithme pseudo-polynomial pour le résoudre optimalement. Pour le sens exact à donner à ces « *a priori* », voir [60].

*

† OPTIMALEMENT — de manière optimale, en donnant la valeur et/ou une solution optimale.

*

† OPTIMALITÉ — caractère optimal, généralement d'une solution, d'une valeur ou d'un algorithme.

*

ORDONNANCEMENT DE TÂCHES MULTIPROCESSEURS (problème) — problème d'ordonnancement pour lequel les tâches ont besoin de plusieurs processeurs pour leur exécution. — voir aussi : CUSP, RCPSP. — chapitres 1 et 4.

*

◇ ORDONNANCEMENT SUR MACHINES PARALLÈLES (problème de décision)

Instance. Un ensemble T de tâches ; un nombre $m \in \mathbb{N}$ de machines ; une durée $l(t) \in \mathbb{N}$ pour chaque tâche $t \in T$; une date de fin $D \in \mathbb{N}$.

Question. Existe-t-il un ordonnancement de T sur m machines dont la durée totale est D ou moins. En d'autres termes : existe-il une fonction $\sigma : T \to \mathbb{N}$ telle que $\forall u \geq 0$, le nombre de tâches $t \in T$ pour lesquelles $\sigma(t) \leq u < \sigma(t) + l(t)$ est au plus m et tel que $\forall t \in T : \sigma(t) + l(t) \leq D$?

Commentaire. Complexité : $\mathcal{NP}$-complet dès $m = 2$ (BI-PARTITION) mais se résout en temps pseudo-polynomial pour tout m fixé ($Pm||C_{\max}$). $\mathcal{NP}$-complet au sens fort pour m quelconque (PARTITION EN TRIPLETS est un cas particulier). — Approximabilité : admet un PTAS pour m quelconque et un FPTAS pour tout m fixé [68]. — Garey & Johnson : [SS8] Multiprocessor scheduling. — pages 13, 18, 20, 38, 40, 96, 109, 115, 118, 127, 173, 175–176.

*

$P||C_{\max}$ — voir Ordonnancement sur machines parallèles.

*

3-Partition — voir Partition en triplets.

*

Partition — voir Bi-Partition.

*

◇ Partition en triplets (problème de décision)

Instance. Un ensemble A de $3m$ éléments ; une borne $B \in \mathbb{N}$; une taille $s(a) \in \mathbb{N}$ pour chaque $a \in A$ telle que $B/4 < s(a) < B/2$ et que $\sum_{a \in A} s(a) = mB$.

Question. Peut-on partitionner A en m ensembles disjoints $A_1, A_2, \ldots, A_m$ tels que $1 \leq i \leq m$, $\sum_{a \in A_i} s(a) = B$?

Commentaire. Complexité : $\mathcal{NP}$-complet au sens fort. — Garey & Johnson : [SP15] 3-Partition. — page 38.

*

Pmtn — abréviation de « préemption » ; dans un modèle d'ordonnancement, spécifie que les préemptions sont autorisées. — voir aussi : Préemption. — page 42.

*

Polynomial (algorithme) — dont le temps d'exécution est « rapide ».

⋆ Formellement : dont le temps d'exécution ou de calcul est un polynôme de la taille des données.

*

† Polynomialement — de manière polynomiale.

⋆ Se dit généralement d'un algorithme, spécifiant ainsi que son temps d'exécution est une fonction polynomiale de la taille de l'instance.

*

Préemption — interruption de l'exécution d'une tâche pour la reprendre plus tard, éventuellement sur une autre machine. Autoriser les préemptions simplifie généralement les problèmes d'ordonnancement. — page 42.

*

PSEUDO-POLYNOMIAL (algorithme) — dont le temps d'exécution est « rapide » si les données sont raisonnablement petites.

⋆ Formellement : qui est polynomial si l'on impose une borne sur les valeurs des données (mais non leur nombre).

*

⋆⋆ PSEUDO-POLYNOMIALE (transformation) — toute fonction f d'un problème de décision $\mathcal{P}_1$ vers un problème de décision $\mathcal{P}_2$ qui vérifie que, pour $t(I)$ la taille d'une instance et $m(I)$ la valeur du plus grand nombre de l'instance I : (a) pour toute instance I de $\mathcal{P}_1$, I a une réponse affirmative si, et seulement si, $f(I)$ aussi ; (b) f peut être calculée en temps polynomial en $t(I)$ et $m(I)$; (c) la taille d'une instance de I est bornée supérieurement par un polynôme de $t(f(I))$; (d) $m(f(I))$ est bornée supérieurement par un polynôme de $t(I)$ et $m(I)$.

Propriété : s'il existe une transformation pseudo-polynomiale d'un problème $\mathcal{P}_1$ vers un problème $\mathcal{P}_2$ et que $\mathcal{P}_1$ est $\mathcal{NP}$-difficile au sens fort, alors $\mathcal{P}_2$ l'est aussi. Pour plus de détails, voir [60, page 101].

*

PTAS (algorithme) — *Polynomial-Time Approximation Scheme* (schémas d'approximation en temps polynomial) : algorithme dépendant d'un paramètre ε et qui, pour toute valeur fixée de ce paramètre, est une $(1+\varepsilon)$-approximation. — voir aussi : PTAS (classe), APPROXIMATION, FPTAS. — chapitre 9.

*

PTAS (classe) — ensemble des problèmes pour lesquels il existe un PTAS. — voir aussi : PTAS (algorithme), FPTAS, APPROXIMATION.

*

RCPSP (problème) — *Resource Constrained Project Scheduling Problem* (problème de gestion de projet avec contraintes de ressources) : modèle général d'ordonnancement où les tâches partagent une ou plusieurs ressources disponibles en quantités limitées. — voir aussi : CUSP, ORDONNANCEMENT DE TÂCHES MULTIPROCESSEURS. — page 22.

*

◇ RING LOADING PROBLEM (problème de décision)

Instance. Un nombre de clients n ; des demandes $d_{i,j} \in \mathbb{N}, 1 \leq i, j \leq n$ et une borne $T \in \mathbb{N}$.

