Résolution séquentielles et parallèles des problèmes de découpe / placement

Toufik Saadi

Université de Paris 1 Panthéon-Sorbonne

THÈSE DE DOCTORAT EN INFORMATIQUE

Présentée et soutenue publiquement

par

Toufik SAADI

le 20 Novembre 2008

RÉSOLUTION SÉQUENTIELLE ET PARALLÈLE DES PROBLÈMES DE DÉCOUPE/PLACEMENT

Jury :

M. Alain CHATEAUNEUF, *Professeur à l'Université Paris 1 Panthéon-Sorbonne*	
M. Didier EL BAZ, *Chargé de recherche (HDR) au LAAS de Toulouse*	
M. Aristotelis GIANNAKOS, *MCF à l'Université de Picardie Jules Verne*	
M. Vassilios GIAKOUMAKIS, *Professeur à l'Université de Picardie Jules Verne*	
M. Pascal GOURDEL, *Professeur à l'Université Paris 1 Panthéon-Sorbonne*	*Président*
M. Mhand HIFI, *Professeur à l'Université de Picardie Jules Verne*	*Directeur*
M. Imed KACEM, *MCF (HDR) à l'Université de Technologie de Troyes*	*Rapporteur*
M. Ridha MAHJOUB, *Professeur à l'Université Paris 9 Dauphine*	*Rapporteur*

Cette thèse de doctorat est particulièrement dédiée

À mes chers parents,
À ma bien aimée leila et à ma petite lyza,
À mon cher frère,
À mes deux adorables soeurs,
À tous ceux qui ont contribué de près ou de loin à la réalisation et à l'aboutissement de ce travail.

Même s'il est d'usage de commencer par remercier son directeur de thèse, mes remerciements pour M. Mhand Hifi, professeur à l'université de Picardie Jules Verne, vont plus loin que les platitudes habituelles pour avoir accepté de diriger cette thèse. Je ne saurais être suffisamment reconnaissant pour son soutien, sa disponibilité, ses encouragements dans les moments de doute et son amitié tout au long de ces années.

Je remercie, tous particulièrement, Mm. Rym M'hallah, professeur à l'université de Koweït city. Sa passion pour la recherche m'a impressionné. J'ai été très honoré de travailler et de publier avec elle.

Je tiens à remercier tout particulièrement M. Pascal Gourdel, professeur à l'université de Paris-I Panthéon Sorbonne pour avoir accepté de présider le jury et pour l'intérêt qu'il porte à ce travail.

J'ai été très honoré que M. Ridha Mahjoub, professeur à l'université Paris Dauphine, ait accepté de rapporter ma thèse. Je tiens à lui exprimer ma profonde reconnaissance ainsi que mes plus vifs remerciements.

Un immense merci à M. Imad Kacem, maître de conférence (HDR) à l'université de technologie Troyes, d'avoir accepter de rapporter cette thèse. Ces remarques et conseils m'ont permis d'améliorer la qualité de ce rapport.

J'ai eu la chance d'être l'étudiant en DEA de M. Vassilios Giakoumakis, professeur à l'université de Picardie Jules Verne. J'ai apprécié pendant le DEA, sa pédagogie malgré la complexité du domaine de recherche. Je le remercie d'avoir accepter d'examiner ma thèse et d'être parmi les membres du jury.

Je remercie M. Didier El Baz, chargé de recherche (HDR) au Laboratoire d'Architectures et d'Analyse des Systèmes de Toulouse (LAAS), pour l'intérêt qu'il porte à mes travaux et pour avoir accepté de faire partie de ce jury.

Je suis très sensible à la présence dans ce jury de M. Alain Chateauneuf, professeur à l'université de Paris-I Panthéon Sorbonne. Je tiens à lui exprimer mes plus vifs remerciements.

Je remercie également, M. Aristotelis GIANNAKOS pour l'intérêt qu'il porte à mes travaux de recherche. Je tiens à lui exprimer mes profonds remerciements.

J'ai eu le plaisir de rencontrer à plusieurs reprises, M. Moussa Elkihal, chargé de recherche au Laboratoire d'Architectures et d'Analyse des Systèmes de Toulouse (LAAS), lors des conférences de la société française de recherche opérationnelle et d'aide à la décision (ROADEF). Sa passion pour le parallélisme et le calcul scientifique ma impressionné. J'ai été très honoré par sa présence pendant ma soutenance. Je tiens à lui exprimer ma profonde reconnaissance pour ces conseils et orientations.

J'ai eu le plaisir et la chance de rencontrer lors de la conférence IWOR, M. Ramon Alvares-Valdes, professeur à l'université de Valancia. Sa passion pour les problèmes de découpe et la qualité de ces travaux sur ce problème sont exceptionnelles. Je le remercie d'avoir repondu à chaque un de mes mails pendant ces années de thèse et de sa disponibilité.

Je tiens également à remercier, M. Marc Tissot, directeur pédagogique à l'IFCP. Sa grande culture, sa grande expérience dans l'enseignement et sa curiosité scientifique m'ont permis d'apprendre à adapter mes enseignements aux besoins des étudiants et d'être à leur écoute. Je remercie l'équipe pédagogique de l'IFCP pour leur sympathie, disponibilité et amitié. Travailler avec eux été un vrai plaisir.

Je remercie, Mm. Christine Lambrechts, responsable de formation à l'ITIN, pour m'avoir donné l'occasion et la chance d'enseigner dans une grande école comme l'ITIN. La très bonne organisation de la formation et l'exceptionnel niveau m'a permis de monter en compétence dans les métiers des systèmes et réseaux informatiques et d'avoir une vision très claire de ces métiers dans les entreprises.

Je remercie, Mm. Odile Piton, maître de conférence à l'université de Paris-I Panthéon Sorbonne, pour sa sympathie et sa disponibilité.

Je tiens aussi à remercier tous les membres du laboratoire CERMSEM UMR-CNRS 8095 permanents ou thésards (et ex-), avec lesquels j'ai passé des années sympathiques. Qu'ils me pardonnent de ne pas les avoir tous nominativement cités.

Une pensée à Marie-Lou pour le formidable travail qu'elle accomplie au sein du laboratoire CERMSEM.

Résumé

Les problèmes de découpe et de placement sont des problèmes combinatoires. Ils sont classés dans la catégorie des problèmes NP-Complets et admettent de nombreuses applications en industrie, en systèmes multiprogrammes et multiprocesseurs. Ces problèmes se présentent lorsqu'on se propose d'optimiser l'utilisation d'une ou de plusieurs entités disponibles en y plaçant des sous entités prédéterminées. Dans le cas où l'on dispose d'une seule entité en stock, ces problèmes sont connus sous le nom de problèmes de découpe (non) contraints (non) pondérés à deux dimensions. Nous proposons dans cette thèse, plusieurs méthodes de résolution exactes et approchées, séquentielles et parallèle du problème de découpe et de placement à deux dimensions. La méthode exacte proposée s'appuie sur une méthode de séparation et évaluation et opère par des constructions par pièces. Deux nouvelles bornes supérieures et inférieures sont proposées pour réduire l'espace de recherche. Nous proposons, également, plusieurs méthodes de résolution approchées. La première méthode combine une recherche par faisceaux et une construction par bandes. La construction par bandes génère un ensemble de bandes optimales, alors que la recherche par faisceaux filtre les meilleures bandes pendant la construction, en utilisant une fonction d'évaluation. L'influence de cette fonction sur la solution finale est évaluée en comparant une recherche par faisceaux locale (LBS) et une recherche par faisceaux globale (GBS). LBS implémente une politique d'évaluation locale de la solution obtenue pendant la construction alors que GBS fait une évaluation qui inclue la solution courante et une estimation de la solution (des solutions) générée(s) par les futures constructions. La deuxième méthode approchée est une méthode coopérative. Nous proposons un schéma de coopération entre l'algorithme GBS et une méthode complémentaire. La méthode complémentaire va contribuer à la découverte de nouvelles régions dans l'espace de recherche et à diversifier les chemins développés par GBS. La méthode complémentaire va aussi contribuer au calcul d'une borne supérieure plus fine qui va améliorer les performances de GBS. La troisième méthode proposée est une adaptation de GBS pour le problème de découpe à trois dimensions. La méthode est basée sur la résolution approchée de plusieurs problèmes de découpe à deux dimensions. Finalement, nous proposons, deux méthodes de résolutions approchées parallèles pour le problème de découpe à deux dimensions. La première méthode parallèle proposée (PGBS) s'appuie sur le paradigme maître/esclave et les spécificités de la recherche par faisceaux. Nous appliquons les mêmes mécanismes d'évaluation et de réduction de l'espace de recherche que ceux utilisés par GBS, tout en exploitant le parallélisme pour élargir l'espace de recherche et diversifier la recherche d'une solution approchée. De la même manière, la deuxième méthode présente une amélioration de PGBS. L'algorithme présenté améliore le schéma de coopération entre les processeurs et introduit une procédure complémentaire qui affine la recherche et améliore l'efficacité du parallélisme.

Abstract

Several real life industrial applications require the allocation of a set of rectangular items (or pieces) to a larger rectangular standardized stock unit. For instance, in the glass, plastic or metal industries, rectangular components have to be cut from a large sheet of material. Similarly, when filling the pages of a newspaper, the editor has to arrange articles and advertisements, presented as rectangular areas, on pages of fixed dimensions. Generally, industrial applications require cutting or packing the largest number of items/pieces into a rectangular unit as to minimize the waste of the rectangular stock unit. In this thesis, we propose exact, approximate and parallel algorithms for the Two-Dimensional Cutting stock Problem. The exact algorithm is a branch-and-bound that uses efficient lower and upper bounding schemes. It starts with a lower bound reached by the approximate two-stage algorithm. At each internal node of the branching tree, a tailored upper bound is obtained by solving (relaxed) knapsack problems. To speed up the branch and bound, we implement, in addition to ordered data structures of lists, symmetry, duplicate, and non-feasibility detection strategies which fathom some unnecessary branches. The first approximate algorithm combines a strip-generation procedure with beam search (BS). beam search is a truncated branch-and-bound which investigates a subset of elite nodes and permanently fathoms the others. It chooses the elite nodes based on an evaluation operator. The importance of the evaluation operator on the performance of BS is highlighted by comparing a local beam search (LBS) that uses a priority cost-evaluation operator to a global beam search (GBS) that adopts a global cost-evaluation operator. The second approximate algorithm uses a search strategy and a fast filling procedure. The search strategy uses a beam search method and the second strategy uses a local filling procedure for improving the quality of the obtained solutions. The third algorithm applies an adaptation of global beam search strategy to solve the Container Loading Problem. We propose, also, two parallel algorithms for approximately solving large-scale twos-staged two-dimensional cutting problem. The fist solution procedure, tries to explore in parallel a subset of elite nodes following the master-slave paradigm. The master processor serves to guide the search-resolution and each slave processor develops its proper way and trying a global convergence. The second algorithm considers three key features : a search strategy based on beam search, a fast filling procedure based on strip generation, and a tighter upper bound applied for refining the selected paths. The algorithm explores, in parallel, a subset of elite nodes following the master-slave paradigm. The master processor guides the search-resolution by maintaining a global list while each slave processor develops its own path according to its internal lists and an internal processor that completes the global paths.

Table des matières

Introduction

Les problèmes de l'optimisation combinatoire font souvent appel à des méthodes d'énumération implicite, de programmation dynamique, de programmation linéaire ainsi qu'à des méthodes d'intelligence artificielle. En général, la recherche arborescente est le point principal qui rapproche ces techniques de résolution.

L'ordre de recherche considéré lors de la résolution est fondé sur la visite de tous les sommets créés afin d'extraire une meilleure solution locale au fur et à mesure que le développement de l'arborescence s'effectue. L'évaluation d'une fonction objectif consiste à comparer la solution en cours à des bornes inférieures et supérieures.

Les problèmes de découpe et de placement sont des problèmes combinatoires. Ils sont classés dans la catégorie des problèmes NP-complets et admettent de nombreuses applications en industrie, en systèmes multiprogrammes et multiprocesseurs, ainsi que dans le placement de circuits intégrés. Ces problèmes se présentent lorsqu'on se propose d'optimiser l'utilisation d'une ou de plusieurs entités disponibles en taille limitée en y plaçant des sous entités pré-déterminées. Dans le cas où l'on dispose d'une seule entité en stock, ces problèmes sont connus sous le nom de problèmes de découpe (non) contraints (non) pondérés à deux dimensions.

Dans d'autres cas, l'utilisation de toutes les sous entités peut être indispensable. Ce sont les problèmes de strip packing ou bin packing, pour lesquels l'entité disponible peut se présenter sous forme d'une bande de largeur fixée et de longueur illimitée. En revanche, si l'on possède une quantité limitée d'entités en stock et d'un ensemble de sous entités limité par une borne supérieure, alors ce problème se généralise au problème du carnet de commandes.

Le problème de découpe a été posé par Kantorovich ([27],1960) afin de trouver une modélisation cohérente des problèmes reliant l'organisation et la théorie de la production. Il fut repris par Gilmore et Gomory([61],1961-1963) pour la résolution du problème de découpe à une et à deux dimensions en utilisant des méthodes de la programmation linéaire (pour une résolution approchée). Les mêmes auteurs ([62],[63] ,1965-1966) l'ont généralisé aux problèmes de découpes à deux et à plusieurs dimensions en s'appuyant sur d'autres méthodes de résolution exactes. Ces deux auteurs ont montré l'intérêt commercial et industriel de ces méthodes. Peu après, ce problème a été utilisé par Cood[7] pour l´étude des systèmes multiprogrammes et par Garey et Graham[55] pour les systèmes multiprocesseurs. Par la suite, une généralisation du problème a été proposée pour l'interprétation d'un ensemble de variétés de problèmes industriels (papier, verre, cuir, bois).

Dans la pratique, l'utilisateur est contraint par le type de matériel utilisé pour réaliser un plan de découpe, par conséquence, la recherche d'une solution au problème diffère selon l'outil considéré : outil guillotine, outil non guillotine ou bien outil non orthogonal.

Un autre cas plus difficile se présente lorsque la surface du rectangle (ou des rectangles) est munie de certains points défectueux (surface non utilisable). Hahn [76] a traité ce problème.

Un autre problème de découpe qui semble intéressant est le problème de découpe circulaire (non) contraint. Les méthodes utilisent soit, une recherche gloutonne, soit une recherche locale, mais dans le cas circulaire plusieurs auteurs ont discuté la difficulté d'une éventuelle modélisation pour ce problème.

En général, dans les problèmes de découpe, on associe à chaque sous entité un coût (ou un profit) de production. La fonction objectif peut être considérée comme étant linéaire ou non linéaire, selon les différents aspects du problème. Les principaux objectifs à optimiser sont donnés par la minimisation de la surface perdue, minimisation des entités nécessaires à la satisfaction des commandes, minimisation du coût de manipulation et de stockage ainsi que le maximum de répétitions de l'ordonnancement des commandes. La considération simultanée de ces objectifs conduit souvent à une fonction économique non linéaire. Nous nous intéressons principalement dans cette thèse au cas où la fonction économique est linéaire.

Dans la même famille des problèmes de découpe, on trouve le problème d'assemblage de pièces rectangulaires dans une surface minimum. En général, on associe à chaque pièce un profit qui correspond exactement à sa surface et une borne supérieure sur le nombre d'occurrences pour chaque type de pièces dans la solution. En particulier, on peut s'intéresser au cas où la surface d'assemblage est donnée sous forme d'un rectangle, comme on peut le généraliser au cas d'une surface minimum (convexe) en utilisant toutes les rotations possibles sur les pièces rectangulaires.

Dans cette thèse, nous étudions le cas où la surface d'assemblage est présentée sous forme rectangulaire horizontale ou verticale minimum.

Après les deux premiers chapitres consacrés à la présentation des problèmes de découpe et de placement (chapitre 1) et à la description de quelques méthodes de résolution (chapitre 2), notre contribution comporte trois parties. La première partie décrit l'étude d'une variante du problème de découpe (le problème de découpe contraint à deux dimensions à deux niveaux) en se basant sur les méthodes de résolution séquentielles approchées. Nous présentons également, dans cette partie, une adaptation de cette méthode au problème de chargement/remplissage. La deuxième partie entreprend la résolution exacte d'une autre variante du problème de découpe (le problème de découpe double contraint à deux dimensions) en s'appuyant sur une procédure de séparation et évaluation. La dernière partie est consacrée à la résolution parallèle approchée du problème de découpe contraint à deux dimensions à deux niveaux.

Plus précisément, la première partie comporte trois chapitres. Le premier chapitre présente une méthode approchée pour le problème de découpe contraint à deux dimensions à deux niveaux

(Constrained Two-Staged Two-Dimensional Cutting Problem). Nous proposons un algorithme combinant une recherche par faisceaux et une construction par bandes. La construction par bandes génère un ensemble de bandes optimales ; alors que la recherche par faisceaux filtre les meilleures bandes pendant la construction, en utilisant une fonction d'évaluation. L'influence de cette fonction sur la solution finale est évaluée en comparant une recherche par faisceaux locale (LBS) et une recherche par faisceaux globale (GBS). LBS implémente une politique d'évaluation locale de la solution obtenue pendant la construction alors que GBS fait une évaluation qui inclue la solution courante et une estimation de la solution (des solutions) générée(s) par les futures constructions. Le deuxième chapitre présente une méthode coopérative approchée pour le même problème (le problème de découpe contraint à deux dimensions à deux niveaux). Nous proposons un schéma de coopération de l'algorithme GBS et une méthode complémentaire. La méthode complémentaire va contribuer à la découverte de nouvelles régions dans l'espace de recherche et à diversifier les chemins développés par GBS. La méthode complémentaire va aussi contribuer au calcul d'une borne supérieure plus fine qui va améliorer les performances de GBS. Le troisième chapitre est consacré au problème de chargement/remplissage. Après une présentation du problème et de ses domaines d'application, nous proposons une méthode de résolution approchée. La méthode est basée sur la résolution approchée de plusieurs problèmes de découpe à deux dimensions.

La deuxième partie comporte un chapitre qui présente un algorithme exact pour le problème de découpe double contraint à deux dimensions (Double-Constrained Two-Dimensional Guillotine Cutting Stock Problem). L'algorithme s'appuie sur une méthode de séparation et évaluation et opère par des constructions par pièces. Deux nouvelles bornes supérieures et inférieures sont proposées pour réduire l'espace de recherche.

La dernière partie est consacrée aux méthodes de résolution parallèle. Elle est composée de deux chapitres. Le premier chapitre introduit quelques notions de parallélisme et le deuxième chapitre présente deux algorithmes parallèles pour le problème de découpe contraint à deux dimensions et à deux niveaux. La première méthode parallèle proposée (PGBS) s'appuie sur le paradigme maître/esclave et les spécificités de la recherche par faisceaux. Nous appliquons les mêmes mécanismes d'évaluation et de réduction de l'espace de recherche que ceux présentés dans le chapitre 3, tout en exploitant le parallélisme pour élargir l'espace exploré et diversifier la recherche d'une solution approchée. De la même manière, la deuxième méthode présente une amélioration de PGBS. L'algorithme présenté améliore le schéma de coopération entre les processeurs et introduit une procédure complémentaire qui affine la recherche et améliore l'efficacité du parallélisme.

Enfin, le chapitre 9 résume les travaux effectués au cours de cette thèse et propose une ouverture sur de nouvelles directions de recherche.

Première partie

QUELQUES PROBLÈMES COMBINATOIRES ET MÉTHODES DE RÉSOLUTION

Chapitre 1

Les problèmes de découpe et de placement

Contenu

Nous commençons dans ce chapitre, par la présentation de quelques définitions préliminaires et propriétés sur lesquelles se basent les modélisations et résolutions des problèmes de découpe non contraint, contraint et double contraint à deux dimensions. Ensuite, nous faisons une interprétation du problème de sac à dos à une et à deux dimensions et le rapport existant entre celui-ci et les problèmes de découpe.

1.1 Problèmes de découpe

Nous présentons dans cette section les différentes variantes de problèmes de découpe traitées dans cette thèse, ainsi qu'une vue plus générale sur d'autres problèmes qui restent toujours difficiles à résoudre, dans le sens de la recherche et du développement des algorithmes exacts de résolution.

Nous définissons l'ensemble des variétés des bandes exploitées par nos modélisations et types de matériels utilisés pour résoudre le problème de découpe.

1.1.1 Problème de découpe non contraint

1.1.1.1 Problème non pondéré

Une instance du problème de découpe non contraint non pondéré à deux dimensions est définie par le couple (R, S). $R = (L, W)$ est l'entité disponible représentée par le rectangle initial de longueur L et de hauteur W. L'ensemble des sous-entités correspond à un ensemble de pièces rectangulaires représenté par S tel que $|S| = n$. Chaque type de pièces est représenté par des petites dimensions (l_i, w_i), pour $i = 1, ..., n$, où l_i (resp.w_i) désigne la longueur (resp. hauteur) de la pièce i.

Soient $\mathcal{P}$ l'ensemble fini des vecteurs représentant les différents plans de découpe réalisables, $\xi = (\xi_1, ..., \xi_n)$ avec ξ_i le nombre de répétitions de la ième pièce dans le plan ξ et F une application définie par :

$$F : \mathcal{P} \longrightarrow N, \text{tel que}, F(\xi) = LW - \sum_{i=1}^{n} \pi_i \xi_i.$$

où π_i désigne la surface de la pièce $i = 1, ..., n$ (la surface de la pièce i est donnée par $\pi_i = l_i w_i$).