Question. Existe-t-il une fonction $\phi : \{(i,j) : 1 \leq i < j \leq n\} \to \{0,1\}$ telle que $L = max_{1 \leq k \leq n} L_k \leq T$, où : $L_k = \sum\{d_{i,j} : \phi(i,j) = 1, k \in [i,j)\} + \sum\{d_{i,j} : \phi(i,j) = 0, k \notin [i,j)\}$?

Commentaire. Complexité : $\mathcal{NP}$-complet [114]. — Approximabilité : approximable à 2 (en fait à un facteur additif de $\max d_{ij}$) [114]. Admet un PTAS [79]. — pages 21, 24, 27, 40, 173.

*

◇ Sac-à-dos (problème de décision)

Instance. Un ensemble fini d'objets U ; une taille $s(u) \in \mathbb{N}$ et un prix $p(u) \in \mathbb{N}^+$ pour chaque $u \in U$; une taille du sac $S \in \mathbb{N}$ et une borne $P \in \mathbb{N}$.

Question. Existe-t-il un sous-ensemble d'objet $U' \subset U$ tel que $\sum_{u \in U'} s(u) \leq S$ et $\sum_{u \in U'} p(u) \geq P$?

Commentaire. Complexité : $\mathcal{NP}$-complet mais se résout en temps pseudo-polynomial. — Approximabilité : admet un FPTAS [85, 67]. — Garey & Johnson : [MP9] Knapsack. — pages 22, 38, 44, 61, 173–174.

*

Taille (des données, d'une instance, d'un problème) — place mémoire nécessaire au codage des données d'un problème, pour une représentation raisonnable, c'est-à-dire sans gaspillage inutile.

Notations et abréviations

Un écrivain devrait avoir la précision d'un poète
et l'imagination d'un scientifique.
— Vladimir Nabokov

Ce chapitre récapitule les notations utilisées dans ce document pour les problèmes de rangement d'objets multiboîtes. Certains termes ne sont formellement définis que dans le texte même ; dans ce cas, se reporter aux renvois.

$\lvert X\rvert$	pour un ensemble X, sa cardinalité.
$\lfloor x\rfloor$	pour un réel x, le plus grand entier plus petit ou égal à x.
$\lceil x\rceil$	pour un réel x, le plus petit entier plus grand ou égal à x.
C_n^k	pour deux entiers n et k, le nombre de combinaisons de k éléments parmi n.
$1..n$	abréviation pour $1, 2, ..., n$, l'ensemble des entiers de 1 à n.
$\mathcal{B}$	un problème de rangement d'objets multiboîtes.
$\mathbf{B}$	l'ensemble des boîtes d'une instance.
B_i	la boîte numéro i dans une instance.
Δ_{i_1,i_2}	$\Delta_{i_1,i_2} = \lvert H_{i_1} - H_{i_2}\rvert$: la différence de hauteur entre B_{i_1} et B_{i_2}.
Δ	$\Delta = \max_{1\leq i_1<i_2\leq m} \Delta_{i_1,i_2}$: la plus grande différence de hauteur entre deux boîtes. Au chapitre 6 (et seulement celui-là), où il n'est pas question de différences de hauteur mais de graphes, Δ représente l'opérateur de différence symétrique.
H	la hauteur maximale des boîtes, lorsque celle-ci est fixée.
H_i	$H_i = \sum_{O_j\in B_i} h_j$: la hauteur de la boîte B_i.
$H_{\max}$	$H_{\max} = \max_{i=1..m} H_i$: la hauteur de la plus grande boîte ; lorsqu'on parle de problèmes, c'est le modèle où il faut minimiser la hauteur de la plus grande boîte (voir section 1.2).
$H^*_{\max}$	la valeur optimale pour $H_{\max}$.
h_j	la hauteur de l'objet O_j.
$h_{\max}$	$h_{\max} = \max_{j=1..n} h_j$: la hauteur du plus grand objet.
$h_{\min}$	$h_{\min} = \min_{j=1..n} h_j$: la hauteur du plus petit objet.

I	une instance.
i	un numéro de boîte.
j	un numéro d'objet.
m	généralement le nombre de boîtes ; lorsqu'on parle de problèmes, c'est le modèle où il faut minimiser le nombre de boîtes (voir section 1.2).
m^*	la valeur optimale pour m.
n	le nombre d'objets d'une instance.
N	le modèle où il faut maximiser le nombre d'objets rangés (voir section 1.2).
N^*	la valeur optimale pour N.
$\mathcal{NP}$-d	$\mathcal{NP}$-difficile.
$\mathcal{NP}$-f	$\mathcal{NP}$-difficile au sens fort.
$\mathbf{O}$	l'ensemble des objets d'une instance.
O_j	l'objet numéro j dans une instance.
$\mathcal{P}$	un problème d'ordonnancement de tâches multiprocesseurs, généralement associé à un problème $\mathcal{B}$ (voir chapitre 4).
p.ex.	par exemple.
p$\mathcal{P}$	soluble en temps pseudo-polynomial.
S	une solution.
$size_j$	généralement la largeur (*i.e.* le nombre de boîtes nécessaires) de O_j ; lorsqu'on parle de problèmes, c'est le modèle où tous les objets ont une largeur fixe (voir section 1.2).
$size_{\max}$	$size_{\max} = \max_{j=1..n} size_j$: la largeur du plus gros objet.
$size_{\min}$	$size_{\min} = \min_{j=1..n} size_j$: la largeur du plus étroit objet.
W	généralement $W = \sum_{O_j \in S} w_j$: le poids total d'une solution S ; lorsqu'on parle de problèmes, c'est le modèle où il faut maximiser le poids des objets rangés (voir section 1.2).
W^*	la valeur optimale pour W.
w_j	le poids de l'objet O_j (voir aussi W).
$w_{\max}$	$w_{\max} = \max_{j=1..n} w_j$: le poids de l'objet le plus important.
$w_{\min}$	$w_{\min} = \min_{j=1..n} w_j$: le poids de l'objet le moins important.

Lorsqu'une confusion est possible, il est précisé en exposant à quel problème, quelle instance, *etc.* il est fait référence. Par exemple n^I est le nombre d'objets de l'instance I tandis que $H^S_{\max}$ est la hauteur de la plus grande boîte de la solution S.