Une solution du problème consiste à trouver le couple (ξ, F) tel que $\xi \subset \mathcal{P}$

Une solution optimale du problème est le couple $(\xi^*, F(\xi))$ tel que le profit du plan de découpe réalisable ξ^* soit égale à F^* de valeur minimum (i.e. un plan de découpe optimal qui minimise la chute sur le rectangle initial).

1.1.1.2 Problème pondéré

Une instance du problème de découpe pondéré non contraint à deux dimensions est définie par le triplet (R, S, c). R et S ont la même signification que dans le cas du problème non pondéré. En revanche, ce problème diffère du précédent par l'introduction du vecteur profit $c = (c_i)$, $i = 1, ..., n$, où à chaque pièce i on fait correspondre un profit c_i ($c_i \neq l_i w_i$), pour $i = 1, ..., n$. L'application précédente peut s'écrire cette fois sous la forme suivante :

$$F : \mathcal{P} \longrightarrow N, \text{tel que}, F(\xi) = \sum_{i=1}^{n} c_i \xi_i$$

Dans ce cas, le but est de trouver le couple (ξ^*, F^*) qui maximise la fonction d'utilité F.

1.1.2 Problème de découpe pondéré contraint

Une instance du problème de découpe contraint est définie par le quadruplet (R, S, c, b). R représente le rectangle initial, S l'ensemble des pièces, c le vecteur profit associé aux pièces et $b = (b_i)$, $i = 1, ..., n$ une borne supérieure (appelée aussi demande) pour le nombre d'occurrences de chaque type de pièces dans un plan de découpe réalisable.

Ainsi, pour plus de clarté, ce problème peut être formulé par le programme suivant :

$$\text{Max } F = \sum_{i=1}^{n} c_i \xi_i,$$
$$\xi_i \leq b_i, \xi_i \in N \text{ pour } i = 1, ...n, \xi \in \mathcal{P}$$

où ξ_i est le nombre d'occurrences de la pièce du type i dans le plan de découpe réalisable $\xi \in \mathcal{P}$ contraint par la borne supérieure bi.

De la même manière, une solution optimale pour ce problème est représentée par le couple (ξ^*, F^*), tel que ξ^* donne le meilleur plan de découpe de valeur maximum F^* sous les contraintes $\xi_i \leq b_i$, pour $i = 1, ..., n$.

1.1.3 Problème de découpe pondéré double contraint

Une instance du problème de découpe double contraint est définie par (R, S, c, a, b). R représente le rectangle initial, S l'ensemble des pièces, c le vecteur profit associé aux pièces, $a = (a_i) i = 1, ..., n$ et $b = (b_i) i = 1, ..., n$ représentent respectivement une borne inférieure et une supérieure pour le nombre d'occurrences de chaque type de pièce dans un plan de découpe réalisable.

De la même manière, ce problème peut être formulé par le programme suivant :

$$\text{Max } F = \sum_{i=1}^{n} c_i \xi_i, \xi_i \leq b_i,$$
$$\xi_i \geq a_i, \xi_i \in N \text{pour } i = 1, ...n, \xi \in \mathcal{P}$$

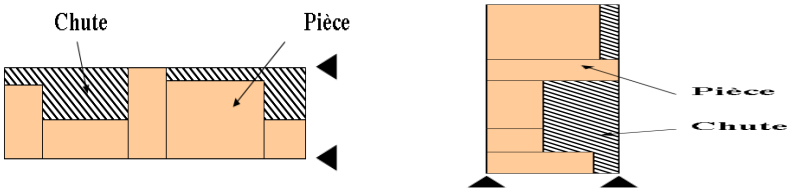
Une solution optimale pour ce problème est représentée par le couple (ξ^*, F^*), tel que (ξ^* donne le meilleur plan de découpe de valeur maximum F^* sous les contraintes $\xi_i \leq b_i$ et $\xi_i \geq a_i$ pour $i = 1, ..., n$.

Lorsque la contrainte $a_i = 0, i = 1, \ldots, n$, le problème est équivalent au problème contraint et que lorsque $a_i = b_i, i = 1, \ldots n$ le problème devient le problème de découpe avec satisfaction exacte des demandes.

1.1.4 Définitions

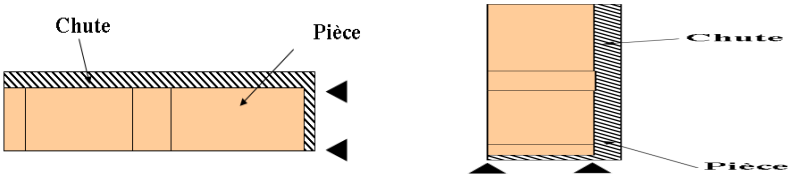
Pour le moment on suppose que l'on possède une plaque rectangulaire sur laquelle on veut effectuer une dissection afin de produire un ensemble de pièces appartenant à S (placement des pièces de dimensions inférieures aux dimensions de la plaque rectangulaire utilisée).

Nous présentons, dans cette section, une vue générale sur les bandes et les modèles de découpes utilisés dans les chapitres suivants.



(a). Bande générale horizontale. (b). Bande générale verticale.

FIGURE 1.1 – Bandes générales horizontale et verticale.

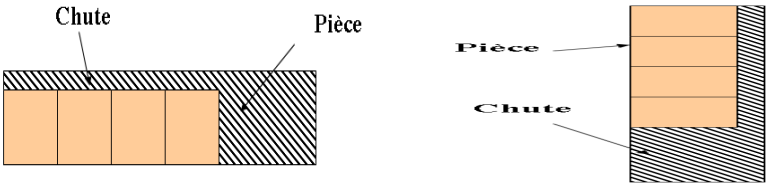


(a). Bande uniforme horizontale. (b). Bande uniforme verticale.

FIGURE 1.2 – Bandes uniformes horizontale et verticale.

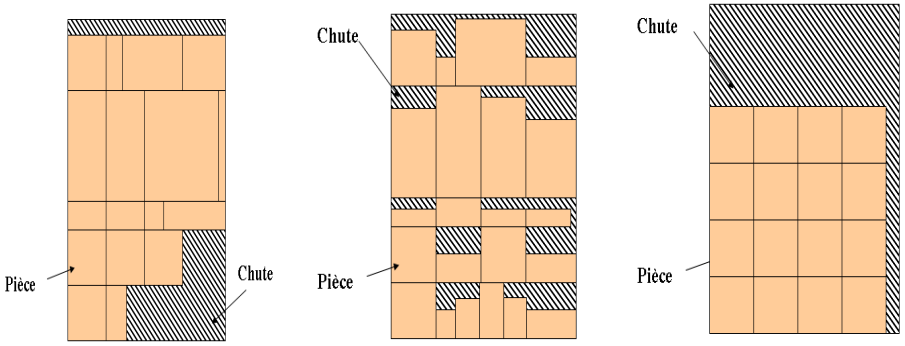
1.1.4.1 Les principaux modèles de bandes

1. Une bande générale horizontale (figure 1.1.(a)) (resp. verticale (figure 1.1.(b)) est un ensemble de pièces rectangulaires accotées de telle façon qu'il existe une ligne horizontale (resp. verticale) ayant les propriétés suivantes :
   - elle touche toutes les pièces sur leur longueur (resp. hauteur).
   - un des demi-plans défini par celle-ci contient toutes les pièces.
2. Une bande uniforme est une bande pour laquelle existe exactement deux lignes horizontales (resp. verticales) ayant les propriétés précédentes. Dans ce cas, toutes les pièces découpées de la bande sont de même hauteur (resp. longueur) (figure 1.2.(a) et 1.2.(b)).
3. Une bande homogène horizontale (resp. verticale) est une bande uniforme où il n'y a qu'un seul type de pièces participant (figure 1.3.(a) et 1.3.(b)).
4. Une bande optimale de longueur l et de hauteur w (resp. de hauteur w et de longueur l) est une bande générale occupant la surface maximum de la bande (l, w) (resp. (w, l)).



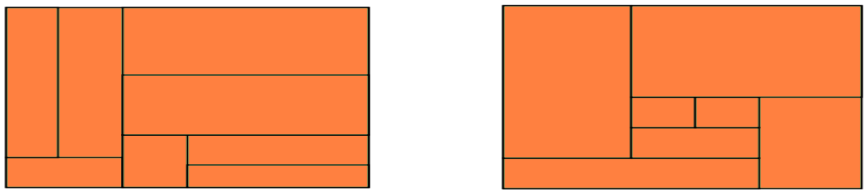
(a). Bande homogène horizontale. (b). Bande homogène verticale.

FIGURE 1.3 – Bandes homogènes horizontale et verticale.



(a). Uniforme. (b). Général. (c). Homogène.

FIGURE 1.4 – Modèles de découpe uniforme, générale et homogène.



(a). Modèle de découpe guillotine. (b). Modèle de découpe non guillotine.

FIGURE 1.5 – Modèles de découpe guillotine et non guillotine.

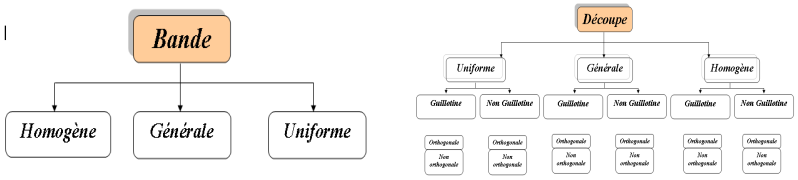
1.1.4.2 Les principaux modèles de découpes

1. Un modèle de découpe uniforme (ou dissection uniforme) est un modèle constitué par la combinaison de bandes uniformes verticales et horizontales (figure 1.4.(a)).
2. Un modèle de découpe générale (ou dissection générale) est un modèle constitué par la combinaison de bandes générales verticales et horizontales (figure 1.4.(b)).
3. Un modèle de découpe homogène est une dissection uniforme pour laquelle la solution est caractérisée par la répétition d'un seul type de pièces de l'ensemble S (figure 1.4.(c)).

1.1.4.3 Les différents types de découpe

En pratique, les utilisateurs sont souvent contraints par le matériel de découpe disponible, i.e. la façon de procéder pour la réalisation d'un plan de découpe est différente. Les outils fréquemment utilisés sont :

1. ***La découpe du type guillotine :*** qui réalise une production d'un ensemble de sous-entités dans une entité disponible. Si on suppose que l'entité est une plaque rectangulaire, alors la découpe est effectuée par la dissection en allant d'un coté à son opposé parallèlement aux deux autres (figure 1.5).
2. ***La découpe du type non-guillotine :*** en général cette découpe engendre une solution meilleure que celle réalisée par les découpes du type guillotine. En effet, cette découpe consiste



(a). Modèles de bandes. (b). Modèle de découpe.

FIGURE 1.6 – Classification des modèles de découpe et de bandes.

à utiliser le même procédé que dans la découpe guillotine et de plus, elle peut être effectuée tout en marquant des arrêts avant d'atteindre le côté opposé du (sous-) rectangle à découper (figure 1.5).

3. ***La découpe non-orthogonale :*** cette découpe ne prend pas en considération l'orientation des pièces. Cette fois les pièces peuvent être pivotées et translatées (on effectue des rotations sur les pièces, donc elles ne sont pas fixées).

1.1.4.4 Niveau de découpe guillotine

Le niveau de découpe guillotine est une constante K qui limite le nombre de changements de la découpe. On parle alors, de problème de découpe à deux niveaux si K est fixé à deux, c'est-à-dire le nombre de changements de la découpe est limité à deux.

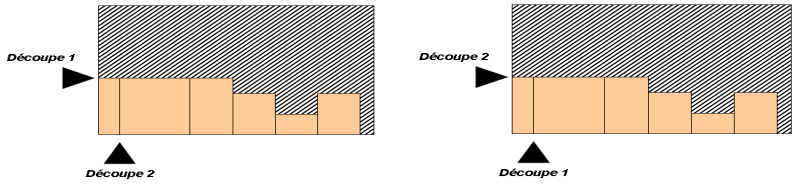
1.2 Les hypothèses posées sur les différents problèmes

On a vu précédemment les différents types de découpes que l'on peut utiliser. Dans notre étude, l'ensemble des algorithmes proposés résolvent les différentes variantes des problèmes décrits ci-dessus sous certaines hypothèses données dans l'ordre suivant :

1. D'une part, on suppose que les découpes effectuées sur le rectangle initial, pour les différents problèmes précédents, sont du type guillotine et que les pièces de l'ensemble S sont fixées (i.e. la pièce de longueur l et de hauteur w n'est pas la même que celle qui possède la longueur w et la hauteur l).
2. D'autre part, on suppose que les dimensions L, W, l_i et w_i, pour $i = 1, ..., n$, sont des entiers.

1.3 Problème de découpe contraint à deux dimensions et à deux niveaux

Une instance du problème de découpe contraint à deux dimensions est définie par le quadruplet (R, S, c, b). R représente le rectangle initial, S l'ensemble des pièces, c le vecteur profit associé aux pièces et $b = (b_i)i = 1, ..., n$ une borne supérieure pour le nombre d'occurrences de chaque type de pièces dans un plan de découpe réalisable.



(a). Horizontale. (b). Verticale.

FIGURE 1.7 – Plans de découpe horizontale et verticale.

1. La production d'une pièce dans ce problème se fait par deux niveaux de découpes guillotines (une première horizontale et une deuxième verticale dans l'ordre).
2. Un plan de découpe réalisable horizontale est une solution obtenue d'un processus de découpe commençant par une découpe horizontale (figure 1.7.(a)).
3. Un plan de découpe réalisable verticale est une solution obtenue d'un processus de découpe commençant par une découpe verticale (figure 1.7.(b)).
4. Toutes les pièces de l'ensemble S sont fixées (i.e. la pièce de longueur l et de hauteur w n'est pas la même que celle qui possède la longueur w et la hauteur l).
5. Les dimensions L, W, l_i et w_i, pour $i = 1, ..., n$, sont des entiers positifs.

1.4 Problème de découpe double contraint à deux dimensions

Une instance du problème de découpe double contraint à deux dimensions est définie par (R, S, c, a, b).
R représente le rectangle initial, S l'ensemble des pièces, c le vecteur profit associé aux pièces et $a = (a_i)i = 1, ..., n$ et $b = (b_i)i = 1, ..., n$ représentent respectivement une borne inférieure et supérieure pour le nombre d'occurrences de chaque type de pièces dans un plan de découpe réalisable.

1. Un plan de découpe réalisable est produit par des découpes guillotines successives.
2. Toutes les pièces de l'ensemble S sont fixées (i.e. la pièce de longueur l et de hauteur w n'est pas la même que celle qui possède la longueur w et la hauteur l).
3. Les dimensions L, W, l_i et w_i, pour $i = 1, ..., n$, sont des entiers positifs.

1.5 Quelques problèmes combinatoires

Notre intérêt s'est porté sur les deux problèmes précédents, cependant, la compréhension des problèmes suivants est nécessaire.

1.5.1 Problème de sac-à-dos

Ce problème a été introduit par Gilmore et Gomory [62, 63] pour la modélisation et la résolution du problème de découpe à une dimension. La fonction utilisée par les deux auteurs prend le nom

de la fonction knapsack définie par :

$$KP(c) = \begin{cases} \text{Maximiser} \mathcal{F}(x) = \sum_{i=1}^{n} c_i x_i \\ \text{s.c. } \sum_{i=1}^{n} l_i x_i \leq b \\ x_i \in N, \text{pour} i = 1, \ldots, n. \end{cases}$$

La résolution de ce type de problème s'effectue à l'aide de la programmation dynamique que nous verrons en détail dans le chapitre 2. Ce problème a été généralisé au problème de sac-à-dos à deux dimensions représenté par la fonction knapsack G définie de la façon suivante :
Si on possède un ensemble de pièces rectangulaires de dimensions (l_i, w_i), pour $i = 1, ..., n$ et un vecteur $c = (c_1, c_2, ..., c_n)$ associé aux différentes pièces, alors $G(x, y)$ est le maximum de la somme $c_1x_1 + c_2x_2 + ... + c_nx_n$, où $x_1, ..., x_n$ sont des entiers tels qu'on puisse partager le rectangle (x, y) en x_i rectangles (l_i, w_i), pour $i = 1, ..., n$.

Trouver une solution optimale satisfaisant la fonction knapsack $G(x, y)$ pour un x et un y, est un problème difficile (utilisation des découpes non-orthogonales et sans discrétisation sur la longueur et la hauteur du rectangle initial). Cependant, en pratique, la plupart des découpes utilisées se ramènent à calculer une autre fonction knapsack $\mathcal{F}$. Cette fonction est définie de la même manière que la fonction G à part que les découpes effectuées possèdent la particularité d'être du type guillotine. Nous verrons plus de détails dans le chapitre 2 où nous présentons l'adaptation de la fonction knapsack au problème de découpe contraint à deux dimensions.

1.5.2 Problème de strip packing

Le problème de strip packing est défini par le quadruplet (B, S, c, b). $B = (L, W)$ où L désigne la longueur de la bande (qui est supposée égale à l'infini) et W la largeur de cette bande. S constitue l'ensemble des pièces à découper sur la bande et les vecteurs $c = (c_i)i = 1, ..., |S|$ et $b = (b_i)i = 1, ..., |S|$ représentent respectivement les profits et les bornes supérieures sur chaque type de pièce de l'ensemble S. Ce problème peut être formulé comme suit :

Étant donné une instance du problème de strip packing et son ensemble des plans de découpe réalisables $\mathcal{P}$, le but est alors de trouver le plan de découpe optimal $\xi \in \mathcal{P}$ réalisant un profit maximum (pour le cas de la surface, le profit devient la minimisation des chutes et de la longueur d'utilisation de la bande), la fonction F est donnée par :

$$F(\xi) = \sum_{i=1}^{n} \pi_i \xi_i, \text{avec } \xi_i \leq b_i, \pi_i \in N, \text{pour } i = 1, ...n, \text{et } \xi \in \mathcal{P}$$

où ξ_i représente le nombre d'occurrences de la pièce i dans le plan de découpe ξ, et c_i le profit de la pièce i.

1.6 Conclusion

A travers une synthèse des propriétés des différents modèles et types de découpe, nous avons montré la diversité des problèmes de découpe. En effet, Dychhoff [32] explicite la relation entre les problèmes de découpe et les problèmes de placement (d'emballage). Les problèmes de découpe peuvent être considérés comme étant le placement des pièces dans des objets de grande taille. De la même manière, les problèmes de placement peuvent être considérés comme étant des problèmes de découpe ; ceci justifie le nom donné à ce type de problème, à savoir les problèmes de découpe et de placement "CP" (Cutting and Packing). En revanche, certaines contraintes liées aux applications industrielles sont exclusivement imposées dans le problème de découpe ou dans le problème de placement. Citons à titre d'exemple la contrainte guillotine imposée par l'outil de découpe. Ainsi, en raison de la diversité de ces problèmes et de leurs domaines d'application, de nombreuses variantes apparaissent sous différentes formes et sous différents noms dans la littérature. Néanmoins, une analyse minutieuse de ces problèmes montre l'existence de plusieurs aspects communs. Dans ce sens, il existe plusieurs classifications des problèmes de découpe et de placement dans la littérature, à savoir la classification présentée dans [16] par Dyckhoff et Finke, basée sur quatre aspects : le nombre de dimensions, le genre de tâches, l'assortiment de grands objets (plaques) et de petits objets (pièces). Une autre classification due à Hifi [33] ; s'appuyant sur les spécificités du support considéré, les types de pièces à placer et sur les contraintes de découpe considérées. Tous les problèmes décrits dans ces classifications ont été démontrés soit NP-difficiles, soit NP-complets. Il s'ensuit que seuls les problèmes de petites tailles sont solvables par les méthodes exactes dans un temps raisonnable. Une solution alternative permettant de résoudre les instances de grande taille consiste à utiliser les méthodes heuristiques. Celles-ci peuvent donner des solutions satisfaisantes en un temps raisonnable. Afin de mieux aborder la suite de cette thèse ; on propose dans le chapitre suivant, un survol des différents travaux réalisés sur les problèmes de découpe, ainsi qu'un rappel de quelques approches exactes et heuristiques pour les problèmes de découpe contraint qui ont servi comme point de départ à notre étude.

Chapitre 2

Travaux sur les problèmes de découpe et de placement

Contenu

Les problèmes de l'optimisation combinatoire font souvent appel à des méthodes d' énumération implicite, de programmation dynamique, de programmation linéaire ainsi qu'à des méthodes d'intelligence artificielle. En général, la recherche arborescente est le point principal qui rapproche ces techniques de résolution. L'ordre de recherche considéré lors de la résolution est fondé sur la visite de tous les sommets créés afin d'extraire une meilleure solution locale au fur et à mesure que le développement de l'arborescence s'effectue. L'évaluation d'une fonction objectif consiste à comparer la solution en cours à des bornes inférieures et supérieures.