Outils et logiciels

Les armes d'autruis ou elles te tombent du dos,
ou elles te pèsent, ou elles te serrent.
— Niccolò Machiavel

Aucun travail ne se fait sans l'aide des œuvres d'autres personnes[1]. Je veux donc parler ici des livres et surtout des logiciels qui ont grandement simplifié et aidé ma tâche. Pour le support intellectuel, moral et affectif, les remerciements seront une place plus appropriée.

Mon système d'exploitation a été ce que j'apprécie le plus pour faire du développement, ou tout autre application sérieuse : un `Linux`[2] (Mandrake 8.2, avec l'environnement KDE), indispensable pour la puissance de ses interpréteurs de commandes et ses petits programmes miracles, notamment `make`, `grep` et `awk`, qui permettent de transformer le cauchemar de milliers de tests en une douce torpeur automatisée... Pour ce qui est de l'édition de texte, je n'aurais fait aucune infidélité à `nedit` (simple, mais tellement pratique !).

Pour la programmation, tous les algorithmes présentés dans ce document ont été réalisés en Objective Caml (`ocamlopt`, version 3.06), un langage très puissant, efficace, et mêlant habilement les styles fonctionnel, objet et impératif. Le compilateur pour ce langage est développé par l'INRIA et est disponible pour la plupart des plateformes[3]. L'apprentissage, très rapide, de ce langage aux nombreuses possibilités a été grandement simplifié par le livre de Chailloux, Manoury et Pagano [30].

Pour divers outils j'ai aussi fait confiance au C et à l'excellent compilateur `gcc` (version 2.96). Mon manuel de référence pour ce langage est la documentation écrite par Cassagne [29].

L'analyse des résultats de mes tests a été facilitée par l'utilisation du tableur de `OpenOffice.org`[4] (version 1.1).

Pour la rédaction, je suis un inconditionnel de cette petite merveille nommée TEX (avec une petite sur-couche de LATEX, tout de même). Le format de ce document a été conçu et réalisé par mes soins. Je n'en aurais cependant jamais eu les compétences

[1] Peu de textes se privent de la suavité des lieux communs.
[2] *Linux Online !* — `http://www.linux.org`.
[3] *The O'Caml language* — `http://www.ocaml.org`.
[4] *OpenOffice.org* — `http://www.openoffice.org`.

sans la lecture du *TEXbook* de Knuth [81] et du *Petit livre de TEX* de Séroul [116]. Diller et son *LATEX line by line* [42] m'ont aussi apporté de nombreux et précieux éclairages sur ce format parfois obscur qu'est LaTeX. Je dois aussi aux auteurs des bibliothèques LaTeX[5] `babel`, `epsfig`, `epic` et `eepic`, et `rotating` de m'avoir épargné une part énorme d'un travail déjà si bien réalisé par leurs soins, et dépassant ma maîtrise actuelle de TeX et LaTeX.

Pour toutes les figures, `xfig`[6] est mon outil, tandis que je confie les graphiques à `gnuplot`[7]. Je ne peux pas, non plus, ne pas citer les programmes `dvips`, `xdvi` et autre `ghost view`, indispensables à l'obtention et à la lecture d'un document fini.

En préparant ma soutenance, j'ai découvert et utilisé l'impressionnant format `beamer`[8] pour des présentations avec LaTeX et le très pratique et polyvalent METAPOST[9] pour les figures.

Un grand, grand merci aux auteurs de tous ces programmes et manuels.

[5] Disponibles *via CTAN: the Comprehensive TEXArchive Network* — `http://www.ctan.org`
[6] *Xfig, Drawing Program for the X Window System* — `http://www.xfig.org`.
[7] *Gnuplot homepage* — `http://www.gnuplot.info`.
[8] *The LaTeX Beamer Class Homepage* — `http://latex-beamer.sourceforge.net/`.
[9] *MetaPost* — `http://cm.bell-labs.com/who/hobby/MetaPost.html`.

Liste des algorithmes

3.1 : Résolution de $B|size_j, h_j = 1|H_{\max}$ 42
3.2 : Résolution de $B|size_j, h_j = 1|N$ 43
5.1 : Un algorithme récursif pour $B2|any_j|H_{\max}$ 59
5.2 : Un algorithme itératif pour $B2|any_j|H_{\max}$ 59
5.3 : Un FPTAS pour $Bm|any_j|H_{\max}$ 62
6.1 : Recherche d'un chemin réducteur 73
7.1 : Calcul de $L_{n_{\min}}$ 88
7.2 : Calcul de $L_+^{L_V}$ 91
8.1 : AMP 97
8.2 : APBS 110
8.3 : APO 113
8.4 : APOM 115
8.5 : ADR 117
8.6 : ADO 119
8.7 : ADOM 120
9.1 : Un PTAS pour $Bm|size_j|H_{\max}$ 126

Liste des figures

1.1 : Passage d'un ordonnancement à un rangement. 18
1.2 : Passage d'un ordonnancement de tâches multiprocesseurs à un rangement d'objets multiboîtes. 19
1.3 : Un anneau unidirectionnel (réseau SDH). 26
1.4 : Un anneau bidirectionnel (réseau SONET). 26
4.1 : Un exemple d'abaissement d'un ordonnancement. 49
4.2 : Un exemple d'abaissement non optimal. 50
4.3 : Un exemple de mauvais abaissement pour $Pm|size_j, p_j = 1|C_{\max}$. . . . 53
4.4 : Un exemple de mise-à-niveaux optimale pour $B2|size_j|H_{\max}$. 54
5.1 : Principe de résolution de $B2|any_j|H_{\max}$. 58
6.1 : Résolution de $B|allow_j, h_j = 1|H_{\max}$ par un flot maximal. 67
6.2 : Un exemple de facteur optimal admettant un chemin réducteur. 73
6.3 : Un contre exemple pour la réciproque de la proposition 6.2. 76
7.1 : Instances problématiques pour les bornes de volume. 86
7.2 : Instances problématiques pour la borne $L_{n_{\min}}$. 88
8.1 : Ajout d'un objet selon la règle MP. 97
8.2 : Influence de l'ordre des objets sur la règle MP. 98
8.3 : Un exemple d'instance pour lequel AMP est mauvais. 101
8.4 : Un exemple atteignant la borne du théorème 8.3. 106
8.5 : Transformation d'une solution optimale en solution MPD. 108
8.6 : Un exemple d'instance pour lequel ABPS est exécrable. 111
8.7 : Un exemple d'instance pour lequel ABPSD est mauvais. 112
8.8 : Un exemple d'instance pour lequel APO et APOM sont mauvais. . . . 114
8.9 : Un exemple d'exécution d'ADR. 116
8.10 : Un exemple d'instance pour lequel ADR est mauvais. 118
10.1 : Un exemple de mutation. 132
10.2 : Un exemple de croisement. 132
11.1 : Performances des bornes L_V, $L_{n_{\min}}$ et $L_{\max}$ en fonction du nombre d'objets. 143
11.2 : Gains apportés par Gen-AMP et Gen-ADR en fonction du nombre de générations. 156
11.3 : Gains apportés par Gen-AMP et Gen-ADR par rapport à Alea-AMP et Alea-ADR en fonction du nombre de solutions générées. 156
11.4 : Gains apportés par Gen-AMP et Gen-ADR en fonction du nombre de générations et selon la taille des instances. 157