Le problème de découpe est un problème combinatoire. Il est classé dans la catégorie des problèmes NP-difficiles et admet de nombreuses applications en industrie, en systèmes multiprogrammés et multiprocesseurs, ainsi que dans le placement de circuits intégrés. Ce problème se présente lorsque l'on se propose d'optimiser l'utilisation d'une(plusieurs) entité(s) disponible(s) en taille limitée en y plaçant des sous-entités prédéterminées. Une politique optimale de placement peut ne pas utiliser certaines de ces sous-entités. Dans d'autres cas, l'utilisation de toutes les sous-entités peut être indispensable comme dans les problèmes de strip packing, pour lesquels l'entité disponible peut se présenter sous forme d'une bande de largeur fixée et de longueur illimitée. Par contre, si l'on possède une quantité limitée d'entités en stock, et un ensemble de sous-entités limité par une borne supérieure, alors ce problème se généralise au problème du carnet de commandes. En général, dans les problèmes de découpe, on associe à chaque sous-entité un coût (ou un profit) de production. La fonction objectif peut être considérée comme étant linéaire ou non-linéaire, selon les différents aspects du problème. Les principaux objectifs à optimiser sont donnés par la minimisation de la surface perdue, minimisation des entités nécessaires à la satisfaction des commandes, minimisation

du coût de manipulation et de stockage ainsi que le maximum de répétitions de l'ordonnancement des commandes.

La considération simultanée de ces objectifs conduit souvent à une fonction économique non-linéaire. Nous nous intéressons principalement dans cette thèse au cas où la fonction économique est linéaire. Depuis son introduction par Kantorovich [27], le problème de découpe a fait l'objet de nombreux travaux dans ces différentes variantes. Dans ce chapitre, nous faisons un survol des différents travaux réalisés sur ce problème.

2.1 État de l'art

Le problème de découpe a été posé par Kantorovich [27] afin de trouver une modélisation cohérente des problèmes reliant l'organisation et la théorie de la production.

Il fut repris par Gilmore et Gomory [61] pour la résolution du problème de découpe à une et deux dimensions en utilisant des méthodes de la programmation linéaire (pour une résolution approchée). Les mêmes auteurs [62], [63] l'ont généralisé aux problèmes de découpe à deux et plusieurs dimensions s'appuyant sur d'autres méthodes de résolution exacte. Ces deux auteurs montraient l'intérêt commercial et industriel porté sur ces méthodes.

Peu après, ce problème a été utilisé par Codd [7] pour l'étude des systèmes multiprogrammés et par Garey et Graham [55] pour l'étude des systèmes multiprocesseurs.

Par la suite, une généralisation du problème a été proposée pour l'interprétation d'un ensemble de variétés de problèmes industriels (papier, verre, cuir, ...etc).

Dans ces problèmes, généralement, l'utilisateur est contraint par le type de matériel disponible pour une éventuelle réalisation d'un plan de découpe pour le problème implémenté. La recherche d'une solution au problème diffère selon l'outil considéré : outil non-guillotine outil non-orthogonal ou bien outil guillotine .

Beasley [24], s'est intéressé à l'étude du problème de découpe lorsque l'on dispose d'un matériel du type non-guillotine, ainsi que Daniels et Ghandforoush [21] qui ont développé un algorithme et qui ont montré son efficacité par rapport à l'algorithme de Beasley.

On connait peu de chercheurs qui se sont intéressés au cas du matériel non-orthogonal. Parmi ceux qui apparaissent dans la littérature, Haessler [73] par l'utilisation de la programmation non linéaire, Biro et Boros [30] en se servant d'une modélisation sous forme d'un réseau de flot, ainsi que Rinnoy kan, De wit et Wijmenga [4]. Un autre cas plus difficile se présente lorsque la surface du rectangle (ou des rectangles) est munie de certain points défectueux (surface non utilisable), ce problème a été traité par Hahn [76].

Par ailleurs, de nombreux auteurs se sont intéressés à l'étude du problème en prenant en compte le matériel guillotine, à titre d'exemple, Gilmore et Gomory [61], [62], [63], Haessler et Sweeney [74], [75], [60], Herz [20], Adamowicz et Albano [28], [29], Dyson et Gregory [70], Christofides et Whitlock [56], [57], De Cani [59], Wang [66], Beasley [22],[23], Vasko [10], Farley [2], [1], Kuntz et Uhry [15], Hadjiconstantinou et Christofides [58], Viswanathan et Bagchi [26], Carnieri, Mendoza et Gavinho [6], Blazewicz [19], Fayard et Zissimopoulos [8], [81].

Plusieurs approches exactes et heuristiques ont vu le jour pour le problème guillotine, ces dernières années. Nous nous sommes intéressés notamment à Hifi et al [34], [38],[39], [40], [43], Hifi et Roucairol [37], Lodi et Monaci [3], Morabito et Garcia [72] et Alvarez-Valdes et al [67].

Ces méthodes utilisent différentes approches, notamment, la programmation dynamique [34], [38], [39], [40], [43], la génération de colonnes [72] et des formulations en programme linéaire en nombres entiers [3].

Parmi les méthodes exactes ; Hifi a proposé [32] une amélioration de l'algorithme exact de Viswanathan et Bagchi's [26] pour le problème de découpe guillotine contraint à deux dimensions, il introduit le problème de sac-à-dos dans l'algorithme original et améliore l'algorithme en utilisant les propriétés de la résolution par programmation dynamique, ainsi que de nouvelles bornes supérieures et inférieures qui améliorent considérablement les troncatures dans l'algorithme original. Peu après et avec la même stratégie Hifi et Roucairol [37] ont proposé un algorithme d'approximation et un algorithme exact pour le même problème, l'algorithme exact est basé sur une stratégie de construction avec un retour arrière. Plus tard, Lodi et Monaci [3] ont proposé une formulation en programme linéaire en nombres entiers, la même année, Belov et Scheithauer [14] ont combiné les deux derniers algorithmes pour développer un algorithme branch and cut and price.

En ce qui concerne les méthodes approchées, Hifi et al [38] ont résolu le problème de découpe à deux dimensions en utilisant une approche hybride qui combine deux heuristiques :
- Une recherche en profondeur sur un arbre de recherche utilisant une technique de hill climbing. - Une procédure de programmation dynamique basée sur une résolution d'une série de problèmes de sac-à-dos à une dimension. La programmation dynamique permettait d'obtenir une borne inférieure initiale de bonne qualité qui est comparée à une borne supérieure pendant le développement de l'arborescence de recherche. Dans la même année, Hifi et M'Hallah [39] ont proposé, une méthode qui combine un algorithme glouton et une stratégie de hill climbing.
Ensuite, Alvarez-Valdes et al [67] ont proposé une approche qui combinait une construction par bandes et la stratégie GRASP, ainsi qu'une méthode Path Relinking.

Dans la suite de notre étude, nous nous pencherons sur le problème guillotine et tout particulièrement au problème de découpe contraint et double contraint à deux dimensions.

2.2 Heuristiques et méta-heuristiques

2.2.1 Les algorithmes par génération de bandes

Les algorithmes de génération de bandes (SGA) procèdent en deux étapes ; dans la première étape, ces algorithmes génèrent un ensemble de bandes. Et dans une deuxième étape ils cherchent une bonne combinaison de ces bandes.

Une bande (α, β) avec $0 < \alpha \leq L$ et $0 < \beta \leq W$, est obtenue en appliquant une découpe guillotine sur le rectangle initial (L, W). La première découpe peut être horizontale ou verticale. Si la première découpe est horizontale alors la bande générée est une bande horizontale (L, β). de la même manière, si la première bande est verticale alors la bande générée est verticale (α, W). Dans ce qui suit et sans perte de généralité, on considère que la première découpe est horizontale.

– **Génération de bandes uniformes :**
La première étape des algorithme SGA consiste à générer une bande uniforme $(L, \overline{w}_j)$ pour chaque hauteur $\overline{w}_j$, $j \in J, J = 1, ..., r$. Les bandes uniformes générées sont construites en fonction de la solution optimale du problème de sac-à-dos borné suivant :

$$(KP_{L,\overline{w}_j}) = \begin{cases} f_{\overline{w}_j}(L) = \max \sum_{i \in S_{\overline{w}_j}} c_i x_{ij} \\ \text{s.c.} \sum_{i \in S_{\overline{w}_j}} l_i x_{ij} \leq L \\ x_{ij} \leq b_i, x_{ij} \in N, i \in S_{\overline{w}_j}. \end{cases}$$

où $S_{\overline{w}_j} = S_{\overline{w}_j^u}$, x_{ij} est le nombre d'apparitions de la pièce du type i dans la bande $(L, \overline{w}_j)$, et $f_{\overline{W}_j}(L)$ le profit de la bande. $S_{\overline{w}_j^u} = i \in I, w_i = \overline{w}_j$, cela signifie que $S_{\overline{w}_j^u}$ est l'ensemble de types de pièces dont la hauteur est égale à $\overline{w}_j$
La résolution du $(KP_{L,\overline{W}})$ en utilisant la programmation dynamique donne la solution optimale de chaque $(KP_{L,\overline{w}_j}), j \in J$.
Dans la deuxième étape, les algorithmes SGA, déterminent le nombre d'occurrences y_i de chaque bande uniforme afin de construire le meilleur plan de découpe sans violer la borne supérieure b_i de chaque type de pièce $i \in I$. De cette manière, un plan de découpe optimal est construit en fonction de la solution du problème de sac-à dos suivant :

$$(BKP_{L,W}) = \begin{cases} g_L(W) = \max \sum_{j \in J} f_{\overline{w}_j}(L) y_j \\ \text{s.c.} \sum_{j \in J} \overline{w}_j y_j \leq W \\ y_j \leq a_i, y_j \in N, j \in J. \end{cases}$$

Le nombre d'apparitions y_i de la bande $(L, \overline{w}_j), j \in J$ dans le rectangle (L, W) est borné par a_j exprimé par :

$$a_j = min\Big\{ \left\lfloor \frac{W}{\overline{w}_j} \right\rfloor, min_{i \in s_{\overline{w}_j}} \Big\{ \left\lfloor \frac{b_i}{x_{ij}} \right\rfloor, \text{avec } x_{ij} > 0 \Big\} \Big\}.$$

– **Génération de bandes générales :**
Pour générer des bandes générales, Hifi et M'hallah ([39]) ont utilisé le même problème de sac-à-dos précédent. Cependant, les pièces appartenant à une bande, peuvent avoir des hauteurs différentes.
Dans ce cas, $S_{\overline{w}_j} = S_{\overline{w}_j^g}$, avec, $S_{\overline{w}_j^g} = i \in I, w_i \leq \overline{w}_j$, cela signifie que $S_{\overline{w}_j^g}$ est l'ensemble pièces dont la hauteur est inférieure ou égale à $\overline{w}_j$
Dans le même article, les auteurs proposent le programme linéaire en nombres entiers suivant pour combiner N bandes générales et r bandes uniformes afin de construire un plan de découpe général.

$$(IP_{L,W}) = \begin{cases} h_L(W) = \max \sum_{j\in J} \sum_{m=0}^{N_{j+1}} f_{\overline{w}_j{}^m}(L) y_j^m \\ \text{s.c.} \sum_{j\in J} \sum_{m=0}^{N_{j+1}} \overline{w}_j y_j^m \leq W \\ \sum_{j\in J} \sum_{m=0}^{N_{j+1}} x_{ij}^m y_j^m \leq b_i, i \in I \\ y_j^m \in N, j \in J, m = 0, \ldots, N_j + 1. \end{cases}$$

où $x_{ij}^{N_j+1}, i \in I$ est la valeur de la solution optimale de $(BK_{L,\overline{w}_j}), j \in J$ et $f_{\overline{w}_j}^{N_j+1}(L)$ représente la valeur de la fonction objectif qui lui correspond.
y_j^m représente le nombre d'occurrences du m$^{\text{éme}}$ type de bandes $(L, \overline{w}_j)$ dans le rectangle (L, W) et x_{ij}^m représente le nombre d'occurrences de la $i^{\text{éme}}$ pièce dans cette bande.

Le problème $(IP_{L,W})$ étant NP-difficile, les auteurs utilisent un algorithme glouton pour trouver une approximation.

2.2.2 Un algorithme amélioré par génération de bandes

L'idée de base de l'algorithme amélioré de génération de bandes (ESGA) consiste à diviser le rectangle initial (L, W) en deux sous-rectangles (L, β) et $(L, W - \beta)$. Le premier sous-rectangle est résolu en utilisant l'algorithme SGA général et le deuxième sous-rectangle est résolu en utilisant une procédure alternative.
Par ailleurs, ESGA détermine le point de division β en utilisant une procédure de discrétisation sur les hauteurs des pièces et un calcul d'une borne supérieure des sous-rectangles et en comparant cette dernière à une borne inférieure de départ .

- **La procédure de discrétisation**
  La procédure de discrétisation limite la division sur un ensemble fini de sous-rectangles qui participent potentiellement à la meilleure solution.
  Cette technique ; utilisée aussi par Christofides et Whitlock [57] pour le problème non contraint, est basée sur la construction de l'ensemble des combinaisons linéaires sur les hauteurs des pièces.
  Pour un sous-rectangle (L, β), l'ensemble des combinaisons linéaires sur les hauteurs est :

$$Q(L,\beta) = \Big\{ y \Big| y = \sum_{i\in S_\beta} w_i z_i \leq \beta, z_i \leq min\big\{ b_i, \left\lfloor \frac{L}{l_i} \right\rfloor \left\lfloor \frac{\beta}{w_i} \right\rfloor, z_i \in N \big\} \Big\}.$$

  où $S_\beta = \Big\{ i \in I : \Big| w_i \leq \beta \Big\}$

- **La borne supérieure**
  Dans l'algorithme ESGA, la borne supérieure permet de faire une estimation du profit généré par la découpe d'un sous-rectangle. Soit $\beta \in Q_{(L,W)}$ une découpe valide et (L, β) le sous-rectangle courant généré par cette découpe avec les points (L_0, β_0) définis par :

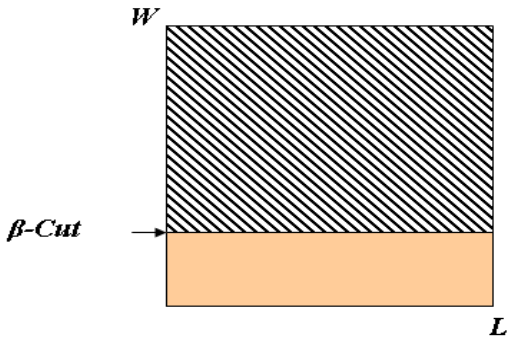


FIGURE 2.1 – Division en sous-rectangles ($\beta - cut$).

$L_0 = max_{x \in P_{(L,\beta)}}\Big\{x\Big\}$ et $\beta_0 = max_{y \in P_{(L,\beta)}}\Big\{y\Big\}$
où $P(L,\beta) = \Big\{\ x \Big| x = \sum_{i \in I} l_i z_i \leq L, w_i \leq \beta, z_i \leq min\Big\{b_i, \Big\lfloor \frac{L}{l_i} \Big\rfloor \Big\lfloor \frac{\beta}{w_i} \Big\rfloor \Big\}, z_i \in N\ \Big\}$.
La borne supérieure $U_{(L,\beta)}$ du sous-rectangle (L,β) est obtenue suivant la solution optimale du problème de sac-à-dos KP_U, cela signifie que

$$U_{(L,\beta)} = max\Big\{\ \sum_{j \in J} f_{\overline{w}_j}(L) z_j \Big| \sum_{j \in J} \overline{w}_j z_j \leq \beta, z_j \in N, j = 1 \in J\ \Big\},$$

où $f_{\overline{w}_j}(L)$ est le profit généré par la bande générale $(L, \overline{w}_j)$ et z_j le nombre d'occurrences de cette bande dans le sous-rectangle (L,β).

– **Le choix du point de division**

Dans l'algorithme ESGA, les auteurs utilisent la programmation dynamique pour résoudre KP_U ; ils calculent toutes les bornes supérieures des sous-rectangles possibles (L,y) avec $0 \leq y \leq W$.
Par ailleurs, pour chaque $\beta - cut$ qui génére le sous-rectangle courant (L,β) et le sous-rectangle restant $(L, W-\beta)$, le processus d'évaluation suivant est lancé :

$$Lower_{(L,\beta)} + U_{(L,W-\beta)} \leq Best_{(L,W)}$$

où $Best_{(L,W)}$ est la meilleure solution courante de (L,W) et $Lower_{(L,\beta)}$ la meilleure solution réalisable courante obtenue sur le sous-rectangle (L,β) (voir figure 2.1).
Ce processus permet de décider, si cette découpe est susceptible d'améliorer la meilleure solution courante de (L,W). Cette découpe est négligée dans le cas contraire.

2.2.3 Un algorithme hybride de génération de bandes

L'algorithme HESGA peut être considéré comme une amélioration de l'algorithme ESGA ; les auteurs introduisent dans cette approche basée sur une recherche locale. Le rectangle initial (L,W) est considéré comme la racine et les pièces représentent les feuilles. A chaque étape du développement de l'arbre de recherche ; l'algorithme exécute la procédure ESGA sur les deux sous-rectangles (L,β) et $(L, W-\beta)$.

Afin de réduire le temps de résolution, les auteurs introduisent deux stratégies (HC1 et HC2) : Supposons que le rectangle (L,W) peut être divisé en deux sous-rectangles (L, β) et $(L, W - \beta)$

- **HC1 :** La première stratégie réduit l'ensemble $Q(L, W)$ en utilisant une approche qui exploite la contrainte de la borne supérieure (b_i) de chaque pièce (voir Hifi [36], Christofides et Whitlock[57]).
- **HC2 :** La deuxième stratégie concerne la borne supérieure $U_{(L,W-\beta)}$ de chaque sous-rectangle ; cette stratégie introduit un paramètre ρ dans le processus d'évaluation du profit généré par un sous-rectangle comme suit :

$$Lower_{(L,\beta)} + \rho \left[U_{(L,W-\beta)}\right] \leq Best_{(L,W)}$$

où $Best_{(L,W)}$ est la meilleure solution courante de (L, W) , $Lower_{(L,\beta)}$ la meilleure solution réalisable courante obtenue sur le sous-rectangle (L, β) et ρ, une constante.

2.2.4 Un algorithme de recherche par trajectoires

Alvarez-Valdes et al ont proposé dans [67] une approche qui intègre une diversification et intensification. Cette approche génère plusieurs solutions en explorant des trajectoires donnant des solutions de bonne qualité, puis démarre de l'une de ces solutions (appelée solution initiale) afin de trouver de meilleures solutions dans le voisinage des chemins empruntés ; ces solutions sont appelées solutions guidées.

Pour plus d'explication, on se propose dans la suite de cette section, de détailler quelques méthodes utilisées par les auteurs pour générer une solution initiale et guider la recherche afin d'améliorer la qualité de celle-ci.

– **La méthode BLP :**

BLP [1] est l'une des méthodes les plus simples pour trouver un plan de découpe réalisable d'un rectangle initiale $R = (L, W)$ en utilisant un ensemble de pièces S.

Dans un algorithme BLP classique, les pièces sont ordonnées initialement selon un critère de sélection qui peut être différent d'une version à une autre. La première pièce est positionnée dans le coin inférieur gauche du rectangle R (première bande) ; à chaque itération une pièce est sélectionnée de l'ensemble des pièces restantes, et placée à droite de la pièce précédente si l'espace est suffisant dans la bande courante, sinon cette dernière est positionnée sur la première pièce de la bande précédente (nouvelle bande).

L'algorithme trouve une solution réalisable, s'il ne trouve pas d'espace pour positionner une pièce (voir figure 2.2).

Naturellement, pour chaque critère de sélection, l'algorithme produit une solution différente, dans ce sens Alvarez-Valdes et al ont proposé une implémentation (appellée GBLP) qui utilise un ordre décroissant des w_i à chaque construction d'une nouvelle bande, cependant, GBLP utilise une sélection par ordre décroissant des l_i pour développer une bande.

1. *Bottom-Left Procedure*

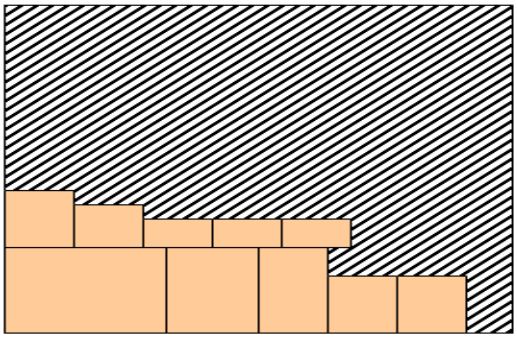


FIGURE 2.2 – Plan de découpe réalisable par BLP.

– **La méthode GRASP :**
GRASP [2] est une méthode itérative qui procède en deux phases : construction et recherche locale. La phase de construction génère une solution réalisable et la phase de recherche locale explore le voisinage de cette solution afin de trouver un optimum local. Alvarez-Valdes et al ont proposé une adaptation de la méthode GRASP pour le problème de découpe contraint à deux dimensions. La phase de construction peut être décrite comme suit :

- A chaque itération de la procédure de construction ; l'algorithme maintient une liste d'éléments candidats et procède à une évaluation des profits générés par ces éléments en utilisant une fonction gloutonne ;
- La procédure d'évaluation est utilisée pour restreindre la liste des éléments candidats à la construction (cette liste est appelée RCL) ;
- Le choix de l'élément à injecter dans la construction d'une solution partielle courante, se fait aléatoirement parmi les éléments de RCL ;
- A chaque fois qu'un élément est injecté dans la construction d'une solution partielle, la liste RCL est réévaluée ;

Les auteurs proposent deux versions de cette procédure, selon les éléments considérés dans la phase de construction : *Grasp based on pieces (pour une construction par pièces) et* Grasp based on strips pour une construction par bandes.
De la même manière, les auteurs proposent deux méthodes de recherche locale, selon le type de construction (par pièce ou par bande).

– **L'algorithme PR :**
L'algorithme PR [3] combine des solutions obtenues par "'GRASP piece"' et "'Grasp strip"' ; en effet les deux procédures produisent des solutions de structures différentes dans le sens où la méthode GRASP Strip produit des solutions de bonne qualité alors que les solutions produites par Grasp piece créent une diversité. La combinaison des deux caractéristiques (qualité et diversité) produisent généralement de meilleures solutions.
L'algorithme construit les m meilleures solutions produites par les deux méthodes. L'ensemble des $2m$ solutions est ordonné par ordre décroissant des profits et chaque solution

2. *Greedy Randomized Adaptive Search Procedure*
3. *Path Relinking*

de cet ensemble est utilisée comme solution de guidage pour toutes les autres solutions de l'ensemble (une de ces solutions est appelée solution initiale).
Pour chaque bande de la solution de guidage, l'algorithme tente de l'ajouter au début de la solution initiale, pour rendre la structure de la solution générée compatible avec les dimensions du rectangle initial, là où les dernières bandes peuvent être retirées.
Dans le cas où le remplacement produit une violation des demandes b_i sur certaines pièces, les bandes appartenant à la solution initiale contenant ces pièces sont retirées. L'espace libéré est fusionné à l'espace non découpé du rectangle initial et découpé en utilisant la procédure GBLP. A la fin de ce processus, la structure de la solution de guidage est imposée à la solution initiale, mais les solutions intermédiaires peuvent être meilleures que les deux solutions.

2.3 Méthodes exactes

2.3.1 Programmation linéaire en nombres entiers

Dans la résolution des problèmes d'optimisation, il est souvent important de pouvoir formuler le problème étudié sous forme d'un programme linéaire pour lequel il existe de nombreuses méthodes de résolution. Ces dernières permettent de cerner le problème en identifiant sa fonction objectif et ses contraintes.
Il existe de nombreux algorithmes permettant de résoudre les programmes linéaires dans le cas où les variables utilisées sont des variables réelles. Nous citons la méthode du simplexe (voir par exemple Dantzing [12] et Dantzing [13]).
A titre d'exemple, une première modélisation en programmation linéaire en nombres entiers du problème de découpe unidimensionnel a été donnée par Gilmore et Gomory [61].

Par ailleurs, Lodi et Monaci [3] ont présenté un modèle en programmation linéaire traitant un type de configuration par bandes. La principale particularité de ce modèle est la vérification explicite de la contrainte guillotine en distinguant les pièces qui initialisent une bande (première pièce d'une bande) et celles qui sont découpées d'une bande. Il est donc intéressant de revoir en détail ce modèle.

2.3.1.1 Le Modèle M1

On considère m types de pièces ; on rappelle que chaque type de pièce i avec $(i = 1, ..., m)$ est caractérisé par sa dimension (l_i, w_i), son profit c_i et sa demande a_i. Le nombre total de pièces dans le problème est $n = \sum_{i=1}^{m} a_i$.
On considère que les pièces sont ordonnées par ordre décroissant des hauteurs $(w_1 \leq w_2 \leq ... \leq w_n)$.

Le modèle démontre que n bandes potentielles peuvent être initialisées en utilisant les n pièces.

Soit une bande k, si cette bande est utilisée, elle est initialisée par la pièce k $(k = 1, ..., n)$, donc la possibilité de découper les n pièces de cette bande peut être exprimée par une variable binaire x_{jk} comme suit :

$$x_{jk} = \begin{cases} 1, \text{si la pièce j est découpée de la bande k}, (k = 1, ..., n; j = k, ..., n) \\ 0, \text{sinon} \end{cases}$$

Le modèle M1 est donc le suivant :

$$\begin{cases} \max \sum_{j=1}^{n} c_j \sum_{k=1}^{j} x_{jk} & & (1) \\ \text{s.c.} \sum_{k=1}^{j} x_{jk} \leq 1 & (j = 1, ..., n); & (2) \\ \sum_{j=k+1}^{n} l_i x_{jk} \leq (L - l_k) x_{kk} & (k = 1, ..., n-1); & (3) \\ \sum_{k=1}^{n} w_k x_{kk} \leq W & & (4) \\ x_{jk} \in \{0, 1\}, & (k = 1, ..., n); (j = k, ..., n); & (5) \end{cases}$$

La fonction objectif maximise la somme des profits alors que, l'équation (2) garantit que chaque pièce est découpée une et une seule fois de la bande dont la hauteur est supérieure ou égale à la hauteur de la pièce. L'équation (3) garantit que la contrainte sur la largeur des bandes est respectée. Cette équation garantit aussi, que si la pièce k n'est pas découpée de la bande, celle-ci reste vide. L'équation (4) impose la contrainte de la hauteur.

2.3.1.2 Le modèle M2

Le deuxième modèle présenté par Lodi et Monaci [3] est un modèle à variables entières. Soit $\alpha_0 = 0, \alpha_i = \alpha_{i-1} + a_i$ et $n = \alpha_m$ (a_i est la demande de la pièce de type i). Soit β_k =Min $\{i : \alpha_i \geq k\}$ la pièce d'initialisation de la bande k avec $k = 1, ..., n$. Soit q_k qui dénote l'initialisation des bandes ($q_k = 1$, si la bande k est utilisée et si au moins une pièce de l'ensemble β_k l'initialise). Soit $w_1 \geq w_2 \geq ... \geq w_m$ et x_{ik} le nombre de pièces de type i découpées de la bande k, $i > \beta_k$

Le modèle M2 est le suivant :

$$\begin{cases} \max \sum_{i=1}^{m} c_i \sum_{k=1}^{\alpha_i} x_{ik} + \sum_{k=1}^{n} c_{\beta_k} g_k & & (1) \\ \text{s.c.} \sum_{k=1}^{\alpha_i} x_{ik} + \sum_{k=1}^{\alpha_i} q_k \leq a_i & (i = 1, ..., m); & (2) \\ \sum_{i=\beta_k}^{m} l_i x_{ik} \leq (L - l_{\beta_k}) q_k & (k = 1, ..., n); & (3) \\ \sum_{k=1}^{n} w_{\beta_k} q_k \leq W & & (4) \\ x_{ik} \in Z_+ (\forall i, k = 1, ..., \alpha_i); & (5) & \\ q_k \in B; (k = 1, ..., n); & (6) & \end{cases}$$

Ce modèle est à $O(mn)$ variables et $O(n)$ contraintes. La fonction objectif maximise la somme des profits des bandes initialisées. L'équation (2) garantit que le nombre de pièces du type i utilisées dans les constructions de bandes est inférieure ou égale à la demande a_i, alors que l'équation (3) garantit que la contrainte sur la largeur des bandes est respectée. Finalement, l'équation (4) impose la contrainte de la hauteur W.

2.3.2 Programmation dynamique

L'idée principale de la programmation dynamique consiste à décomposer le problème initial en sous-problèmes de petites tailles. L'évaluation de ces derniers permet, par la suite, d'éliminer les moins intéressants de l'espace de recherche. Ainsi, les techniques de la programmation dynamique permettent de trouver une succession de coordonnées de décisions afin d'atteindre la solution optimale.

Gilmore et Gomory dans [62] ont proposé la première fonction récursive basée sur la programmation dynamique pour le problème de découpe à deux niveaux. Cette formulation a été améliorée par Beasley [25] pour répondre au problème de découpe guillotine non contraint.

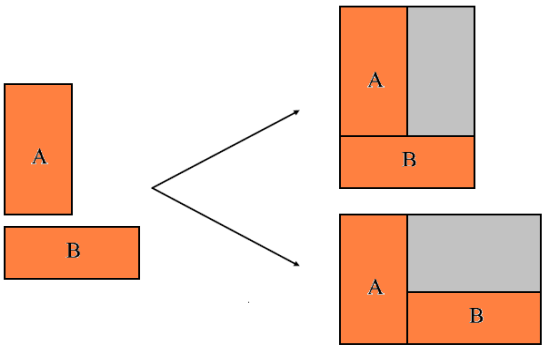


FIGURE 2.3 – Constructions horizontales et verticales.

Par la suite, Hifi a proposé [32] une amélioration de l'algorithme exact de Viswanathan et Bagchi's [26] pour le problème de découpe guillotine contraint à deux dimensions. Peu après Hifi et Roucairol [37] ont proposé un algorithme exact pour le même problème basé sur une stratégie de construction avec un retour arrière.
Dans la suite, nous allons décrire le principe de cet algorithme et la stratégie de construction qu'il implémente.

2.3.2.1 Stratégie de construction

- Un plan de découpe forme une construction horizontale si la combinaison des deux rectangles $(x1, y1)$ et $(x2, y2)$ engendre un rectangle de dimensions $(x1 + x2, maxy1, y2) \leq (L, W)$.
- Un plan de découpe forme une construction verticale si la combinaison des deux rectangles $(x1, y1)$ et $(x2, y2)$ engendre un rectangle de dimensions $(maxx1, x2, y1 + y2) \leq (L, W)$.

A chaque rectangle guillotine R, l'algorithme fait correspondre respectivement une fonction intermédiaire $g(R)$ et une fonction complémentaire $h(R)$. La fonction $g(R)$ représente la valeur de la somme des profits des pièces constituant R et $h(R)$ est l'estimation du reste de la surface du support, noté $P = \text{surface(P)/surface(R)}$. C'est aussi la valeur maximale de la région P composée par un sous-ensemble de pièces (en respectant les contraintes sur les pièces). Finalement, le but est de construire le rectangle guillotine de valeur maximale $f(.) = g(.) + h(.)$. Généralement, il n'est pas évident de calculer la valeur de $f(.)$ en un temps raisonnable. Pour trouver une estimation pour la fonction $h(R)$. L'algorithme utilise une borne supérieure pour le problème initial $f0(R)$.
Dans le cas d'un problème de maximisation si la valeur maximale sur $f0(.)$ est réalisée pour un certain R et que $h0(R) = 0$, alors R est une solution optimale.

2.3.2.2 Principe de l'algorithme

Le principe de l'approche s'appuie essentiellement sur l'utilisation de deux listes. La première liste, notée Open, comporte au départ une copie de chaque type de pièces. Elle représente la liste globale de l'algorithme. La deuxième liste, notée Clist, est initialement vide. Elle représente la liste intermédiaire qui sert à stocker toutes les constructions susceptibles d'améliorer la solution en cours. A chaque étape de l'algorithme un élément R, de valeur $f0$ maximum, est choisi dans

la liste Open. Cet élément est un rectangle guillotine qui est immédiatement transféré vers la liste Clist. Par la suite, tous les éléments de la liste Clist (y compris R) sont combinés avec le rectangle choisi R. Ces combinaisons sont réalisées en appliquant les constructions horizontales et verticales. Chaque rectangle guillotine $R0$, issu d'une des constructions considérées, est introduit dans la liste Open s'il vérifie les conditions suivantes :

1. les dimensions de $R0$ ne dépassent pas les dimensions du support rectangulaire.
2. le contenu de R ne viole pas les contraintes sur les demandes.

L'algorithme s'arrête soit lorsque le rectangle R, choisi dans la liste Open, réalise la valeur $h0(R) = 0$, soit lorsque, la borne supérieure $f0(R)$ est inférieure ou égale à la valeur de la meilleure solution en cours.

2.3.3 Méthodes par séparation et évaluation

La méthode par séparation et évaluation (Branch and Bound) est une méthode générale pour calculer la solution exacte d'un problème d'optimisation combinatoire. Elle est basée sur une recherche arborescente. L'idée principale est de réduire l'espace de recherche en utilisant des évaluations et des règles de dominance. Pour ce fait, on calcule des bornes qui, comparées à une solution déjà trouvée, permettent d'éliminer des branches de l'arborescence sans les parcourir. Les règles de dominance permettent d'identifier des sous-ensembles ne contenant pas de solutions optimales.
Ainsi, afin d'éviter le parcours de tous les nœuds, il est nécessaire d'effectuer une évaluation pour chacun des nœuds crées. Cette évaluation s'appuie sur le calcul d'une borne inférieure et d'une borne supérieure de la fonction objectif.

De nombreux algorithmes exactes ont fait l'objet de plusieurs publications dans la dernière décennie, citons Christofides et Hadjiconstantinou [58], Hifi et Zissimopoulos [52], Hifi [52], Cung et al [80].

2.4 Conclusion

A travers un survol des méthodes et approches existantes, on a montré le nombre important de travaux sur les problèmes de découpe depuis leur introduction par Kantorovich [27]. Cependant, vu le nombre de variantes du problème et des spécificités de chaque variante, l'efficacité des méthodes dépendent principalement de la variante traitée et surtout des contraintes considérées. Ainsi, une étude minutieuse du problème traité, des différentes contraintes et des méthodes de résolution des différentes variantes est une étape importante dans la conception de nouveaux algorithmes.

Nous nous sommes intéressés dans la deuxième partie de cette thèse à la résolution approchée, du problème de découpe contraint à deux dimensions à deux niveaux. Pour ce problème, on présente deux nouvelles méthodes séquentielles approchées .

Deuxième partie

RÉSOLUTION SÉQUENTIELLE APPROCHÉE

Chapitre 3

Recherche par faisceaux pour le problème de découpe contraint à deux dimensions et à deux niveaux [1]

Contenu

Dans ce chapitre, nous nous intéressons au problème de découpe-placement à deux dimensions et à deux niveaux. Ce problème correspond au cas où l'on veut optimiser l'utilisation d'une entité disponible en taille limitée en y plaçant des sous-entités prédéterminées. On considère que l'entité disponible est donnée sous forme rectangulaire de dimensions fixées (L, W), qu'on appelle rectangle initial. Les sous-entités ont aussi des formes rectangulaires qu'on appelle pièces. Chaque pièce i est caractérisée par ses dimensions (l_i, w_i), le profit (c_i) qu'elle génère dans son placement dans le rectangle initial *(cas du profit)* ou sa surface $(c_i = l_i * w_i)$ dans le *cas de la surface*. Chaque pièce i est également caractérisée par une borne supérieure appelée aussi *demande* (b_i) qui limite le nombre d'apparitions de cette pièce dans le plan de découpe. On se propose de découper le rectangle initial en pièces appartenant à l'ensemble des pièces disponibles S tout en *(a)* minimisant la surface des pièces produites n'appartenant pas à cet ensemble, appelée chute ou *(b)* maximisant le profit des pièces produites appartenant à l'ensemble. Dans notre étude, nous supposons qu'on utilise des *découpes du type guillotine* qui consiste à prendre la découpe d'un côté à son opposé parallèlement aux deux autres. De plus, on considère que toutes les pièces de l'ensemble S sont fixées (i.e. la pièce de largeur l_i et de hauteur w_i n'est pas la même que celle qui possède la largeur w_i et la hauteur l_i).

1. *Le contenu de ce chapitre a fait l'objet d'une publication dans "Informs Journal On Computing [42]" et d'une conférence internationale "The Fifth ALIO/EURO conference on combinatorial optimization [43]"*

Ce problème est désigné dans la littérature par le problème de découpe contraint à deux dimensions et à deux niveaux noté FC2TDC [1]. Il est NP-difficile et a de nombreuses applications en industrie, notamment en placement de circuits intégrés, l'industrie de production de timbres, de bois et tout autre environnement qui impose une découpe guillotine à deux niveaux (voir Morabito et Garcia [72].

Nous proposons dans ce chapitre des heuristiques qui combinent une procédure de construction de bandes et une recherche par faisceaux. La construction par bandes génère un ensemble de bandes optimales ; alors que la recherche par faisceaux filtre les meilleures bandes pendant la construction de la solution en utilisant une fonction d'évaluation. L'influence de cette fonction sur la solution finale est évaluée en comparant une recherche par faisceaux locale (LBS) et une recherche par faisceaux globale (GBS). LBS implémente une politique d'évaluation locale de la solution obtenue pendant la construction alors que GBS fait une évaluation qui inclue la solution courante et une estimation de la solution (des solutions) générée(s) par les futures constructions. Les résultats expérimentaux vont dans le sens d'une efficacité particulière des heuristiques sur un ensemble d'instances, classées, moyennes et larges.

1. *Constrained Two-Staged Two-Dimensional Cutting Problem*

3.1 Construction de bandes générales

Dans cette section, on rappelle le principe de génération de bandes générales. On considère une instance FC2TDC, une bande $(L, \overline{w}_j)$ avec $0 < \overline{w}_j \leq W$ est obtenue en appliquant une découpe guillotine sur le rectangle initial (L, W). La première découpe peut être horizontale ou verticale. Si la première découpe est horizontale alors la bande générée est une bande horizontale $(L, \overline{w}_j)$. de la même manière, si la première bande est verticale alors la bande générée est verticale $(\overline{w}_j, W)$. Dans ce qui suit et sans perte de généralité, on considère que la première découpe est horizontale.

La bande $(\overline{w}_j, L)$ est construite en fonction de la solution optimale du problème de sac-à-dos borné suivant :

$$KP^g_{(L,\overline{w}_j)} = \begin{cases} f_{\overline{w}_j}(L) = \max \sum_{i \in S_{\overline{w}_j}} c_i x_{ij} \\ \text{s.c.} \sum_{i \in S_{\overline{w}_j}} l_i x_{ij} \leq L \\ x_{ij} \leq b_i, x_{ij} \in N, i \in S_{\overline{w}_j}. \end{cases}$$

où x_{ij} est le nombre d'apparitions de la pièce du type i dans la bande $(L, \overline{w}_j)$, et $f_{\overline{W}_j}(L)$ le profit de la bande. Avec $S_{\overline{w}_j} = \{i \in I, w_i \leq \overline{w}_j\}$, cela signifie que $S_{\overline{w}_j}$ est l'ensemble de type de pièces dont la hauteur est inférieure ou égale à $\overline{w}_j$.

Soit r, le nombre de hauteurs distincts des pièces et p le plus grand index de la pièce de hauteur w_p, tel que $w_1 < w_2 < ... < w_p$; la résolution du $(KP^g_{L,\overline{w}_p})$ en utilisant la programmation dynamique donne la solution optimale de chaque $KP^g_{(L,\overline{w}_j)}, j \leq p$; elle permet ainsi, la construction de p bandes générales.

Cette idée n'est pas nouvelle, elle a été utilisée par plusieurs auteurs, notamment Hifi et Roucairol [37], Hifi et M'Hallah [39] et Alvarez-Valdes et al [67] pour le même problème (voir chapitre 2).

3.2 Recherche par faisceaux

Dans cette section, on décrit la recherche par faisceaux [2] et l'importance de la fonction d'évaluation dans ce type de recherche. On présente ensuite l'adaptation de cette recherche à notre problème et comment nous utilisons les deux fonctions d'évaluation pour résoudre FC2TDC.

3.2.1 Description

La recherche par faisceaux est une méthode de recherche arborescente classique ; elle a été introduite pour les problèmes d'ordonnancement par Ow et Morton [64] et elle a eu beaucoup de succès dans plusieurs autres problèmes d'optimisation. Son objectif est d'éviter le parcours exhaustif de l'espace de recherche en développant un espace de recherche partiel. Cette méthode appelée aussi *recherche par programmation dynamique avec filtrage* est une méthode de séparation

2. *Beam Search en anglais*

et évaluation sur un espace de recherche réduit.
A chaque niveau de l'arbre de recherche, un sous-ensemble de nœuds (appelé *nœuds élites*) est considéré pour le branchement, les autres nœuds sont écarté de la recherche. Le nombre β de nœuds considérés à chaque niveau est appelé *largeur du faisceau* . L'efficacité de cette méthode est considérablement influençable par la politique de filtrage à chaque niveau et donc par la fonction d'évaluation.

La figure 3.1 montre un pseudo code d'une recherche par faisceaux à partir d'un nœud racine N_0.

Initialisation

$N = \{N_0\}$, et $M = \emptyset$;

Phase Itérative

1. Pour chaque nœud de N.
   (a) Brancher sur le nœud et générer les nœuds fils ;
   (b) Évaluer chaque nœud et l'insérer dans M;
2. Insérer dans N les $min\{\beta, |M|\}$ meilleurs nœuds de M;
3. $M = \emptyset$;

Condition d'arrêt

Si $N = \emptyset$ Alors sélectionner le nœud qui génère le meilleur profit et arrêter ;
Sinon, répéter la *Phase Itérative* ;

FIGURE 3.1 – Algorithme de recherche par faisceaux.