11.5 : Gains apportés par Gen-AMP et Gen-ADR en fonction du nombre de générations et selon le type des instances. 158
B.1 : Temps d'exécution des principales heuristiques selon le nombre d'objets. 178
B.2 : Temps d'exécution des principales heuristiques selon le nombre de boîtes. 178
B.3 : Temps d'exécution de l'algorithme génétique selon le nombre maximum d'individus. 179
B.4 : Temps d'exécution de l'algorithme génétique selon le nombre total d'individus. 179

Liste des tables

4.1 : Quelques exemples de problèmes associés. 48
4.2 : Complexités comparées de problèmes associés. 48
6.1 : Complexité de $B|allow_j, h_j = 1|H_{\max}$ selon l'algorithme de flot. 68
8.1 : Comparaison théorique des heuristiques du chapitre 8. 122
10.1 : Caractéristiques des instances pour le réglage de l'algorithme génétique. 134
11.1 : Comparaison des bornes inférieures pour $B|size_j|H_{\max}$. 142
11.2 : Comparaison des heuristiques pour $B|size_j|H_{\max}$ (domination). 146
11.3 : Comparaison des heuristiques pour $B|size_j|H_{\max}$ (optimalité). 147
11.4 : Comparaison des heuristiques pour $B|size_j|H_{\max}$ (écarts moyens). . . . 148
11.5 : Comparaison des heuristiques pour $B|size_j|H_{\max}$ (écarts maximums). . 149
11.6 : Tests de AMP et ADR sur des ordres aléatoires (domination). 152
11.7 : Tests de AMP et ADR sur des ordres aléatoires (optimalité). 152
11.8 : Tests de AMP et ADR sur des ordres aléatoires (écarts moyens). 153
11.9 : Tests de AMP et ADR sur des ordres aléatoires (écarts maximums). . . 153
11.10 : Gains apportés par Gen-AMP selon le nombre de générations. 155
11.11 : Gains apportés par Gen-ADR selon le nombre de générations. 155
A.1 : Complexité des problèmes de rangement pour l'objectif $H_{\max}$. 173
A.2 : Complexité des problèmes de rangement pour l'objectif m. 174
A.3 : Complexité des problèmes de rangement pour l'objectif N. 174
A.4 : Complexité des problèmes de rangement pour l'objectif W. 174

Bibliographie et références

Dans l'ensemble, les livres furent son expérience.
— Henri Michaux

1. Abdelfattah Abouelaoualim. Heuristiques avec performances pour les problèmes de rangements multiboîtes. Mémoire de DEA, INPG, Grenoble, France, juin 2003. (42)

2. Ravindra K. Ahuja, Thomas L. Magnanti et James B. Orlin. *Network Flows*. Prentice Hall Inc., 1993. (67, 76)

3. Dante Alighieri. *Divina Commedia*. 1320.

4. Helmut Alt, Norbert Blum, Kurt Mehlhorn et Manfred Paul. Computing a Maximum Cardinality Matching in a Bipartite Graph in Time $O(n^{1.5}\sqrt{m/\log n})$. *Information Processing Letters*, 37(4), 237–240, 1991. (69, 76)

5. Richard P. Anstee. Simplified existence theorem for (g, f)-factors. *Discrete Applied Mathematics*, 27, 29–38, 1990. (69)

6. Roberto Aringhieri et Mauro Dell Amico. Solutions for the SONET Ring Assignment with Capacity Constraints. Dans *MIC 2001 - 4th Metaheuristics International Conference*, pages 661–666, Porto, Portugal, 16-20 juillet 2001. — voir aussi [7]. (27)

7. Roberto Aringhieri et Mauro Dell Amico. A Variable-Neighborhood Variable-Objective Tabu Search Algorithm for the SONET Ring Assignment with Capacity Constraints. Dans C. Rego et B. Alidaee (ed.), *Adaptive Memory and Evolution: Tabu Search and Scatter Search*. Kluwer, à paraître. — voir aussi [6].

8. Armen S. Asratian, Tristan M.J. Denley et Roland Häggkvist. *Bipartite Graphs and their Applications*. Cambridge University Press, 1998. (69)

Dans cette bibliographie, des références liées (p.ex. une conférence et un article sur les mêmes résultats) se font explicitement référence par un « voir aussi ... ». Les numéros entre parenthèses (p.ex. « (20, 96, 101, 106–109) ») au bout de la dernière ligne d'un enregistrement indiquent la ou les page(s) où la référence est citée (en fait : la ou les page(s) où débute le paragraphe concerné).