3.2.2 Recherche par faisceaux appliquée au problème de découpe contraint à deux dimensions

L'adaptation de la recherche par faisceaux pour le FC2TDC nécessite la définition des nœuds de l'arborescence et du mécanisme de branchement ; dans ce contexte, un nœud est représenté par une paire de sous-rectangles $(L, W-y)$ et (L, y), avec $y \leq W$ et $(L, W-y)$ la solution réalisable courante. $(L, W-y)$ est la solution construite par combinaison des bandes et (L, y) le reste de la surface du rectangle initial. Par conséquent, la racine de l'arborescence est représentée par les deux sous-rectangles $(L, 0)$ et (L, W). Dans cette configuration aucune bande n'est encore placée dans le rectangle initial.

Soit r, le nombre de hauteurs distincts des pièces et p le plus grand index de la pièce de hauteur w_p, tel que $w_1 < w_2 < ... < w_p$; le branchement sur le nœud $u = ((L, W-y), (L, y)), y \leq W$ est équivalent à la découpe de la bande (L, β), $\beta \leq y$, $\beta \in \{\overline{w_1}, \overline{w_2}, ..., \overline{w_r}\}$ du sous-rectangle (L, y). On fait remarquer qu'il y aura au maximum r branchement sur le nœud u ; chaque branchement correspond à la découpe d'une bande $(L, \overline{w_j})$, $j \in J$. Chacun de ces nœuds correspond à deux sous-rectangles $(L, W-y+\overline{w_j}), j \in J$.

Le processus de recherche peut être décrit comme suit :

- Sur le nœud racine, aucune bande n'est découpée ; On pose $b_i^{'} = b_i$, $i \in I$, l'algorithme crée r bandes générales différentes $(L, \overline{w_j}), j \in J$ par la résolution de $KP^g_{(L,\overline{w}_j)}$. Chaque bande $j \in J$ correspond à un nœud fils, à développer.
- Pour effectuer un branchement sur le nœud $((L, W-y),(L,y))$ pendant le parcours, l'algorithme effectue les opérations suivantes :

  1. calculer les nouvelles demandes $b_i^{'} = b_i - v_i$, v_i est le nombre d'apparitions de la pièce du type i dans la bande.
  2. pour $j = r, \ldots 1$, résoudre $KP^g_{(L,\overline{w}_j)}$.
  3. pour chaque bande $(L, \overline{w}_j), j = 1, ..., r$ et $\overline{w}_j \leq y$,
     - (a) découper la bande du sous-rectangle (L, y)
     - (b) obtenir les nouveaux nœuds $((L, W - y + \overline{w}_j), (L, Y - \overline{w}_j))$
     - (c) calculer une évaluation $\hat{Z}$ du nouveau nœud

Dans l'étape 3c, l'algorithme fait une évaluation des nœuds en utilisant la fonction d'évaluation. On considère deux types de fonctions, locale *(LBS)* et globale *(GBS)*.

3.2.3 Recherche par faisceaux − utilisation de la stratégie LBS

3.2.3.1 Principe

LBS [3] [4] choisit les $min\Big\{\beta, |M|\Big\}$ nœuds ayant la meilleure solution réalisable parmi les nœuds de M. Pour évaluer le nœud $u = ((L, W-y),(L,y)), y \leq W$ de M, LBS utilise l'opérateur local qui calcule $\hat{Z}_u^{Local}$, $\hat{Z}_u^{Local}$ est la somme des profits des pièces qui constituent le sous-rectangle $(L, W-y)$.
$\hat{Z}_u^{Local} = \sum_{i \in I} c_i v_i$, avec v_i est le nombre d'apparitions de la pièce du type i dans $(L, W-y)$. La figure 3.2 montre un pseudo code de l'algorithme LBS.

3.2.4 Recherche par faisceaux − utilisation de la stratégie GBS

GBS [5] [6] utilise une fonction d'évaluation globale ; cette fonction permet de sectionner le nœud le plus potentiellement bénéfique pour la solution réalisable finale. GBS calcule une évaluation $\hat{Z}_u^{Global}$ du nœud $u = ((L, W-y),(L,y)), y \leq W$ de M en combinant :

(i) $\hat{Z}_u^{Local}$, la valeur de la solution réalisable locale du sous-rectangle $(L, W-y)$, et

(ii) $U_{(L,y)}$, une borne supérieure du sous-rectangle complémentaire (L, y).

La figure 3.2 montre un pseudo code le l'algorithme GBS.

---

3. *Local Beam Search*

4. *Cette méthode a été présentée lors du 7ème congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision −ROADEF− 2006 [40]*

5. *Global Beam Search*

6. *Cette méthode a été présentée lors de la conférence, International Conference on Metaheuristics and Nature Inspired computing, META, 2006 [46]*

`Entrée` : Une instance FC2TDC ;

`Sortie` : Une solution approximative de valeur $Best_{(L,W)}$;

Initialisation

$b_i^{'} = b_i,\ i \in I$;
$M = \emptyset$;
$N = \{u = ((L,0),(L,W))\}$;

Phase Itérative

1. Pour chaque nœud u de N;
   (a) Brancher sur le nœud u et générer les nœuds fils de u en créant r bandes générales différentes $(L,\overline{w_j}), j \in J$ par la résolution de $KP^g_{(L,\overline{w}_j)}$;
   (b) Pour chaque bande $(L,\overline{w_j}), j \in J$, générer le nœud $u_j = ((L, W-\overline{w_j}),(L,\overline{w_j}))$, calculer $\hat{Z}^{Local}_{u_j}$ et insérer u_j dans M ;
2. Insérer dans N les $min\{\beta, |M|\}$ nœuds de M ayant le plus grand $\hat{Z}^{Local}_u$;
3. $M = \emptyset$;

Condition d'arrêt

Si $N = \emptyset$ Alors sélectionner le nœud qui génère le meilleur profit $\hat{Z}$ et arrêter ;
Sinon, répéter la *Phase Itérative* ;

FIGURE 3.2 – L'algorithme LBS.

`Entrée` : Une instance FC2TDC ;

`Sortie` : Une solution approximative de valeur $Best_{(L,W)}$;

Initialisation

$b_i^{'} = b_i,\ i \in I$;
Résoudre $U_{(L,W)}$;
$M = \emptyset$;
$N = \{u = ((L,0),(L,W))\}$;

Phase Itérative

1. Pour chaque nœud u de N;
   (a) Brancher sur le nœud u et générer les nœuds fils de u en créant r bandes générales différentes $(L,\overline{w_j}), j \in J$ par la résolution de $KP^g_{(L,\overline{w}_j)}$;
   (b) Pour chaque bande $(L,\overline{w_j}), j \in J$, générer le nœud $u_j = ((L, W-\overline{w_j}),(L,\overline{w_j}))$, calculer $\hat{Z}^{Global}_{u_j} = \hat{Z}^{Local}_{u_j} + U_{u_j}$ et insérer u_j dans M ;
2. Calculer $\eta = max_{u \in M}\left\{\hat{Z}^{Global}_u\right\}$;
3. Pour chaque nœud $u \in M$, calculer l'écart entre $\hat{Z}^{Local}_u$ et η ;
4. Insérer dans N les $min\{\beta, |M|\}$ de M ayant le plus grand écart ;
5. $M = \emptyset$;

Condition d'arrêt

Si $N = \emptyset$ Alors sélectionner le nœud qui génère le meilleur profit $\hat{Z}$ et arrêter ;
Sinon, répéter la *Phase Itérative* ;

FIGURE 3.3 – L'algorithme GBS.

3.2.4.1 La borne supérieure

Une borne supérieure du profit génère par la découpe du sous-rectangle (L, y) est la solution optimale du problème de sac-à-dos suivant :

$$U_{(L,y)} = max\Big\{ \sum_{j \in J} f^g_{\overline{w}_j}(L) t_j \;\Big|\; \sum_{j \in J} \overline{w}_j t_j \leq y, t_j \in N, j \in J \Big\}$$

où $f^g_{\overline{w}_j}$ est le profit de la bande générale optimale $(L, \overline{w}_j)$ obtenue par la résolution du problème de sac-à-dos $KP^g_{(L,\overline{w}_j)}$, avec $b_i^{'} = b_i - v_i$, v_i est le nombre d'apparitions de la pièce du type i dans la solution réalisable correspondant à la bande $(L, W - y)$. L'entier t_j est le nombre d'apparitions de la bande générale $(L, \overline{w}_j)$ dans (L, W).

La résolution du précédent problème de sac-à-dos avec $y = W$ par la programmation dynamique permet de calculer les bornes supérieures de tous les sous-rectangles possibles (L, y), avec $y = 1, ..., W$; par conséquence, le calcul de $U_{(L,y)}$ se fait une seule fois sur le nœud racine avec $b_i^{'} = b_i$ et sa valeur sera utilisée pendant le développement de l'arborescence. Ce principe permet de calculer à l'avance, toutes les bornes supérieures des sous-rectangles.

3.2.4.2 Réduction de l'espace de recherche

Afin de réduire l'espace de recherche, On néglige chaque nouveau nœud $u = ((L, W - y), (L, y))$ de M avec :

$$\hat{Z}_u^{Global} = \hat{Z}_u^{Local} + U_{(L,y)} \leq Z^*$$

avec Z^* la meilleure solution réalisable courante.

3.2.4.3 Filtrage des nœuds

Afin de sélectionner les nœuds élites, on définit pour l'ensemble M, le sous ensemble η avec

$$\eta = max_{u \in M} \left\{ \hat{Z}_u^{Global} \right\}$$

Pour chaque nœud $u \in M$, on calcule l'écart entre $\hat{Z}_u^{Local}$ et η, on sélectionne, ensuite $min\{\beta, |M|\}$ ayant le plus grand écart. Dans le cas où plusieurs nœuds de M gérèrent les mêmes écarts, on sélectionne le nœud ayant le plus grand $\hat{Z}_u^{Local}$.
L'écart entre $\hat{Z}_u^{Local}$ et η est calculé par l'équation suivante : $Dev = 100 \frac{\hat{Z}_u^{Local} - \eta}{\hat{Z}_u^{Local}}$

3.2.4.4 Solution initiale

GBS démarre généralement sans solution initiale, cependant, afin d'améliorer la séparation et l'évaluation, tout spécialement dans le cas de stratégies en meilleur d'abord, GBS démarre avec la solution obtenue par LBS avec $\beta = 1$.

3.2.4.5 Exemple

La figure 3.4 illustre un exemple de construction d'une solution réalisable avec GBS. La figure 3.4.(a) montre l'ensemble des pièces à placer dans le rectangle initial présenté dans la figure 3.4.(b). Dans le cas où la largeur du faisceau $\beta = 1$, le noeud (1) (voir la figure 3.4.(c)) sera sélectionné pour

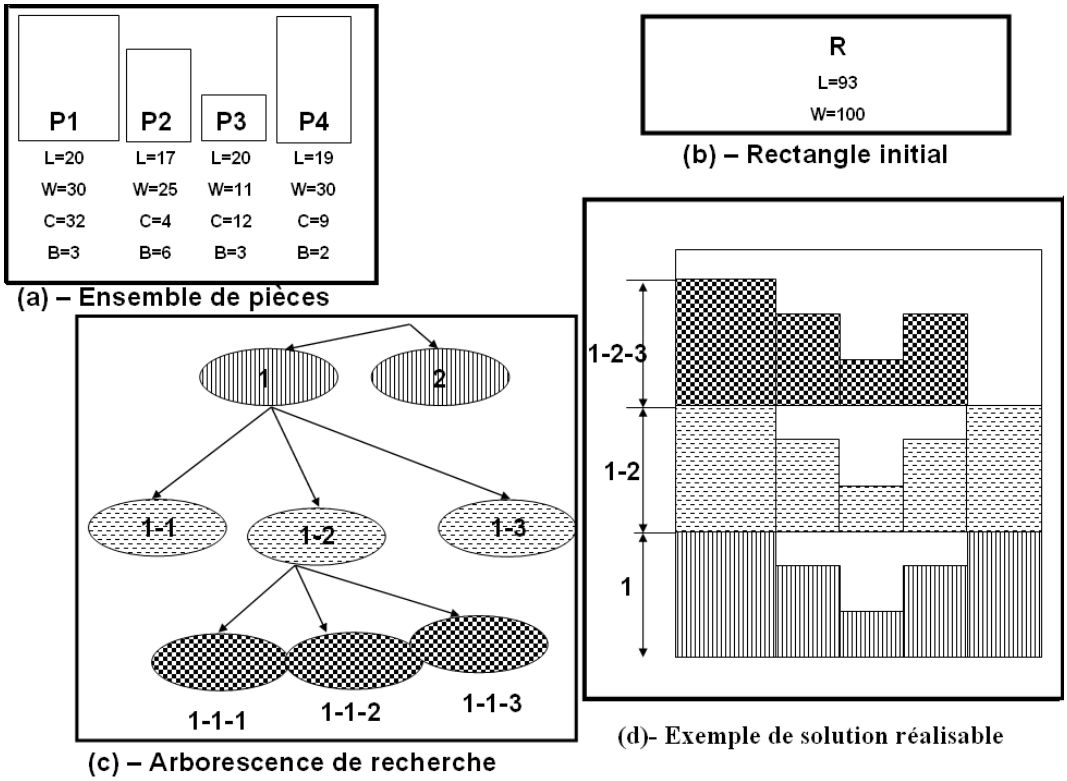


FIGURE 3.4 – Exemple de résolution GBS.

le développement, ce noeud correspond à la bande (1) présentée dans la figure 3.4.(d) de hauteur $w_1 = 30$ est d'une largeur $l_1 = 93$. Comme mentionné précédemment, cette bande est obtenue par la résolution d'un problème de sac à dos d'une capacité égale à la largeur du rectangle initial. La solution réalisable courante, dans ce stade du développement est égale à la somme des profits générés par les pièces placées dans la bande $(32 + 4 + 12 + 4 + 9)$.
Le chemin dans l'arbre de recherche représenté par les nœuds (1,1-2 et 1-2-2) correspond, ainsi à la solution réalisable présentée dans la figure 3.4.(d). Cette solution correspond à la superposition de 3 bandes optimales. La fonction d'évaluation est utilisée à chaque niveau pour sélectionner la bande (le nœud) élite qui va favoriser la qualité de la solution réalisable finale.

3.3 Résultats numériques

Dans cette section, on évalue la performance des algorithmes proposés sur 42 instances, classées dans la littérature comme moyennes et larges. Toutes les approches ont été codées en C++ et testées sur un pentium centrino 2.8 GHZ et 512 Méga octets de mémoire centrale (Nous avons limité les temps d'exécutions à 600 secondes).

3.3.1 Les instances

Les instances traitées, consistent en (i) 20 instances non-pondérées ($\forall i \in I, c_i = l_i w_i$) et (ii) 22 autres instances pondérées ($\forall i \in I, c_i \neq l_i w_i$). Chaque instance est traitée en horizontale (la première découpe est horizontale) et en verticale (la première découpe est verticale).
Les 20 instances de taille moyenne sont extraites de Alvarez-Valdes et al [67], les solutions optimales de ces instances sont connues et sont rapportées dans le même article. Dix de ces instances sont non-pondérées (APT30,...,APT39) et les dix autres sont pondérées (APT40,. . . ,APT49). La table 3.1, présente la dimension du rectangle initial (L, W), le nombre de type de pièces n et l'optimum de chaque instance pour une première découpe horizontale ($OptH$) et une première découpe verticale ($OptV$). Les instances pondérées (respectivement non-pondérées) sont signalées par P (respectivement S)
Nous avons également considéré 22 instances larges. Les dix premières instances sont pondérées (Nice25,. . . , Nice500, Path25,. . . , Path500) et ont été extraites de (Alvarez-Valdes et al [67]), dix autres instances (Nice25P,. . . , Nice500P, Path25P,. . . , Path500P) ont été obtenues par une transformation des 10 premières en instances non-pondérées ; enfin, les deux dernières (Nice1000P, Path1000P) sont de très grandes instances contenant 1000 pièces. On note que l'optimum de la pluspart de ces instances n'est pas connu. La table 3.2, présente la dimension du rectangle initial L, W, le nombre de type de pièces de chaque instance n et l'optimum de l'instance s'il existe (le signe * signifie que l'optimum de l'instance n'est pas connu).

Inst	Type	L	W	n	$OptH$	$OptV$
ATP30	P	927	152	38	140168	140197
ATP31	P	856	964	51	820260	821073
ATP32	P	307	124	56	37880	37973
ATP33	P	241	983	44	235580	234670
ATP34	P	795	456	27	356159	357741
ATP35	P	960	649	29	614429	614336
ATP36	P	537	244	28	129262	128814
ATP37	P	440	881	43	384478	385811
ATP38	P	731	358	40	259070	259137
ATP39	P	538	501	33	266135	266378
ATP40	S	683	138	46	63945	65584
ATP41	S	837	367	36	202305	196559
ATP42	S	167	291	59	32589	33012
ATP43	S	362	917	49	208998	212062
ATP44	S	223	496	39	70940	69784
ATP45	S	188	578	33	74205	69988
ATP46	S	416	514	42	146402	147021
ATP47	S	393	554	43	144317	142935
ATP48	S	931	254	34	165428	162458
ATP49	S	759	449	25	206965	211784

TABLE 3.1 – Instances moyennes

3.3.2 Objectifs

Nous nous somme fixés plusieurs objectifs à travers ces simulations. Le premier objectif est d'identifier le niveau β qui fait un bon compromis entre la qualité des solutions et le temps d'exécution. Évidemment, augmenter la largeur du faisceau β améliore toujours la valeur de la fonction objectif, mais fait augmenter le temps de résolution. Le deuxième objectif est d'analyser l'effet de

Inst	Type	L	W	n	$OptH$	$OptV$
nice25	S	1000	1000	25	860829	834157
nice50	S	1000	1000	50	*	*
nice100	S	1000	1000	100	*	*
nice200	S	1000	1000	200	*	*
nice500	S	1000	1000	500	*	*
path25	S	1000	1000	25	892765	699707
path50	S	1000	1000	50	750215	924908
path100	S	1000	1000	100	*	754753
path200	S	1000	1000	200	*	*
path500	S	1000	1000	500	*	*
nice25P	P	1000	1000	25	441993	428662
nice50P	P	1000	1000	50	*	*
nice100P	P	1000	1000	100	*	*
nice200P	P	1000	1000	200	*	*
nice500P	P	1000	1000	500	*	*
nice1000P	P	1000	1000	1000	*	*
path25P	P	1000	1000	25	441993	356088
path50P	P	1000	1000	50	*	487447
path100P	P	1000	1000	100	*	386033
path200P	P	1000	1000	200	*	*
path500P	P	1000	1000	500	*	*
path1000P	P	1000	1000	1000	*	*

TABLE 3.2 – Instances larges

la stratégie de parcours (par meilleur d'abord ou par profondeur d'abord) sur les performances des algorithmes. Notre troisième objectif est d'évaluer l'influence de la fonction d'évaluation sur la qualité des solutions obtenues et le temps de résolution en comparant les solutions obtenues par LBS et GBS, aux solutions optimales (si elles sont connues) et les résultats obtenus par des heuristiques de la littérature.

Pour analyser ces quatre aspects, nous avons considéré différents niveaux β. Nous avons traité chaque instance avec le niveau considéré en utilisant LBS^P_β (LBS avec un parcours en profondeur d'abord), LBS^M_β (LBS avec un parcours en meilleur d'abord), GBS^P_β (GBS avec un parcours en profondeur d'abord), GBS^M_β (GBS avec un parcours en meilleur d'abord).

3.3.3 Premier groupe d'instances

Notre étude a commencé par les instances de tailles moyennes. Les trois dernières colonnes de la table 3.3, présentent le nombre d'instances résolues à l'optimum ($NOpt$), la moyenne du pourcentage de déviation par rapport à l'optimum (Dev) et la moyenne des temps de résolution en seconde (AvT). Le pourcentage de déviation Dev d'une solution z^* par rapport à l'optimum Opt est calculé par l'équation $Dev = 100\frac{Opt-z^*}{Opt}$.

L'analyse de la table 3.3 indique que le pourcentage de déviation de GBS^M_2 et GBS^P_2 est tellement satisfaisant qu'il n'est pas nécessaire d'augmenter la largeur du faisceau β ; par ailleurs, l'augmentation de β permet d'améliorer la qualité des solutions de LBS.

Le temps de résolution de LBS et GBS avec une stratégie de recherche en meilleur d'abord est plus court que son correspondant avec une stratégie en profondeur d'abord. GBS^P_2 consomme en moyenne 0.2 secondes pour trouver l'optimum sur toutes les instances avec une première découpe horizontale ou verticale. GBS^P_2 trouve la solution optimale de 38 instances sur les 40 traitées.

	Recherche par faisceaux locale : LBS						Recherche par faisceaux globale : GBS			
	LBS_β^M			LBS_β^P			GBS_β^M		GBS_β^P	
	$\beta=1$	$\beta=2$	$\beta=3$	$\beta=1$	$\beta=2$	$\beta=3$	$\beta=1$	$\beta=2$	$\beta=1$	$\beta=2$
NOpt	2	2	5	2	4	8	7	37	7	40
Dev	7.40	5.29	4.97	7.40	5.03	4.00	1.99	0.01	1.99	0
Av.T	0.01	0.01	0.02	0.01	0.03	0.05	0.01	0.01	0.01	0.20

TABLE 3.3 – Résultats numériques de LBS et GBS avec $\delta = 1, 2$ et 3, en utilisant les stratégies de recherche en profondeur d'abord et meilleur d'abord sur les instances de taille moyenne.

GBS_2^P donne de meilleures solutions que GBS_2^M mais nécessite plus de temps d'exécution (0.20 versus 0.01 en moyenne). En général, la stratégie de recherche en profondeur d'abord est plus lente que la recherche par meilleur d'abord.

L'exécution de LBS avec $\beta \geq 4$ est visiblement non intéressante, car le temps de résolution devient supérieur au temps nécessaire à GBS_2 sans aucune amélioration de la qualité des solutions. Cette conclusion met en évidence l'importance de la fonction d'évaluation globale dans la recherche par faisceaux. Une bonne fonction d'évaluation, fait une évaluation non seulement de la solution locale mais aussi de l'influence de cette solution dans la solution finale. Cependant les résultats de LBS_2 peuvent être utilisés comme solutions de départ pour GBS. L'échec de LBS_2 peut être interprété par l'existence de maxima locaux dans l'espace de recherche.

Nous avons également comparé les performances des algorithmes proposés avec les performances de $HESGA_b$ et PR. La table 3.4 rapporte les solutions optimales des instances (Opt), ainsi que la déviation de l'optimum des solutions produites par $HESGA_b$, PR, LBS et GBS. Table 3.4 montre clairement que LBS est moins performant que $HESGA_b$ et PR mais GBS_2^P et GBS_2^M sont plus performants que $HESGA_b$ et PR en termes de qualité des solutions produites et du temps de résolution.
En conclusion, la stratégie de recherche en profondeur peut être considérée comme une stratégie de recherche efficace car elle produit de meilleures solutions en explorant un plus grand espace de recherche. Cependant cette stratégie est consommatrice en temps de calcul.

3.3.4 Deuxième groupe d'instances

Afin de tester la performance de nos algorithmes sur des grandes instances, nous avons comparé les résultats de LBS et GBS aux meilleures bornes inférieures qui existent dans la littérature pour ces instances, notamment, le solver Cplex V9, PR, $HESGA_b$.

Les solutions du solveur Cplex, ont été obtenues en utilisant le modèle de programmation linéaire M1 (Lodi et Monaci [3]) présenté dans le chapitre 2 et en limitant le temps d'exécution à 3000 secondes sur un pentium IV 2,8 GHz (comme ce qui a été décrit dans Alvarez-Valdes et al [67]). Les solutions de PR ont été obtenues par Alvarez-Valdés sur un pentium IV 2,8 GHz, en limitant le temps d'exécution de PR à 600 secondes. Enfin, les solutions de $HESGA_b$ ont été obtenues sur un Pentium centrino avec 2.8 GHZ et 512 Méga octets de mémoire centrale.

La table 3.5 résume les résultats obtenus par Cplex, $HESGA_b$, PR, LBS et GBS, la colonne

				Recherche par faisceaux locale : LBS						*Recherche par faisceaux globale : GBS*			
				LBS^M_β			LBS^P_β			GBS^M_β		GBS^P_β	
#Inst	Opt	HESGA[b]	PR	$\beta=1$	$\beta=2$	$\beta=3$	$\beta=4$	$\beta=6$	$\beta=1$	$\beta=2$	$\beta=3$	$\beta=4$	$\beta=6$	$\beta=1$	$\beta=2$	$\beta=1$	$\beta=2$
Horizontale																	
ATP30	140168	140168*	0*	0	134303	134303	135496	136640	132698	134303	134303	135496	136640	140168*	140168*	140168*	140168*
ATP31	820260	820260*	813935	818364	818364	818364	818364	818364	818364	818364	818364	818364	818364	819472	819472	819472	820260*
ATP32	378800	378800*	377793	378028	373212	373212	373112	373112	378028	373212	373212	373112	373112	377797	377797	377797	378800*
ATP33	235580	235580*	235580*	225569	227793	227793	234350	235580*	225569	227793	227793	227793	235580*	235580*	235580*	235580*	235580*
ATP34	356159	356159*	355660	337232	345686	345686	356159*	356159*	337232	345686	345686	345686	356159*	353606	353606	356159*	353606
ATP35	6144429	6144429*	6120820	589498	589498	589498	608824	610662	589498	587421	587421	589498	608824	6097320	6097320	6144429*	6097320
ATP36	129262	129262*	129262*	128714	129262*	129262*	129262*	129262*	128714	129262*	129262*	129262*	129262*	128651	128651	129262*	128651
ATP37	3844478	3844478*	3844478*	375256	375256	375256	383019	383019	375256	375256	375256	375256	383019	3811167	3811167	3844478*	3811167
ATP38	2590700	2590700*	2588190	2518452	2518452	2518452	2541030	2547520	2518452	2518452	2518452	2518452	2541030	2580140	2580140	2590700*	2580140
ATP39	2661350	2661350*	2661350*	2582470	2582470	2661350*	2661350*	2661350*	2582470	2582470	2661350*	2661350*	2661350*	2603820	2603820	2661350*	2603820
ATP40	639450	639450*	639450*	615610	615610	615610	622690	622912	615610	615610	615610	615610	622690	629560	629560	639450*	629560
ATP41	2023050	2023050*	2023050*	2023050*	2023050*	2023050*	2023050*	2023050*	2023050*	2023050*	2023050*	2023050*	2023050*	2023050*	2023050*	2023050*	2023050*
ATP42	325890	325890*	325890*	290130	290130	290197	311581	290130	290130	290130	290130	290197	290130	325890*	325890*	325890*	321120
ATP43	2089980	2089980*	2089980*	2066640	2066640	2066640	2066640	2066640	2066640	2066640	2066640	2066640	2066640	2089980*	2089980*	2089980*	2053010
ATP44	709400	709400*	709010*	706460	706460	706500	706500	706500	706460	706460	706460	706460	706500	709400*	709400*	709400*	684720*
ATP45	742050	742050*	742050*	742050*	742050*	742050*	742050*	742050*	742050*	742050*	742050*	742050*	742050*	742050*	742050*	742050*	742050*
ATP46	1464020	1464020*	1464020*	1305610	1305610	1305610	1305610	1464020*	1305610	1305610	1305610	1305610	1305610	1464020*	1464020*	1464020*	1464020*
ATP47	1443170	1443170*	1443170*	1332790	1332790	1332790	1377620	1406380	1332790	1332790	1332790	1332790	1377620	1406380	1406380	1443170*	1406380
ATP48	1654280	1654280*	1654280*	1549670	1549670	1549670	1654280*	1654280*	1549670	1549670	1549670	1549670	1654280*	1621820	1621820	1654280*	1621820
ATP49	2069650	2069650*	2069650*	1692110	1692110	1692110	1854800	1959060	1692110	1692110	1692110	1692110	1854800	2019390	2019390	2069650*	2019390
Verticale																	
ATP30	140197	140067	140067	133472	133472	133472	139756	139756	133472	133472	133472	138949	139756	140007	140197*	140197*	140197*
ATP31	821073	821073*	812553	803369	812280	812280	812280	812280	803369	812280	812280	812280	812280	812280	821073*	821073*	821073*
ATP32	379730	378930	379730*	354360	354360	354360	375040	377930*	354360	354360	354360	354360	377930*	376540	379730*	376540	379730*
ATP33	2346700	2346700*	2346700*	2299580	2346700*	2346700*	2346700*	2346700*	2299580	2346700*	2346700*	2346700*	2346700*	2346700*	2346700*	2346700*	2346700*
ATP34	3577410	3577410*	3557710	3310560	3510640	3529720	3536240	3510640	3310560	3510640	3510640	3529720	3510640	3362090	3577410*	3362090	3577410*
ATP35	6143360	6143360*	6059590	5780240	5780240	6105440	6105440	5780240	5780240	5780240	5780240	6105440	5780240	5963430	6143360*	6143360*	6143360*
ATP36	1288140	1288140*	1283620	1254690	1274970	1274970	1283060	1274970	1254690	1274970	1274970	1288140*	1274970	1288140*	1288140*	1288140*	1288140*
ATP37	3858110	3858110*	3851640	3610820	3858110*	3858110*	3858110*	3858110*	3610820	3858110*	3858110*	3858110*	3858110*	3826890	3858110*	3858110*	3858110*
ATP38	2591370	2591370*	2591370*	2552260	2553570	2553570	2569820	2569820	2552260	2553570	2553570	2591370*	2591370*	2561920	2591370*	2591370*	2591370*
ATP39	2663780	2663780*	2655070	2558420	2605420	2605420	2649690	2649690	2558420	2605420	2605420	2629510	2649690	2633900	2663780*	2633900	2663780*
ATP40	655840	655840*	655840*	557890	608670	608670	640150	640150	557890	608670	608670	650440	640150	605920	655840*	605920	655840*
ATP41	1965590	1965590*	1965590*	1879690	1879690	1879690	1879690	1879690	1879690	1879690	1879690	1879690	1879690	1951940	1965590*	1951940	1965590*
ATP42	330120	330120*	330120*	327580	321890	321890	321890	321890	321890	321890	321890	321890	321890	321890	330120*	321890	330120*
ATP43	2120620	2120620*	2120620*	1724260	1727780	1826440	2077780	2077780	1724260	1826440	1878970	1826440	2077780	1931150	2120620*	1931150	2120620*
ATP44	697840	697840*	697840*	590230	670180	670180	691050	691050	590230	670180	670180	670180	691050	676390	697840*	676390	697840*
ATP45	699880	699880*	699880*	568250	600290	600290	691620	691620	568250	600290	600290	600290	691620	618630	699880*	618630	699880*
ATP46	1470210	1470210*	1470210*	1337360	1431420	1431420	1431420	1431420	1337360	1431420	1431420	1431420	1431420	1410640	1470210*	1410640	1470210*
ATP47	1429350	1429350*	1429350*	1115630	1220780	1220570	1394240	1429350*	1115630	1220570	1220570	1394240	1429350*	1422560	1429350*	1422560	1429350*
ATP48	1624580	1624580*	1624580*	1465290	1495490	1499970	1532970	1538220	1465290	1624580*	1624580*	1624580*	1624580*	1547370	1624580*	1547370	1624580*
ATP49	2117840	2101690	2117840*	1644860	1839330	1897940	2005700	2117840*	1644860	1839330	1897940	1897940	2117840*	2101690	2117840*	2101690	2117840*
Opt	31	28	1	2	5	7	13	1	5	9	9	14	7	38	40	7	40
Dev	0.05	0.28	7.40	5.29	4.97	2.38	1.49	7.40	5.03	4.00	2.38	1.49	1.99	0.01	0.00	1.99	0.00
Av. T	0.39	1.26	0.01	0.01	0.02	10.39	37.20	0.01	0.03	0.05	5.89	10.20	0.01	0.01	0.20	0.01	0.20

TABLE 3.4 – Résultats numériques de LBS avec $\beta = 1, 2, 3, 4$ et 6, GBS, avec $\beta = 1$ et 2 ; utilisant les stratégies de recherche en profondeur d'abord et meilleur d'abord pour une première découpe horizontale et une première découpe verticale.

Instance	Opt/Best	IFS	$HESGA_b$	PR	cpu	Stratégie de recherche en meilleur d'abord LBS^M_β $\beta = 2$	cpu	$\beta = 4$	cpu	$\beta = 6$	cpu	GBS^M_β $\beta = 2$	cpu	$\beta = 4$	cpu
Horizontale															
nice25	860829	860829	860829	860829	0.85	815631	0.02	843044	0.02	843044	0.03	860829	0.01	860829	0.05
nice50	913738	913738	912395	904955	2.4	787166	0.03	792898	0.01	889058	0.12	913738	0.01	913738	0.01
nice100	911122	906476	900923	908734	11.45	851647	0.14	875485	0.12	875485	0.72	907826	0.05	911122	0.05
nice200	932686	924576	926287	932686	33.81	878069	0.52	878069	0.39	914215	6.36	929053	0.38	931143	0.41
nice500	952015	942548	945991	952015	289.3	929350	1.30	929350	1.19	932063	38.26	923300	1.84	951279	1.86
path25	892765	892765	892765	892765	2.1	812733	0.02	822337	0.03	822337	0.06	892765	0.03	892765	0.06
path50	750215	750215	749627	748655	2.65	709819	0.01	717737	0.03	717737	0.14	749327	0.03	750215	0.05
path100	888578	888578	887967	875592	9.3	786949	0.31	786949	0.08	826680	1.43	878839	0.09	888578	0.08
path200	889174	889174	871602	835895	30.33	622996	0.47	712948	0.55	749086	6.95	877418	1.55	889174	1.59
path500	885112	880314	881801	879465	172.7	881801	39.89	881801	41.87	881801	489.87	881801	44.09	885112	12.87
nice25P	441993	441993	441207	422002	1.14	362483	0.01	400129	0.01	400129	0.02	422539	0.02	441993	0.02
nice50P	449292	446839	444049	434199	2.99	396163	0.01	406954	0.01	406954	0.11	441159	0.02	449292	0.02
nice100P	456306	452069	452069	452904	12.92	376903	0.03	410917	0.03	410917	0.79	453385	0.03	456306	0.05
nice200P	455727	450179	450179	448480	48.41	368576	0.14	409846	0.14	409846	5.62	446203	0.16	455727	0.16
nice500P	455036	454088	454088	450802	183.84	384542	0.94	404646	0.95	404646	52.06	437897	1.41	455036	1.44
nice1000P	451805	450931	451545	451605	601.98	353019	1.98	404223	2.45	404223	284.37	416830	2.92	451805	2.94
path25P	464721	464721	460355	463693	0.67	434662	0.01	434662	0.01	434662	0.03	463793	0.01	464721	0.01
path50P	391626	391626	389812	387795	2.59	389812	0.01	389812	0.01	389812	0.15	389812	0.01	390741	0.01
path100P	456300	456300	456300	445939	14.75	374910	0.06	414422	0.07	415126	1.61	443781	0.08	456300	0.09
path200P	452402	449779	449779	426672	38.47	337653	0.16	337653	0.17	337653	7.04	405382	1.28	452402	1.34
path500P	434478	412731	430243	429127	244.86	282546	0.67	365252	0.66	365252	60.76	416743	39.28	434478	50.52
path1000P	434966	434172	434966	434966	601.53	279241	3.97	310525	8.65	310525	293.34	428394	86.98	434966	159.21
Verticale															
nice25	834157	834157	834157	820363	1.11	698581	0.01	764701	0.19	813768	0.31	834157	0.01	834157	0.01
nice50	854425	852087	854425	842696	2.11	710392	0.01	820877	0.01	824080	0.13	842696	0.01	854425	1.12
nice100	911958	890571	905085	911958	9.41	789996	0.01	872127	0.21	872127	0.73	872127	0.41	911958	3.41
nice200	935290	919515	935290	935290	44.71	760534	0.08	861254	0.09	904720	6.65	927044	4.11	935290	4.55
nice500	960767	935255	960767	960767	316.25	785925	0.21	852415	0.41	945766	36.61	941847	3.42	960767	14.54
path25	699707	699707	694142	699707	0.75	574869	0.01	666543	0.03	666543	0.04	699707	0.01	699707	0.01
path50	924908	924908	924908	915745	1.75	620435	0.01	752034	0.03	752034		924908	0.41	924908	0.41
path100	754753	754753	743339	665339	3.45	589606	0.03	639544	0.55	639544		754678	3.43	754678	3.43
path200	836022	835807	834350	803521	25.65	529479	0.41	613484	0.91	613484		752373	1.33	836022	3.56
path500	808376	784366	808345	808376	285.41	532354	0.42	612072	2.55	760816		717312	2.25	798544	7.54
nice25P	428662	428662	419659	408409	1.31	336736	0.03	383254	0.03	411336		395761	0.12	428662	0.13
nice50P	425921	425921	412766	408518	3.25	343809	0.02	362147	0.02	362147		386889	0.53	425921	0.55
nice100P	451851	449180	449180	451051	11.13	351423	0.01	376469	0.02	376469		399517	0.05	451851	0.43
nice200P	454754	453964	453964	451337	45.03	378138	0.05	392870	0.05	392870		414566	0.08	454754	1.23
nice500P	457381	457381	448071	447519	211.58	378677	0.34	393492	0.55	393492		415838	0.52	448496	14.41
nice1000P	459142	459142	433159	453665	602.66	380813	2.27	391805	2.03	391805		433159	3.34	457439	6.44
path25P	356088	356088	356088	354726	0.81	287538	0.01	317538	0.02	317538		327538	0.03	356088	0.05
path50P	487447	487447	487447	482273	2.28	268186	0.02	294145	0.05	294145		428256	0.8	487447	0.11
path100P	386033	386033	374149	356168	4.81	245677	0.01	299677	0.33	299677		320653	0.55	386033	1.23
path200P	418024	416656	409856	380095	28.77	300873	0.05	366026	0.06	366026		398915	1.54	418024	2.34
path500P	381860	368254	365202	381860	263.8	240340	0.66	286514	0.54	286514		326099	4.53	367936	19.2
path1000P	425768	405251	407097	425768	601.77	346619	2.75	356238	10.43	356238		375327	59.67	407097	87.54
Optima/Best	12	21	11	12		0		0				7		36	
Av. Dev	1.87	0.71	0.98	1.87		19.08	1.32	12.60	1.74			4.46	6.08	0.27	
Av. cpu	108.56		30.00	108.56											

TABLE 3.5 – Résultats numériques de LBS^M_β et GBS^M_β avec $\beta = 2$ et 4 ; utilisant les stratégies de recherche en meilleur d'abord pour une première découpe horizontale et une première découpe vertical.

2 de la table 3.5 donne, la solution optimale de l'instance ; si elle est connue, sinon elle exprime la meilleure solution obtenue par un des algorithmes (les instances dont la solution optimale n'est pas connue sont signalées par $*$). La colonne 3 exprime la déviation entre la solution obtenue et la meilleure solution entière réalisable obtenue par Cplex. La colonne 4 exprime la différence entre la solution obtenue et la solution obtenue par $HESGA_b$. Les colonnes 5 et 6 rapportent la déviation de la solution obtenue par PR par rapport à la meilleure solution, ainsi que le temps de calcul. Les colonnes 7, 9, 11, 8,10 et 12 présentent la déviation de la solution obtenue par LBS_β^M(avec $\beta = 2, 4, 6$) par rapport à la meilleure solution, ainsi que le temps de résolution correspondant. Les colonnes 13, 15, 14 et 16 rapportent la déviation de la solution obtenue par GBS_β^M (avec $\beta = 2$ et $\beta = 4$) par rapport à la meilleure solution et les temps de résolution.

Les trois dernières lignes de la table 3.5, résument le nombre de fois que l'algorithme a trouvé l'optimum, s'il est connu ou le nombre d'améliorations effectuées ($Optima/Best$), la moyenne de déviation ($Av.Dev$) et la moyenne du temps d'exécution ($Av.cpu$).

Généralement la qualité de la solution produite par LBS s'améliore quand on augmente la largeur du faisceau, cependant, cela implique l'augmentation de la durée de traitement. Par conséquent, LBS ne peut pas être utilisée comme une heuristique standard pour des grandes instances. En revanche, GBS_2^M produit de meilleures solutions pour 7 instances, avec un temps de résolution moyen de 6.08 secondes. Avec $\beta = 4$, GBS_4^M produit la meilleure solution dans 36 cas (avec une première découpe horizontale et verticale) dans un temps moyen de 9.21 secondes, Il est à noter que GBS_4^M améliore les solutions réalisables de seulement 8 instances. La déviation moyenne de GBS^M par rapport aux meilleures solutions connues est de $4,46\%$ pour $\beta = 2$ et 0.27% pour $\beta = 4$; le temps de résolution moyen est de 6.08 secondes pour $\beta = 2$ et 9.21 secondes pour $\beta = 4$. Le temps d'exécution moyen a augmenté à cause de la difficulté particulière de certaines instances.

La table 3.6 montre que GBS_2^M améliore les résultats de LBS_6^M dans 33 instances sur les 44 traitées (en incluant les deux positions de la première découpe) et obtient la même solution sur 7 instances. En outre, GBS_4^M améliore les solutions obtenues par GBS_2^M sur 38 instances sur 44. Ces améliorations augmentent, évidemment, les durées de calcul (6.07 secondes pour GBS_2^M pour 9.21 secondes pour GBS_4^M). On note que la comparaison des performances de GBS^M et LBS^M sur les instances pondérées et non-pondérées reste inconcluant car pour quelques instances pondérées, la résolution est plus efficace, alors qu'elle est plus difficile pour d'autres.

GBS_4^M résout mieux que PR, $HESGA_b$ et Cplex avec un temps de résolution moyen largement inférieurau temps donné par PR (108.56). GBS_4^M améliore plus de solutions que PR (36) qui ne produit la meilleure solution que dans 12 cas sur 44 traitées. GBS_4^M produit une déviation inférieure à celle de PR (0.27% pour GBS_4^M et 1.87% pour PR).

Le temps de résolution moyen de $HESGA_b$ est de 30 secondes, sa déviation de la meilleure solution est de 0.98% et il produit 11 meilleures solutions, cela rend GBS_4^M meilleur que $HESGA_b$.

GBS_4^M résout mieux que Cplex en produisant de meilleures solutions (36 versus 21 sur 44) , une moindre déviation (0.27% versus 0.71%) et une meilleure performance en temps de résolution (9.21 secondes versus 3000 secondes).