9. S. F. Assmann, David S. Johnson, Daniel J. Kleitman et Joseph Y.-T. Leung. On a Dual Version of the One-Dimensional Bin Packing Problem. *Journal of Algorithms*, 5(4), 502–525, 1984. (22)

10. Giorgio Ausiello, Pierluigi Crescenzi, Giorgio Gambosi, Viggo Kann, Alberto Marchetti-Spaccamela et Marco Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer, Berlin, 1999. — voir aussi [39]. (21, 37–38, 40, 61, 127, 176, 181)

11. Brenda S. Baker et Edward G. Coffman, Jr. Mutual exclusion scheduling. *Theoretical Computer Science*, 162(2), 225–243, 1996. (22–23)

12. Philippe Baptiste, Claude Le Pape et Wim Nuijten. Satisfiability Tests and Time-Bound Adjustments for Cumulative Scheduling Problems. *Annals of Operations Research*, 92, 305–333, 1999. (22, 87)

13. Jacek Błażewicz, Mieczyslaw Drabowski et Jan Węglarz. Scheduling Multiprocessor Tasks to Minimize Schedule Length. *IEEE Transactions on Computers*, C-35(5), 389–393, 1986. (22, 48, 51)

14. Jacek Błażewicz, Maciej Drozdowski et Klaus Ecker. Management of Resources in Parallel Systems. Dans J. Błażewicz, K. Ecker, B. Plateau et D. Trystram (ed.), *Handbook on Parallel and Distributed Processing*, pages 216–293. Springer, Heidelberg, 1999. (22)

15. Jacek Błażewicz, Maciej Drozdowski et Jan Węglarz. Scheduling Multiprocessor Tasks – a survey. *International Journal of Microcomputer Applications*, 13(2), 89–97, 1994. (22–23, 48, 55, 57)

16. Jacek Błażewicz, Klaus H. Ecker, Erwin Pesch, Günter Schmidt et Jan Węglarz. *Scheduling Computer and Manufacturing Processes*. Springer-Verlag, 1996. (20, 42)

17. Jacek Błażewicz, Jan Karel Lenstra et Alexander H. G. Rinnooy Kan. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5, 11–24, 1983. (23)

18. Hans L. Bodlaender, Klaus Jansen et Gerhard Woeginger. Scheduling with incompatible jobs. *Discrete Applied Mathematics*, 55(3), 219–232, 1994. (22–23)

19. Marco A. Boschetti et Aristide Mingozzi. The two-dimensional finite bin packing problem. part I: New lower bounds for the oriented case. *4OR*, 1(1), 27–42, 2003. — voir aussi [20].

20. Marco A. Boschetti et Aristide Mingozzi. The two-dimensional finite bin packing problem. part II: New lower and upper bounds. *4OR*, 1(2), 135–147, 2003. — voir aussi [19]. (23)

21. Corinne Boutevin. *Problèmes d'ordonnancement et d'affectation avec contraintes de ressources de type RCPSP et line balancing*. Thèse de doctorat, Université Blaise Pascal, Clermont-Ferrand, 2003. (23)

22. Nadia Brauner, Yves Crama, Gerd Finke, Pierre Lemaire et Christelle Wynants. Approximation Algorithms for the Design of SONET Networks. *RAIRO - Operations Research*, 37, 235–247, 2003. — voir aussi [24, 23, 89]. (25)

23. Nadia Brauner, Yves Crama, Pierre Lemaire et Christelle Wynants. Complexité et approximation pour la conception de réseaux SONET/SDH. Rapport technique 61, Les Cahiers du Laboratoire Leibniz-IMAG, 2002. http://www-leibniz.IMAG.fr/LesCahiers. — voir aussi [22, 24, 89]. (25)

24. Nadia Brauner, Yves Crama, Pierre Lemaire et Christelle Wynants. Complexity and Approximation Algorithms for SONET Network Design. Dans *CO'02, International Symposium on Combinatorial Optimization*, Paris, 8-10 avril 2002. — voir aussi [22, 23, 89].

25. Nadia Brauner et Pierre Lemaire. A Set-Covering Approach for SONET Network Design. Rapport technique 62, Les Cahiers du Laboratoire Leibniz-IMAG, 2002. http://www-leibniz.IMAG.fr/LesCahiers. (25)

26. Jacques Carlier, François Clautiaux et Emmanuel Néron. Évaluation par défaut pour les problèmes de gestion de projet à contrainte de ressources : lien avec les problèmes de bin-packing. Dans *EARO'03, École d'Automne de Recherche Opérationnelle*, 28-31 octobre 2003. (23)

27. Jacques Carlier et Emmanuel Néron. A new L.P. based lower bound for the cumulative scheduling problem. *European Journal of Operational Research*, 127(2), 363–382, 2000. (22, 87, 89, 93)

28. Jacques Carlier et Emmanuel Néron. On linear lower bounds for the resource constrained project scheduling problem. *European Journal of Operational Research*, 149(2), 314–324, 2003. (22, 89, 93)

29. Bernard Cassagne. *Introduction au langage C*, 1998. (191)

30. Emmanuel Chailloux, Pascal Manoury et Bruno Pagano. *Développement d'applications avec Objective Caml*. O'Reilly, 2000. (191)

31. Chandra Chekuri et Sanjeev Khanna. On Multi-Dimensional Packing Problems. Dans *SODA: ACM-SIAM Symposium on Discrete Algorithms (A*

Conference on Theoretical and Experimental Analysis of Discrete Algorithms), 1999. (23)

32. Bo Chen, Chris N. Potts et Gerhard J. Woeginger. A Review of Machine Scheduling: Complexity, Algorithms and Approximability. Dans D.-Z. Du et P.M. Pardalos (ed.), *Handbook of Combinatorial Optimization*, pages 21–169. Kluwer Academic Publishers, 1998. (21)

33. Jianer Chen et Antonio Miranda. A Polynomial Time Approximation Scheme for General Multiprocessor Job Scheduling. *SIAM Journal on Computers*, 31(1), 1–17, 2001. (125)

34. Vasek Chvátal. A Greedy Heuristic for the Set-Covering Problem. *Mathematics of Operations Research*, 4(3), 233–235, 1979. (25)

35. Edward G. Coffman, Jr. et Ravi Sethi. A generalized bound on LPT sequencing. *Revue Bleue d'AFCET, (R.A.O. Informatique)*, 10, 17–25, 1976. (108–109)

36. Edward G. Coffman Jr, Michael R. Garey et David S. Johnson. An application of Bin-Packing to Multiprocessor Scheduling. *SIAM Journal on Computers*, 7(1), 1–17, 1978. (20, 115)

37. Edward G. Coffman Jr, Michael R. Garey et David S. Johnson. Approximation Algorithms for Bin-Packing: a survey. Dans Dorit S. Hochbaum (ed.), *Approximation Algorithms for NP-hard Problems*, chapitre 2, pages 46–93. PWS Publishing Company, 1997. (21, 41, 89, 96)