	Stratégie de recherche en meilleur d'abord			
	LBS^M_β		GBS^M_β	
	$\beta = 2$	$\beta = 4$	$\beta = 2$	$\beta = 2$
Optima/Best	0	0	7	35
Av. Dev	19.08	12.60	4.46	0.27
Av. cpu	1.32	1.74	6.08	9.21

TABLE 3.6 – Résultats numériques de HESGA, PR, LBS et GBS. β est fixé à 2 dans LBS et GBS avec une stratégie de recherche en meilleur d'abord.

3.4 Conclusion

Dans ce chapitre, nous avons proposé une résolution approchée du problème de découpe contraint à deux dimensions et à deux niveaux avec une nouvelle méthode de recherche par faisceaux. Cette méthode est une méthode de séparation et évaluation qui explore un sous-ensemble de nœuds à chaque niveau. Le choix des nœuds à explorer se fait en utilisant une fonction d'évaluation nommée aussi, politique de filtrage. La partie expérimentale menée sur des instances moyennes et larges a montré l'intérêt de cet opérateur et son influence directe sur la qualité des solutions réalisables produites et du temps d'exécution.

Chapitre 4

Algorithme coopératif pour le problème de découpe contraint à deux dimensions et à deux niveaux [2]

Contenu

Au fil des années, la puissance de calcul disponible n'a cessé de croître, ce qui a permis de mettre eu point des heuristiques et métaheuristiques de plus en plus complexes. Depuis quelques années, un nombre croissant de méthodes d'optimisation de la littérature proposent de faire coopérer plusieurs heuristiques ou métaheuristiques. Actuellement, les coopérations s'effectuent aussi bien entre heuristiques qu'entre métaheuristiques et méthodes exactes. Ces coopérations permettent d'obtenir des méthodes d'optimisation efficaces sur des problèmes de plus en plus difficiles, et en particulier sur les problèmes de découpe et de placement. L'intérêt de ces approches coopératives est de permettre à différentes méthodes d'optimisation d'allier leurs atouts, dans le but d'améliorer les performances globales obtenues par chacune d'elles. Actuellement, poussées par les performances générales de tels algorithmes, un nombre croissant d'études proposent ce type d'approche pour divers problèmes. Dans ce chapitre, nous proposons un schéma de coopération entre l'algorithme GBS présenté dans le chapitre précédent et une méthode de remplissage. En effet, lors de l'optimisation d'un problème, deux buts sont contradictoires : l'amélioration des solutions existantes

2. *Le contenu de ce chapitre est à paraître dans "International Journal of Operational Research [47]" et a fait l'objet d'une conférence internationale "IEEE, SSSM, Service Systems and Service Management [44]"*

(intensification) et la découverte de nouvelles régions de recherche (diversification).
L'idée principale de l'algorithme proposé est de faire coopérer l'algorithme GBS dans ses deux variantes (stratégie de recherche en meilleur d'abord et en profondeur d'abord) avec une méthode complémentaire qui va contribuer à la découverte de nouvelles régions dans l'espace de recherche et diversifier, ainsi, les chemins développés par GBS. La méthode complémentaire va aussi contribuer au calcul d'une borne supérieure plus fine, qui va améliorer la performance de la stratégie de recherche en profondeur d'abord de GBS sur de grandes instances.

4.1 Quelques méthodes de remplissage

4.1.1 Motivation

Le but des procédures de remplissage est de sélectionner la meilleure bande optimale afin de remplir un sous rectangle donné et en particulier le rectangle initial S. Concernant le problème FC2TDC, nous avons décrit dans le chapitre 2 de cette thèse plusieurs méthodes qui réalisent cette tâche et qui combinent plusieurs bandes optimales pour remplir un sous-rectangle (Hifi et Roucairol [37], Hifi et M'Hallah [39], Alvarez-Valdes et al [67]). Toutes ces méthodes se basent sur la génération de bandes générales optimales en modélisant sous forme de problème de sa-à-dos (voir chapitre 2), cependant chaque méthode utilise une stratégie de remplissage différente.

4.1.2 Stratégies de remplissage

Dans la suite de cette section, on décrit le principe des stratégies mises en œuvre par ces méthodes de remplissage.

La première stratégie, utilisée par Hifi et M'Hallah [39] et Hifi et Roucairol [37] est la construction d'un plan de découpe qui réalise la meilleure combinaison linéaire entre les hauteurs des bandes considérées pour un sous-rectangle.

La deuxième stratégie,utilisée par Hifi et M'Hallah [39] génère l'ensemble des combinaisons linéaires des hauteurs des bandes considérées pour un sous-rectangle et applique à chaque élément de cet ensemble, les étapes suivantes :

(i) continuer la construction d'une solution sur les bandes déjà placées (L, y) en remplissant le sous-rectangle $(L, W - y)$;

(ii) définir à chaque placement d'une bande, un nouvel ensemble de bandes optimales par l'application de SGP (voir section 2.2.1) ;

La troisième stratégie utilise le processus précédent et remplace l'étape (ii) par une procédure BLP (voir section 2.2.4).

La quatrième stratégie considère une procédure constructive avec les étapes suivantes :

(i) placer une bande dans le sous-rectangle courant ;

(ii) réduire le sous-rectangle courant et mettre à jour les demandes des pièces ;

(iii) répéter les étapes précédentes (1 et 2) jusqu'à ce qu'aucune bande ne peut être placée ;

Pour cette stratégie, deux solutions sont possibles, la première consiste à construire les bandes par une procédure BLP et la deuxième utilise la procédure SGP afin de construire des bandes optimales.

La dernière stratégie est la méthode GRASP par bandes (voir section 2.2.4) proposée par Alvarez-Valdes et al [67] qui est équivalente à la dernière stratégie avec une amélioration qui consiste à parcourir la solution générée par la phase de construction, bande par bande, et tenter de remplacer la bande par une ou plusieurs bandes.

4.1.3 Une procédure de remplissage de base (BFP)

Dans cette section, on propose une procédure de remplissage qui permet de remplir un sous-rectangle. Cette procédure peut être considérée comme une amélioration de la procédure proposée par Hifi et M'Hallah [39]. Le principe de BFP [1] est le suivant : Soit w, $w \leq W$, la hauteur du sous-rectangle courant (L, w), r, $r \leq n$ le nombre de bandes optimales de hauteur $\overline{w}_j \leq w$, $S_{\overline{w}_j}$ l'ensemble de pièces de la bande $(L, \overline{w}_j)$, où $\overline{w}_j \leq w$ et $f_{\overline{w}_j}(L)$ la valeur de la solution associée à la bande $(L, \overline{w}_j)$.

Le plan de découpe associé au sous-rectangle (L, w) de profit $h_L(w)$ est la solution optimale du programme linéaire suivant :

$$(IP_{L,w}) = \begin{cases} h_L(W) = \max \sum_{j=1}^{r} f_w(L) y_i \\ \text{s.c.} \sum_{j=1}^{r} \overline{w}_j y_j \leq w \\ \sum_{j=1}^{r} x_{ij} y_j \leq b_i, i \in I \\ y_j \leq a_j, y_j \in N, j = 1, ..., r, \end{cases}$$

où x_{ij} représente le nombre d'occurrences de la ième pièce dans la jème bande de dimension $(L, \overline{w}_j)$ et

$$a_j = min\Big\{ \left\lfloor \frac{\overline{w}_p}{\overline{w}_j} \right\rfloor, min_{i \in f_{\overline{w}_j}} \left\lfloor \frac{b_i}{x_{ij}} \right\rfloor \text{ avec } x_{ij} > 0 \Big\}$$

Le problème $IP_{L,w}$ étant NB-difficile, on le résout par la procédure d'approximation suivante :

1. Mettre $b_i^{reste} = b_i$,où b_i^{reste} représente le reste des demandes de la pièce i.
2. Sélectionner la bande réalisable k avec le plus grand profit selon l'équation suivante :

$$\frac{f_{\overline{w}_k(L)}}{\sum_{i \in S_{\overline{w}_k}} x_{ij}} = max_{j=1,...,s} \left\{ \frac{f_{\overline{w}_j(L)}}{\sum_{i \in S_{\overline{w}_k}} x_{ij}} \right\}$$

   avec s, $s \leq r$ le nombre de bandes non placées.
3. Placer la bande K dans S.
4. Pour chaque pièce i dans la bande k, si $b_i^{reste} \leq x_{ij}$, alors réduire le nombre de pièce de type i dans la bande k à b_i^{reste} et mettre $x_{ij} = b_i^{reste}$
5. Mettre à jour les demandes de toutes les pièces i de la bande k, cela signifie $b_i^{reste} = b_i^{reste} - x_{ij}$; supprimer les pièces ayant une demande nulle dans les futures constructions.
6. Réordonner les pièces dans k dans un ordre décroissant des hauteurs
7. Remplir les régions restantes de la bande k en utilisant la procédure Bottom-left-guillotine procedure (BLGP)
8. Mettre à jour le reste de la hauteur du rectangle initial S ; $W = W - \overline{w}_k$
9. S'il existe $b_i^{reste} > 0$, $i \in I$ et $w_i \leq W$, alors
   (a) Si toutes les bandes sont considérées, alors répéter BLGP sur le reste du sous-rectangle
   (b) Sinon aller à l'étape 2

1. *Basic Filling Procedure*

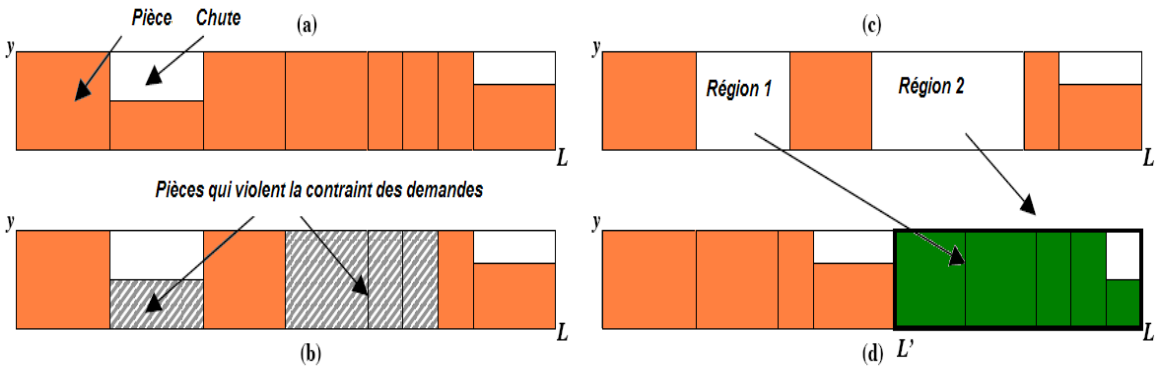


FIGURE 4.1 – Processus BLGP.

4.1.4 Une procédure de résolution complémentaire (BLGP)

BLGP [2] [3] est une méthode gloutonne qui remplit une bande en utilisant les pièces ayant des demandes positives. Pour comprendre comment BLGP procède, on considère la bande générale (L, w) de la figure 4.1.(a). La première phase de BLGP supprime les pièces qui violent la contrainte sur les demandes (4.1.(b)). La deuxième phase remplit la région générée par la première phase, le résultat de la suppression des régions 1 et 2 dans 4.1.(c) est représentée par la région 3 de la figure 4.1.(d). Afin que la deuxième phase remplit la région 3, elle procède à quelques décalages des pièces de la bande et remplit la partie de la bande $(L - L', y)$ par la résolution du problème de sac-à-dos suivant :

$$max\Big\{ \sum_{i \in S_y} c_i z_i \;\Big|\; \sum_{i \in S_y} l_i z_i \leq L - L^{'}, x_i \leq b_i^{reste}, x_i \in N, i \in S_y \Big\},$$

où $S_y = p|(l_p, w_p) \leq (L, L', y), p \in I$.

Naturellement BLGP peut être utilisé comme heuristique sur le rectangle initial S.

4.2 Coopération entre GBS et BFP

L'algorithme coopératif qu'on propose combine l'algorithme GBS (présenté dans le chapitre précédent) et la procédure de remplissage BFP (présentée dans la section 4.1.3) Comme présenté dans le chapitre précédent, un nœud u est représenté par deux sous-rectangles $((L, W - y), (L, y))$, où $(L, W - y)$ est la solution réalisable locale et (L, y) le sous-rectangle à compléter. En d'autres termes, le sous-rectangle $(L, W - y)$ est déjà remplie par GBS alors que (L, y) reste à remplir par les prochains développements (branchements) de GBS.

L'algorithme coopératif utilise une fonction d'évaluation qui combine trois solutions, comme suite :

Soit w une découpe horizontale sur le sous-rectangle complémentaire (L, y), avec $w \leq Y$ et $w \in$

2. *Bottom Left Guillotine Procedure*

3. *Cette méthode a été présenté lors de la Conférence scientifique conjointe en Recherche Opérationnelle et Aide à la Décision, −FRANCOROV/ROADEF2007− [46]*

$\overline{w}_1, ..., \overline{w}_n$. Soit $v = ((L, W-y+w), (L, Y-w))$ un des nœuds résultat de cette découpe. L'algorithme coopératif combine les trois points suivants sur le nœud v.

1. $\hat{Z}_v^{Local}$ la valeur de la solution réalisable du sous-rectangle $(L, W-y+w)$.
2. $BFP_{(L,Y-w)}$ une solution réalisable complémentaire du rectangle initial S.
3. $\overline{U}_{(L,y)}$ une borne supérieure du sous-rectangle complémentaire $(L, y-w)$.

4.3 L'algorithme CGBS

Comme décrit dans la section précédente, l'algorithme coopératif (CGBS [4]) est un algorithme de recherche en faisceaux avec une nouvelle fonction d'évaluation. la fonction d'évaluation de GBS est enrichie par les constructions de la procédure BFP. Dans cette section, on présente le mécanisme de construction des nœuds et comment CGBS gère et réduit son espace de recherche.

4.3.1 Mécanisme de construction des nœuds

En se basant sur le principe de construction de bandes optimales, on propose de généraliser ce principe sur toutes les bandes. Cela est possible en utilisant le principe de la programmation dynamique.

Supposons que la bande (L, w) est placée dans (L, Y), avec $w \leq Y$, $w \in \overline{w}_1, ..., \overline{w}_n$ et (L, w) est construite suivant la solution optimale de $BK^g_{L,w}$. Le nouveau nœud crée peut être présenté par la paire de sous-rectangles $((L, W-Y+w), (L, Y-w))$ avec $w = \overline{w}_1, ..., \overline{w}_s$ et $s \leq n$.
Avant de développer le nœud précédent, noté $u = ((L, W-Y+w), (L, Y-w))$, CGBS applique les étapes suivantes :

1. Mettre à jour les demandes résiduelles de toutes les pièces i, $i \in I$.
2. Générer s, $(s \leq n)$ bandes générales par la résolution du problème $BK^g_{L,w}$ par programmation dynamique en utilisant les demandes mises à jour (on rappelle que s représente le nombre de pièces qui constituent le sous-rectangle (L, Y)).
3. Pour chaque bande $(L, \overline{w}_p)$, $p = 1, ..., s$ obtenue
   (a) Placer la bande dans le sous-rectangle (L, Y) ;
   (b) Obtenir un nouveau nœud $v = ((L, W-Y+\overline{w}_p), (L, Y-\overline{w}_p))$;
   (c) Effectuer une sélection (en utilisant la largeur du faisceau β) et la borne supérieure du sous-rectangle complémentaire $(L, Y-\overline{w}_p)$;
   (d) Appliquer BFP à $(L, Y-\overline{w}_p)$ et mettre à jour la valeur du nœud v.
   (e) Utiliser la valeur de la solution réalisable locale de l'étape $3d$ pour insérer les nœuds de la liste globale.

On rappelle que la borne supérieure globale utilisée dans l'étape $3c$ (pour chaque sous-rectangle (L, α), $\alpha = 0, \ldots, W$) se calcule en se basant sur la solution du problème de sac-à-dos borné suivant :

$$BK_{(L,\alpha)} = \Big\{ max \sum_{j=1}^{s} f_{\overline{w}_j}(L) z_j \leq \alpha \Big| \sum_{j=1}^{s} w_i z_j \leq \alpha, z_j \leq b_j, j = 1, ..., s \Big\},$$

4. *Cooperative Global Beam Search*

où $f_{\overline{w}_j}(L)$ est le profit génère par la bande $(L, \overline{w}_j)$, z_j est le nombre d'occurrences de la bande (L, α). La valeur $\overline{U}_{(L,y)}$ est obtenue par la résolution du problème $BK_{(L,\alpha)}$ précédent avec $\alpha = W$. Comme nous avons noté dans le chapitre précédent, la résolution du problème $BK_{(L,W)}$ permet de calculer les bornes supérieures $\overline{U}_{(L,\alpha)}$ de tous les sous-rectangles possibles (L, α), avec $\alpha = 0, 1, ..., W$.

On constate également que la procédure proposée utilise une borne supérieure du chemin courant, qui peut être transmise du nœud courant à ses successeurs. Obtenir cette borne est très intéressant car elle peut être meilleure que la borne globale. Cependant le calcul de cette borne nécessite la résolution du problème $BK_{(L,\alpha)}$ (en utilisant la programmation dynamique ou même une procédure de séparation et évaluation). Dans ce cas, la mise à jour des demandes résiduelles des pièces est nécessaire et la valeur de la hauteur α doit être positionnée à $(Y, \overline{w}_p)$.

4.3.2 Gestion du processus de recherche

Afin d'accélérer le processus de recherche, nous avons introduit deux stratégies. La première stratégie troncature l'espace de recherche et la deuxième est appliquée dans la sélection des meilleurs chemins à travers les nœuds élites.

4.3.2.1 Réduction de l'espace de recherche

Afin de réduire l'espace de recherche, On néglige chaque nouveau nœud $u = ((L, W-y), (L, y))$ de B avec :

$$\hat{Z}_u^{Global} = \hat{Z}_u^{Local} + \overline{U}_{(L,y)} \leq Z^* \tag{4.1}$$

avec Z^* la meilleure solution réalisable courante.

4.3.2.2 Section des nœuds

Afin de sélectionner les nœuds élites, on définit pour l'ensemble B, le sous-ensemble η avec

$$\eta = max_{u \in B} \left\{ \hat{Z}_u^{Global} \right\} \tag{4.2}$$

Pour chaque nœud $u \in \eta$, on calcule l'écart entre $\hat{Z}_u^{Global}$ et η, on sélectionne, ensuite, $min\{\beta, |M|\}$ ayant le plus grand écart. Dans le cas où plusieurs nœuds de η gérèrent les mêmes écarts, on sélectionne le nœud ayant le plus grand $\hat{Z}_u^{Local}$.

L'écart entre $\hat{Z}_u^{Global}$ et η est calculé par l'équation suivante : $Dev = 100\frac{\hat{Z}_u^{Global} - \eta}{\hat{Z}_u^{Global}}$

4.3.3 L'algorithme CGBS

La figure 4.2 décrit les étapes principales de CGBS appliquées au FC2TDC, CGBS est composé de trois phases principales. La phase d'initialisation, dans laquelle l'algorithme génère toutes les bandes générales par la résolution du problème de sac-à-dos $KP^g_{(L,\overline{w}_j)}$ (voir chapitre 3). Dans cette phase, l'algorithme calcule, aussi, une solution réalisable de départ z^* qui correspond à la résolution

de $IP_(L, W)$ sur le nœud racine.
On note qu'on peut appliquer aussi BLGP sur le rectangle initial S et considérer la meilleure des deux solutions comme solution de départ. Les phases itératives qui opèrent à chaque nœud sont composées de trois étapes :

1. La première étape sert à générer toutes les bandes générales valides sur le nœud (toutes les bandes ayant des hauteurs inférieures ou égales à Y associées à un nœud u).
2. La deuxième étape répète des sélections des nœuds de l'ensemble B et (a) procède à une évaluation globale du nœud et au calcul de la borne supérieure, ainsi qu'à une solution complémentaire. (b) met à jour les nœuds élites et (c) troncature les nœuds ayant une estimation inférieure ou égale à z^*.
3. La dernière étape sert à contrôler la convergence de l'algorithme coopératif.

4.4 Résultats numériques

Dans cette section, on évalue la performance de l'algorithme proposé sur 48 instances, classées dans la littérature comme moyennes, larges et extra-larges. Nous avons codé et testé toutes les approches en C++ sur un pentium centrino 2.8 GHZ et 512 Méga octets de mémoire centrale (Nous avons limité le temps d'exécution à 600 secondes).

4.4.1 Les instances

En plus des instances traitées dans le chapitre précédent, nous avons traité 6 nouvelles instances, considérées extra-larges (ces instances ont été extraites de Hifi et M'Hallah [39]. De la même manière, chaque instance est traitée en horizontale (la première découpe est horizontale) et en verticale (la première découpe est verticale). La table 4.1, présente la dimension du rectangle initial L, W et le nombre de types de pièces de chaque nouvelle instance n. On notre que l'optimum de ces instances n'est pas connu.