38. Edward G. Coffman Jr., Joseph Y.-T. Leung et D. W. Ting. Bin Packing: Maximizing the Number of Pieces Packed. *Acta Informatica*, 9(3), 263–271, 1978. (22)

39. Pierluigi Crescenzi et Viggo Kann. A compendium of NP optimization problems. http://www.nada.kth.se/~viggo/wwwcompendium. — voir aussi [10]. (181)

40. Lawrence Davis (ed.). *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, 1991. (130–132)

41. Kenneth De Jong. *An analysis of the behavior of a class of genetic adaptive systems*. Thèse de doctorat, University of Michigan, Ann Arbor, 1975. (129)

42. Antoni Diller. *LaTeX line by line*. Wiley & Sons, 1993. (192)

43. Maciej Drozdowski. Scheduling multiprocessor tasks - an overview. *European Journal of Operational Research*, 94, 215–230, 1996. (22–24)

44. Jianzhong Du et Joseph Y.-T. Leung. Complexity of Scheduling Parallel Task Systems. *SIAM Journal on Discrete Mathematics*, 2(4), 473–487, 1989. (22, 48, 57)

45. Jack Edmonds et D. Ray Fulkerson. Bottleneck Extrema. *Journal of Combinatorial Theory*, 8, 299–306, 1970. (77)

46. ETSI. ETSI - Telecom Standards. http://www.etsi.org.

47. ETSI. Transmission and Multiplexing (TM) ; Synchronous Digital Hierarchy (SDH) ; Network protection schemes ; Rings and other schemes. Spécification technique, novembre 1997. ref: TS 101 010 v1.1.1. — voir aussi [46]. (25)

48. ETSI. Transmission and Multiplexing (TM) ; Synchronous Digital Hierarchy (SDH) ; Network protection schemes ; Types and characteristics. Spécification technique, novembre 1997. ref: TS 101 009 v1.1.1. — voir aussi [46]. (25)

49. Sandor P. Fekete et Jörg Schepers. On more-dimensional packing I: Modeling. Rapport technique 288, Mathematisches Institut, Universität zu Köln, 1997. — voir aussi [50, 51].

50. Sandor P. Fekete et Jörg Schepers. On more-dimensional packing II: Bounds. Rapport technique 289, Mathematisches Institut, Universität zu Köln, 1997. — voir aussi [49, 51]. (23)

51. Sandor P. Fekete et Jörg Schepers. On more-dimensional packing III: Exact Algorithms. Rapport technique 290, Mathematisches Institut, Universität zu Köln, 1997. — voir aussi [49, 50].

52. Sandor P. Fekete et Jörg Schepers. New classes of lower bounds for bin packing problems. Dans R. E. Bixby, E. A. Boyd et R. Z. Ríos-Mercado (ed.), *Integer Programming and Combinatorial Optimization, Proc. 6th International IPCO Conference*, volume 1412 of *Lecture Notes in Computer Science*, pages 257–270. Springer, juin 1998. — voir aussi [53].

53. Sandor P. Fekete et Jörg Schepers. New classes of lower bounds for bin packing problems. *Mathematical Programming*, 91(1), 11–31, 2001. — voir aussi [52]. (21, 89–90, 92)

54. Wenceslas Fernandez de la Vega et George S. Lueker. Bin packing can be solved within $1 + \varepsilon$ in linear time. *Combinatorica*, 1, 349–355, 1981. (21)

55. Gerd Finke et Nadia Brauner. Packing of multi-bin objects. Document de travail, février 2001. (17, 24)

56. Gerd Finke, Nadia Brauner et Pierre Lemaire. Packing of multi-bin objects. Dans *New trends in scheduling for parallel and distributed systems*, Marseilles, 1-5 octobre 2001. (17, 47)

57. Matteo Fischetti et Silvano Martello. Worst-case analysis of the differencing method for the partition problem. *Mathematical Programming*, 37, 117–120, 1987. (118)

58. Lester R. Ford et D. Ray Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8, 399–404, 1956. (68)

59. Bernard Fortz, Patrick Soriano et Christelle Wynants. A Tabu Search Algorithm for Self-Healing Ring Network Design. *European Journal of Operational Research*, 151(2), 2003. (27)

60. Michael R. Garey et David S. Johnson. *Computers and Intractability (A Guide to the Theory of NP-Completeness)*. W.H. Freeman And Company, 1979. (20–22, 37–38, 40–41, 48, 181–182, 185, 187)

61. Michael R. Garey et David S. Johnson. A 71/60 Theorem for Bin-Packing. *Journal of Complexity*, 1(1), 65–106, 1985. (21, 183)

62. Olivier Goldschmidt, Dorit S. Hochbaum, Asaf Levin et Eli V. Olinick. The SONET Edge-Partition Problem. *Networks*, 41(1), 13–23, 2003. (25)

63. Olivier Goldschmidt, Alexandre Laugier et Eli V. Olinick. SONET/SDH Ring Assignment with Capacity Constraints. *Discrete Applied Mathematics*, 129(1), 99–128, 2003. (27)

64. Ronald L. Graham. Bounds On Multiprocessor Timing Anomalies. *SIAM Journal of Applied Mathematics*, 17(2), 416–429, 1969. (20, 96, 101, 106–109)

65. Ronald R. Graham, Eugene L. Lawler, Jan Karel Lenstra et Alexander H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling theory: A survey. *Annals of Discrete Mathematics*, 5, 287–326, 1979. (23)

66. Katherine Heinrich, Pavol Hell, David G. Kirkpatrick et Guizhen Liu. A Simple existence criterion for $(g < f)$-factors. *Discrete Mathematics*, 85, 313–317, 1990. (69)

67. Dorit S. Hochbaum (ed.). *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company, 1997. (22, 127, 176, 188)

68. Dorit S. Hochbaum et David B. Shmoys. Using Dual Approximation Algorithms for Scheduling Problems: Theoretical and Practical Results. *Journal of the Association for Computing Machinery*, 34(1), 144–162, 1987. (20, 127, 185)

69. John H. Holland. *Adaptation in natural and artificial systems*. University of Michigan Press, 1975. Nouvelle édition : MIT Press, 1992. (129)

70. John E. Hopcroft et Richard M. Karp. An $n^{2.5}$ Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM Journal on Computing*, 2(4), 225–231, 1973. (76)

71. Georges Ifrah. *Histoire universelle des chiffres*, volume 1. Robert Laffont, Bouquins, 1994. — voir aussi [72]. (11)

72. Georges Ifrah. *Histoire universelle des chiffres*, volume 2. Robert Laffont, Bouquins, 1994. — voir aussi [71]. (11)

73. Sandy Irani et Vitus J. Leung. Scheduling with Conflicts, and Applications to Traffic Signal Control. Dans *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, pages 85–94, 1996. (23)

74. Klaus Jansen. An Approximation Scheme for Bin Packing with Conflicts. *Journal of Combinatorical Optimization*, 3, 363–377, 1999. (23)

75. David S. Johnson, Alan Demers, Jeffrey D. Ullman, Michael R. Garey et Ronald L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3(4), 299–325, 1974. (21)

76. Narendra Karmarkar et Richard M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. Dans *23rd Annual Symposium on Foundations of Computer Science*, pages 312–320, Chicago, Illinois, 3-5 novembre 1982. (21, 183)

77. Narendra Karmarkar et Richard M. Karp. The differencing method of the set partitioning. Rapport technique UCB/CSD 82/113 Computer SCi. Division (EECS), University of California, Berkeley, California, 1982. (118)

78. Hans Kellerer, Ulrich Pferschy et David Pisinger. *Knapsack problems*. Springer, 2004. (22)

79. Sanjeev Khanna. A Polynomial Time Approximation Scheme for the SONET Ring Loading Problem. *Bell Labs Technical Journal*, pages 36–41, 1997. (27, 125, 188)

80. M. Klein. A Primal Method for Minimal Cost Flow with Application to the Assignment and Tranportation Problems. *Management Science*, 14, 205–220, 1967. (76)

81. Donald E. Knuth. *The TEXbook — Computers & Typesetting*. Addison-Wesley, 1986. (192)

82. Martine Labbé, Gilbert Laporte et Silvano Martello. An exact algorithm for the dual bin packing problem. *Operations Research Letters*, 17(1), 9–18, 1995. (22)

83. Martine Labbé, Gilbert Laporte et Silvano Martello. Upper bounds and algorithms for the maximum cardinality bin packing problem. *European Journal of Operational Research*, 149(3), 490–498, 2003. (22)

84. Manuel Laguna. Clustering for the Design of SONET Rings in Interoffice Telecommunications. *Management Science*, 40(11), 1533–1541, 1994. (27)

85. Eugene L. Lawler. Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research*, 4, 339–356, 1979. (22, 188)

86. Youngho Lee, Hanif D. Sherali, Junghee Han et Seong-in Kim. A branch-and-cut algorithm for solving an intraring synchronous optical network design problem. *Networks*, 35(3), 223–232, 2000. (27)

87. Pierre Lemaire. Optimisation de réseaux SONET/SDH : éléments théoriques et résolution pratique. Mémoire de DEA, INPG, Grenoble, France, juin 2001. (25)

88. Pierre Lemaire. Rangement d'objets multiboîtes, flots et facteurs de graphes. Dans *Francoro IV, 4º journées francophones de recherche opérationnelle*, Fribourg, Suisse, 18-21 août 2004. (65)

89. Pierre Lemaire, Nadia Brauner et Gerd Finke. Une méthode tabou pour la décomposition de réseaux SONET/SDH en anneaux. Dans *Roadef 2002, 4º congrès de la société française de recherche opérationnelle et d'aide à la décision*, Paris, France, 20-22 février 2002. — voir aussi [22, 24, 23].

90. Pierre Lemaire, Gerd Finke et Nadia Brauner. Packing of multi-bin objects. Dans *CO'02, International Symposium on Combinatorial Optimization*, Paris, France, 8-10 avril 2002. — voir aussi [97]. (17, 37, 47, 57)

91. Pierre Lemaire, Gerd Finke et Nadia Brauner. Packing of Multibin Objects. Rapport technique 69, Les Cahiers du Laboratoire Leibniz-IMAG, 2002. http://www-leibniz.IMAG.fr/LesCahiers. — voir aussi [96, 98].

92. Pierre Lemaire, Gerd Finke et Nadia Brauner. Approximation Algorithms for Multibin Packing Problems. Dans *EARO'03, École d'Automne de Recherche Opérationnelle*, pages 11–15, Tours, France, 28-31 octobre 2003. (95)

93. Pierre Lemaire, Gerd Finke et Nadia Brauner. An Approximation Scheme for the Packing of Multibin Objects. Dans *MAPSP'03, Sixth Workshop on Models and Algorithms for Planning and Scheduling Problems*, Aussois, France, 30 mars-4 avril 2003. — voir aussi [94, 95, 99]. (95, 125)

94. Pierre Lemaire, Gerd Finke et Nadia Brauner. Multibin Packing and the Best-Fit Rule. Rapport technique 71, Les Cahiers du Laboratoire Leibniz-IMAG, 2003. http://www-leibniz.IMAG.fr/LesCahiers. — voir aussi [93, 95, 99]. (175)

95. Pierre Lemaire, Gerd Finke et Nadia Brauner. Multibin Packing with the Best-Fit Rule. Dans Graham Kendall, Edmund Burke et Sanja Petrovic (ed.), *MISTA 2003, The 1st Multidisciplinary International Conference on Scheduling : Theory and Applications*, volume 1, pages 74–88, Nottingham, UK, 13–16 août 2003. — voir aussi [94, 93, 99]. (95, 125)

96. Pierre Lemaire, Gerd Finke et Nadia Brauner. Packing of Multibin Objects. Dans *IEPM'03, International Conference on Industrial Engineering and Production Management*, volume 1, pages 422–431, Porto, Portugal, 26-28 mai 2003. Mise-à-jour disponible à : www-leibniz.imag.fr/~lemaire. — voir aussi [91, 98]. (17, 37, 42, 47, 57, 127)

97. Pierre Lemaire, Gerd Finke et Nadia Brauner. Rangement d'objets multiboîtes. Dans *Roadef 2003, 5º congrès de la société française de recherche opérationnelle et d'aide à la décision*, Avignon, France, 26-28 février 2003. — voir aussi [90]. (17, 37)

98. Pierre Lemaire, Gerd Finke et Nadia Brauner. Packing of Multibin Objects: Models and Complexity. (soumis). — voir aussi [91, 96]. (17)

99. Pierre Lemaire, Gerd Finke et Nadia Brauner. The Best-Fit Rule for Multibin Packing: An extension of Graham's list algorithms. *Sélection de papiers de la conférence MISTA*, à paraître. — voir aussi [94, 93, 95].

100. Joseph Y.-T. Leung, Moshe Dror et Gilbert H. Young. A Note on an Open-end Bin Packing Problem. *Journal of Scheduling*, 4, 201–207, 2001. (21)

101. László Lovász. Subgraphs with prescribed valencies. *Journal of Combinatorial Theory*, 8, 391–416, 1970. (69)

102. László Lovász et Michael D. Plummer. *Matching Theory*. Akadémiai Kiadó, Budapest, 1986. (27, 69–70)

103. R. McNaughton. Scheduling with deadlines and loss functions. *Management Sciences*, 6, 1–12, 1959. (42)

104. Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 1992. (130–132, 135, 137)

105. Henri Michaux. *Lointain intérieur*. 1938.

106. Wil Michiels, Jan Korst, Emile Aarts et Jan van Leeuwen. Performance ratios for the differencing method applied to the balanced number partitioning problem. Dans *Proceedings 20th International Symposium on Theoretical Aspects of Computer Science (STACS'03)*, pages 583–595. LNCS 2607, 2003. (119)

107. Katta G. Murty. *Linear programming*. John Wiley & Sons, Inc, 1983. (66)

108. Young-Soo Myung. An efficient algorithm for the ring loading problem with integer demand splitting. *SIAM Journal on Discrete Mathematics*, 14(3), 291–298, 2001. (27)

109. Vladimir Nabokov. *Tyrants Destroyed*. 1975.

110. Friedrich Nietzsche. *Also sprach Zarathustra*. 1885.

111. Öystein Øre. Graphs and subgraphs. *Transactions of the American Mathematical Society*, 84, 109–136, 1957. (69)

112. Paul Robert. *Petit Robert 1*, 1988. (14, 181)

113. Sartaj K. Sahni. Algorithms for scheduling independent tasks. *Journal of the Association for Computing Machinery*, 23, 116–127, 1976. (21)

114. Alexander Schrijver, Paul Seymour et Peter Winkler. The Ring Loading Problem. *SIAM Journal on Discrete Mathematics*, 11(1), 1–14, 1998. (27, 40, 187–188)

115. András Sebő. A Particular timetable problem: terminal scheduling. *Computers & Mathematics with applications*, 21(1), 137–156, 1991. (28)

116. Raymond Séroul. *Le petit livre de TEX*. Masson, 1996. (192)

117. David Simchi-Levi. New Worst Case Results for the Bin-Packing Problem. *Naval Research Logistics*, 41, 1994. (183)

118. The SONET Home Page. http://www.sonet.com. (25)

119. Patrick Soriano, Christelle Wynants, René Séguin, Martine Labbé, Michel Gendreau et Bernard Fortz. Design and Dimensioning of Survivable SDH/SONET Networks. Dans Brunilde Sansò et Patrick Soriano (ed.), *Telecommunications Network Planning*, chapitre 9, pages 147–167. Kluwer, Norwell, MA, 1998. (25)

120. Alain Sutter, François Vanderbeck et Laurence A. Wolsey. Optimal Placement of Add/Drop Multiplexers: Heuristic and Exact Algorithms. *Operations Research*, 46(5), 719–728, 1998. (27)

121. William T. Tutte. The factors of graphs. *Canadian Journal of Mathematics*, 4, 314–328, 1952. (69)

122. Tsong-Ho Wu. *Fiber Network Service Survivability.* Artech House, Inc., 1992. (25)

123. Jian Yang et Joseph Y.-T. Leung. The Ordered Open-End Bin-Packing Problem. *Operations Research*, 51(5), 759–770, 2003. (21)

124. M. Yue. A simple proof of the inequality $MFFD(L) \leq \frac{71}{60}OPT(L) + \frac{78}{71}, \forall L$ for the MFFD bin-packing algorithm. Rapport technique RRR # 20-91, RUTCOR, Rutgers University, New Jersey, 1991. (183)

Pierre Lemaire — Rangement d'objets multiboîtes

Thèse préparée au sein de l'équipe Recherche Opérationnelle du laboratoire Leibniz-IMAG (UMR CNRS 5522), 46 Avenue Félix Viallet, 38031 GRENOBLE Cedex.

* * *

Rangement d'objets multiboîtes

Résumé : Un objet multiboîte est composé de plusieurs parties identiques qui doivent être rangées dans des boîtes différentes. La hauteur d'une boîte est alors la somme des hauteurs des objets qu'elle contient. Ce concept généralise et englobe de nombreux problèmes de la littérature de la recherche opérationnelle.

Une classification de ces modèles est proposée. Les bases théoriques sont posées. En particulier, la complexité pour les principaux types d'objets et objectifs est déterminée.

Une étude détaillée est effectuée lorsque les objets ont des largeurs constantes et que l'on veut minimiser la hauteur de la boîte la plus haute. Des bornes inférieures, des heuristiques avec de très bonnes garanties de performance et un algorithme génétique sont proposés pour résoudre ce modèle. Leurs comportements théoriques et expérimentaux sont analysés.

Mots clés : rangement (*bin-packing*), ordonnancement, algorithmes d'approximation, algorithmes génétiques

* * *

Packing of Multibin Objects

Abstract: A multibin object is made of several identical parts; each of them must be packed into a different bin. The height of a bin is the sum of the heights of the objects packed into it. This concept generalizes some well-studied problems of the operations research literature.

A classification of multibin models is provided. Their theoretic bases are studied. In particular, the complexity for the main object types and objectives is investigated.

A detailed study is carried out when objects are of constant width and the objective is to minimize the height of the highest bin. Lower bounds, fast heuristics with very good performance garantees and a genetic algorithm are proposed to solve this model. The theoretic and experimental behaviors of these algorithms are analyzed.
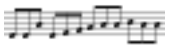
Key words: bin-packing, scheduling, approximation algorithms, genetic algorithms.