Inst	Type	L	W	n	$OptH$	$OptV$
LU1	S	927	964	89	*	*
LU2	S	927	983	133	*	*
LU3	S	1019	1005	127	*	*
LW1	P	1007	1009	125	*	*
LW2	P	1104	1097	110	*	*
LW3	P	1159	1150	121	*	*

TABLE 4.1 – Instances extra-larges

4.4.2 Objectifs

L'algorithme CGBS requiert deux décisions : (i)la valeur β associée à la recherche en faisceaux et (ii) la fréquence du calcul de la borne supérieure γ.
Dans le but de garder le même degré de comparaison, nous avons lancé les deux algorithmes (GBS et CGBS) avec les même paramètres β et afin de mesurer l'influence du paramètre γ, nous avons également considéré le même ensemble de fréquences de calcul de la borne supérieure pour chaque

`Entrée` : Une instance FC2TDC.
`Sortie` : Une solution approximative de valeur $Best_{(L,W)}$.

Phase 1 : Initialisation

- Poser $B = \Big\{((L,0),(L,W))\Big\}$ et $B_\delta = \emptyset$;
- Générer toutes les bandes générales valides qui correspondent au rectangle initial (L,W), par la résolution du problème $BK^g_{L,\overline{w}_n}$ (et/ou BLGP) ;
- Poser $z^* = h_L(W)$, où $h_L(W)$ est la valeur de la solution de $IP_{(L,W)}$;

Phase 2 : Phase Iterative

- Sélectionner de B un nœud $\mu = \Big\{(L,W-Y),(L,Y)\Big\}$ en utilisant la stratégie de recherche *meilleur d'abord* ou *profondeur d'abord* ;

1. Brancher sur μ et générer les bandes générales valides qui correspondent à la solution de $BK^g_{L,\overline{w}_s}$, où $\overline{w}_s$ présente la plus grande pièce (avec le plus grand index) capable d'entrer dans le sous-rectangle (L,Y) ;
2. Pour chaque nouveau nœud génère $\nu = \Big((L,W-Y-\omega),(L,Y+\omega)\Big),\ \omega = \overline{w}_1,\ldots,\overline{w}_s,$;
   (a) Procéder à une évaluation globale basée sur *la borne supérieure* et *la valeur de la solution obtenue par BFP* ;
   (b) Mettre à jour *la meilleure solution courante* comme suivant :
   $$z^* = \max\Big\{z^*,\ \hat{z}^{Local}_\nu + BFP_\nu\Big\};$$
   (c) Insérer le nœud crée ν dans B_δ si $z^* < \overline{U}_\nu + \hat{z}^{Local}_\nu$;
   Ignorer ν sinon ;
3. Phase de sélection :
   (a) Effectuer un filtrage des bandes générées par l'application de l'équation (4.1), soit B_δ l'ensemble des nœud élites produit ;
   (b) Insérer les $\min\{\rho,|B_\delta|\}$ meilleurs nœuds $-$*correspondent au sous-rectangle complémentaire* $-$ de B_δ dans B, qui réalisent la meilleures évaluation selon l'équation (4.2) ;

- Supprimer le nœud μ de B et réduire l'ensemble B_δ à un ensemble vide ;

Phase 3 : Condition d'arrêt

Si $B = \emptyset$, Arrêter ; sinon répéter la *Phase itérative* ;

FIGURE 4.2 – Un algorithme coopératif algorithme pour FC2TDC.

algorithme $\gamma \in 0,1,2,3$. La valeur 0 signifie que la calcul de la borne supérieure se fait uniquement sur le nœud racine (au début de l'algorithme). La valeur 1 (respectivement 2 et 3) signifie que la programmation dynamique est lancée au début de l'algorithme et lorsque un nœud (v) atteint la moitié (respectivement le tiers et le quart) de W (La hauteur du rectangle initial).

4.4.3 Premier groupe d'instances

Nous avons commencé notre étude sur l'ensemble de 20 instances extraites de Alvarez-Valdes et al [67]. Les solutions optimales de ces instances sont connues et sont rapportées dans le même article. Dix de ces instances sont non-pondérées (APT30,...,APT39) et les dix autres sont pondérées (APT40,. . . ,APT49).
Nous avons lancé les deux algorithmes sur les instances moyennes en considérant $\beta \in 1, 2$, les deux stratégies de recherche (en profondeur d'abord et en meilleur d'abord) et en fixant la fréquence $\gamma = 1$.
Les trois dernières colonnes de la table 4.2, présentent le nombre d'instances résolues à l'optimum ($NOpt$), la moyenne du pourcentage de déviation par rapport à la l'optimum (Dev) et la moyenne du temps de résolution en seconde (AvT). Le pourcentage de déviation Dev d'une solution z^* par rapport à l'optimum Opt est calculé par l'équation $Dev = 100\frac{Opt-z^*}{Opt}$.

	GBS				CGBS			
	meilleur d'abord		*profondeur d'abord*		*meilleur d'abord*		*profondeur d'abord*	
	$\beta = 1$	$\beta = 2$	$\beta = 1$	$\beta = 2$	$\beta = 1$	$\beta = 2$	$\beta = 1$	$\beta = 2$
NOpt	7	38	7	40	7	40	7	40
Dev	2.26	0.05	2.26	0.00	2.26	0.00	2.26	0.00
AvT	0.01	0.01	0.01	0.2	0.01	0.01	0.01	0.1

TABLE 4.2 – Performance de CGBS et GBS sur les instances moyennes.

L'analyse de la table 4.2 indique que les deux algorithmes avec les deux stratégies de recherche obtiennent des résultats tellement satisfaisants qu'il n'est pas nécessaire d'augmenter la largeur du faisceau β (Nous avons utilisé le symbole $\circ$ lorsqu'un algorithme produit la meilleure solution).
On observe à travers la table 4.2, que GBS (respectivement CGBS) avec la stratégie de recherche en profondeur d'abord nécessite 0.2 (respectivement 0.1) secondes pour produire la solution finale. Le temps de résolution avec la stratégie de recherche en meilleur d'abord est moins important (On note A^P_β l'algorithme utilisant une stratégie de recherche en profondeur d'abord avec une largeur β et A^M_β l'algorithme utilisant une stratégie de recherche en meilleur d'abord avec une largeur β).
On remarque également que :

- GBS^P_2 et $CGBS^P_2$ produisent le même nombre de solutions optimales, mais $CGBS^P_2$ est plus rapide que GBS^P_2 (0.2 secondes pour 0.1 secondes).
- $CGBS^M_2$ produit plus de solutions optimales que GBS^M_2 (38/40 pour 40/40 instances), mais le temps de résolution est le même pour les deux algorithmes (0.01 secondes).

La table 4.3 rapporte le détail des résultats obtenus par les algorithmes PR, GBS et CGBS sur les instances moyennes.

4.4.4 Deuxième groupe d'instances

Le deuxième groupe d'instances est composé de deux ensembles, les dix premières instances sont pondérées (Nice25,. . . , Nice500, Path25,. . . , Path500) et ont été extraites de Alvarez-Valdes et al [67], dix autres instances (Nice25P,. . . , Nice500P, Path25P,. . . , Path500P) ont été obtenues par une transformation des 10 premières instances en instances non-pondérées et finalement les deux dernières (Nice1000P, Path1000P) sont de très grandes instances contenant 1000 pièces. La solution

			GBS				*CGBS*			
			GBS^M_β		GBS^P_β		$CGBS^M_\beta$		$CGBS^P_\beta$	
#Inst	Opt	PR	$\beta=1$	$\beta=2$	$\beta=1$	$\beta=2$	$\beta=1$	$\beta=2$	$\beta=1$	$\beta=2$
Horizontale										
ATP30	140168	◦	◦	◦	◦	◦	◦	◦	◦	◦
ATP31	820260	813935	819472	819472	819472	◦	819472	◦	819472	◦
ATP32	37880	37793	37797	37797	37797	◦	37797	◦	37797	◦
ATP33	235580	◦	◦	◦	◦	◦	◦	◦	◦	◦
ATP34	356159	355660	353606	◦	353606	◦	353606	◦	353606	◦
ATP35	614429	612082	609732	◦	609732	◦	609732	◦	609732	◦
ATP36	129262	◦	128651	◦	128651	◦	128651	◦	128651	◦
ATP37	384478	◦	381167	◦	381167	◦	381167	◦	381167	◦
ATP38	259070	258819	258014	◦	258014	◦	258014	◦	258014	◦
ATP39	266135	◦	260382	◦	260382	◦	260382	◦	260382	◦
ATP40	63945	◦	62956	◦	62956	◦	62956	◦	62956	◦
ATP41	202305	◦	◦	◦	◦	◦	◦	◦	◦	◦
ATP42	32589	◦	32112	◦	32112	◦	32112	◦	32112	◦
ATP43	208998	◦	205301	◦	205301	◦	205301	◦	205301	◦
ATP44	70940	70901	68472	◦	68472	◦	68472	◦	68472	◦
ATP45	74205	◦	◦	◦	◦	◦	◦	◦	◦	◦
ATP46	146402	◦	◦	◦	◦	◦	◦	◦	◦	◦
ATP47	144317	◦	140638	◦	140638	◦	140638	◦	140638	◦
ATP48	165428	◦	162182	◦	162182	◦	162182	◦	162182	◦
ATP49	206965	◦	201939	◦	201939	◦	201939	◦	201939	◦
Verticale										
ATP30	140197	140067	140007	◦	140007	◦	140007	◦	140007	◦
ATP31	821073	812553	812280	◦	812280	◦	812280	◦	812280	◦
ATP32	37973	◦	37654	◦	37654	◦	37654	◦	37654	◦
ATP33	234670	234670	◦	◦	◦	◦	◦	◦	◦	◦
ATP34	357741	355571	336209	◦	336209	◦	336209	◦	336209	◦
ATP35	614336	605959	596343	◦	596343	◦	596343	◦	596343	◦
ATP36	128814	128362	◦	◦	◦	◦	◦	◦	◦	◦
ATP37	385811	385164	382689	◦	382689	◦	382689	◦	382689	◦
ATP38	259137	◦	256192	◦	256192	◦	256192	◦	256192	◦
ATP39	266378	265507	263390	◦	263390	◦	263390	◦	263390	◦
ATP40	65584	◦	60592	◦	60592	◦	60592	◦	60592	◦
ATP41	196559	◦	195194	◦	195194	◦	195194	◦	195194	◦
ATP42	33012	◦	32189	◦	32189	◦	32189	◦	32189	◦
ATP43	212062	◦	193115	◦	193115	◦	193115	◦	193115	◦
ATP44	69784	◦	67639	◦	67639	◦	67639	◦	67639	◦
ATP45	69988	69989	61863	◦	61863	◦	61863	◦	61863	◦
ATP46	147021	◦	141064	◦	141064	◦	141064	◦	141064	◦
ATP47	142935	◦	142256	◦	142256	◦	142256	◦	142256	◦
ATP48	162458	◦	154737	◦	154737	◦	154737	◦	154737	◦
ATP49	211784	◦	210169	◦	210169	◦	210169	◦	210169	◦
Av. T		1.26	0.01	0.01	0.01	0.2	0.01	0.01	0.01	0.1

TABLE 4.3 – Résultats numériques de PR, GBS et CGBS avec $\beta \in \{1, 2\}$, utilisant les stratégies de recherche en profondeur d'abord et meilleur d'abord pour une première découpe horizontale et une première découpe verticale.

optimale de la plupart de ces instance, n'est pas connue, par conséquent, nous avons comparé les résultats de GBS et CGBS aux meilleures bornes inférieures qui existent dans la littérature pour ces instances, notamment, le solver Cplex V9, PR, $HESGA_b$.

Les solutions du solveur Cplex, ont été obtenues en utilisant le modèle de programmation linéaire M1 (Lodi et Monaci [3]) présenté dans le chapitre 2 et en limitant le temps d'exécution à 3000 secondes sur un pentium IV 2,8 GHz (comme ce qui a été décrit dans Alvarez-Valdes et al [67]). Les solutions de PR ont été obtenues par Alvarez-Valdés sur un pentium IV 2,8 GHz, en limitant le temps d'exécution de PR à 600 secondes. Finalement, les solutions de $HESGA_b$ ont été obtenues sur un pentium centrino avec 2.8 GHZ et 512 Méga octets de mémoire centrale.

La table 4.4 résume les résultats obtenus par le solveur Cplex et les algorithmes HESGA^b, PR, GBS et CGBS sur ce groupe d'instances. La table 4.6 détaille tous les résultats.
La colonne 2 de la table 4.6 donne, si connue, la solution optimale de l'instance, sinon elle exprime la meilleure solution obtenue par un des algorithmes (les instances dont la solution optimale n'est pas connue sont signalées par $*$). La colonne 3 donne la meilleure solution entière réalisable obtenue par Cplex. La colonne 4 exprime la solution obtenue par $HESGA_b$ et la colonne 5 rapporte la solution obtenue par PR. Les deux dernières colonnes expriment les solutions obtenues par GBS et CGBS avec $\beta = 4$. Nous avons, par ailleurs, utilisé le symbole $\circ$ lorsqu'un algorithme correspondant produit la meilleure solution réalisable.

	Cplex	HESGA^b	PR	GBS	CGBS
Nb. Opt/Best	19	10	12	27	44
Av. Dev.	0.9920	0.9893	0.9805	0.9963	1
Av . T.	–	–	108.56	7.91	3.58

TABLE 4.4 – Résultats numériques de PR, GBS et CGBS, avec $\beta \in \{1.2\}$ utilisant les stratégies de recherche en profondeur d'abord et meilleur d'abord pour une première découpe horizontale et une première découpe verticale.

γ	0	1	2	3
Av. Dev	0.99939	1	1	1
Av. T.	3.58	3.58	4.41	5.16

TABLE 4.5 – Sensibilité des solutions produites et le temps de résolution lorsqu'on varie le paramètre γ de 0 à 3.

Pour ce groupe d'instances, nous avons considéré les meilleures versions de GBS et CGBS avec une stratégie de recherche en meilleur d'abord et en fixant $\beta = 4$. Nous avons également fixé $\gamma = 1$. A partir des résultats de la table 4.4, on remarque que :

1. GBS est capable de produire la meilleure solution sur 27 instances (sur les 44 instances traitées) avec un temps moyen de résolution de 7.91 secondes.
2. CGBS avec la même largeur du faisceau $\beta = 4$ est capable de produire les meilleures solutions avec un temps moyen de résolution inférieur à 4 secondes
3. HESGA^b et PR restent compétitifs, mais ils améliorent un nombre limité de solutions et consomment plus de temps.
4. Cplex nécessite un temps de résolution très important.
5. On note que CGBS produit 10 nouvelles solutions sur les 44 traitées (voir la colonne 2 de la table 4.6)

				PR		*GBS*		*CGBS*	
#Inst	Opt/Best	Cplex	HESGA^b	Sol.	CPU	Sol.	CPU	Sol.	CPU
Horizontale									
nice25*	860829	∘	∘	∘	0.85	∘	0.05	∘	0.15
nice50	913738	∘	912395	904955	2.4	∘	0.01	∘	0.31
nice100	911122	906476	900923	908734	11.45	∘	0.05	∘	0.95
nice200	932686	924576	926287	∘	33.81	931143	0.41	∘	0.98
nice500	952015	942548	945991	∘	289.3	951279	1.86	∘	0.89
path25*	892765	∘	∘	∘	2.1	∘	0.06	∘	0.16
path50*	750215	∘	749627	748655	2.65	∘	0.05	∘	0.55
path100	888578	∘	887967	875592	9.3	∘	0.08	∘	0.08
path200	889174	∘	871602	835895	30.33	∘	1.59	∘	1.08
path500	885112	880314	881801	879465	172.7	∘	12.87	∘	4.98
nice25P*	441993	∘	441207	422002	1.14	∘	0.02	∘	0.02
nice50P	*453460*	446839	444049	434199	2.99	449292	0.02	∘	0.02
nice100P	*459214*	452069	452069	452904	12.92	456306	0.05	∘	0.03
nice200P	*456121*	450179	450179	448480	48.41	455727	0.16	∘	0.12
nice500P	455036	454088	454088	450802	183.84	∘	1.44	∘	1.04
nice1000P	*454089*	450931	451545	451605	601.98	451805	2.94	∘	2.32
path25P*	464721	∘	460355	463693	0.67	∘	0.01	∘	0.01
path50P	*391931*	391626	389812	387795	2.59	390741	0.01	∘	0.01
path100P	456300	∘	∘	445939	14.75	∘	0.09	∘	0.09
path200P	*453579*	449779	449779	426672	38.47	452402	1.34	∘	1.04
path500P	434478	412731	430243	429127	244.86	∘	40.52	∘	20.02
path1000P	434966	434172	∘	∘	601.53	∘	150.21	∘	50.21
Verticale									
nice25*	834157	∘	∘	820363	1.11	∘	0.01	∘	0.01
nice50	*868781*	852087	854425	842696	2.11	854425	1.12	∘	0.67
nice100	911958	890571	905085	∘	9.41	∘	3.41	∘	1.09
nice200	935290	919515	∘	∘	44.71	∘	4.55	∘	3.89
nice500	960767	935255	∘	∘	316.25	∘	10.54	∘	7.98
path25*	699707	∘	694142	∘	0.75	∘	0.01	∘	0.01
path50*	924908	∘	∘	915745	1.75	∘	0.41	∘	0.34
path100*	754753	∘	743339	665339	3.45	∘	3.43	∘	2.98
path200	*837069*	835807	834350	803521	25.65	836022	3.56	∘	2.99
path500	808376	784366	803845	∘	285.41	798544	4.54	∘	3.98
nice25P*	428662	428662	419659	408409	1.31	∘	0.13	∘	0.21
nice50P	*426269*	425921	412766	408518	3.25	425921	0.55	∘	0.43
nice100P	451851	449180	449180	451051	11.13	∘	0.43	∘	4.33
nice200P	454754	453964	453964	451337	45.03	∘	1.23	∘	0.99
nice500P	457381	∘	448071	447519	211.58	448496	10.41	∘	5.98
nice1000P	459142	∘	433159	453665	602.66	457439	3.44	∘	2.78
path25P*	356088	∘	∘	354726	0.81	∘	0.05	∘	0.04
path50P*	487447	∘	∘	482273	2.28	∘	0.11	∘	0.01
path100P*	386033	∘	374149	356168	4.81	∘	1.23	∘	0.98
path200P	*418390*	416656	409856	380095	28.77	418024	2.34	∘	1.98
path500P	381860	368284	365202	∘	263.8	367936	5.2	∘	3.76
path1000P	425768	405251	407097	∘	601.77	407097	77.54	∘	26.98

TABLE 4.6 – Résultats numériques de Cplex, HESGA^b, PR, GBS et CGBS sur les instances larges. Le symbole ∘ signifie que l'algorithme produit la meilleure solution.

				GBS		*CGBS*	
#Inst	Opt/Best	Cplex	HESGA^b	Sol.	CPU	Sol.	CPU
Horizontale							
LU1	*887646*	876682	887099	887393	138.56	∘	114.00
LU2	*906712*	900923	906237	906237	310.13	∘	245.24
LU3	*1021124*	1009031	1017575	1017575	347.89	∘	279.58
LW1	*689271*	687786	688840	688840	410.09	∘	322.21
LW2	782566	778028	∘	∘	454.76	∘	314.74
LW3	899890	897373	∘	∘	488.76	∘	412.76
Verticale							
LU1	891516	888762	∘	∘	120.87	∘	100.01
LU2	908982	906600	∘	∘	210.41	∘	115.54
LU3	1022483	1018888	1018888	∘	320.87	∘	210.09
LW1	697164	∘	∘	694657	340.89	∘	301.12
LW2	783836	782149	782149	∘	410.78	∘	385.90
LW3	*890696*	881931	881931	881931	456.81	∘	352.24
Av. Dev.		0.9945	0.9980	0.9985		1	
Av. T					334.24		262.79

TABLE 4.7 – Étude comparative entre Cplex, HESGA^b, GBA et CGBA sur les instances extra-larges. Le symbole ∘ signifie que l'algorithme produit la meilleure solution.

Nous avons mentionné que la qualité des solutions obtenues par CGBS dépendent aussi de la fréquence de calcul des bornes supérieures pendant le développement γ. Dans ce sens, la table 4.5 rapporte la qualité des solutions produites lorsqu'on varie γ de 0 à 3. Cette table nous montre que pour ces instances, nous n'avons pas besoin d'augmenter la valeur de γ. Par conséquent, assigner la valeur 1 à γ est un bon compromis entre la qualité des résultats et le temps de résolution.

4.4.5 Troisième groupe d'instances

Le troisième groupe d'instances est composé de six instances (UL1, UL2, UL3, WL1, WL2, WL3) extraites de Hifi et M'Hallah [39]. Ces instances sont classées extra-larges car la hauteur W du rectangle initial dépasse 1000 unités et le nombre de pièces est supérieur à 1200. Les instances (UL1,UL2, UL3) sont des instances pondérées alors que les instances (WL1,WL2,WL3) sont des instances non-pondérées. Afin de traiter ces instances, nous avons considéré une largeur $\beta = 2$ et une fréquence $\gamma = 1$. La table 4.7 affiche les résultats obtenues par Cplex, HESGA^b, GBS et CGBS.

Les résultats montrent que GBS améliore la solution de UL1-horizontal (comparé à HESGA^b), produit la même solution pour WL2-vertical et échoue dans l'amélioration de la solution de l'instance WL1-vertical. Par ailleurs, CGBS produit la meilleure solution de 5 instances sur les 12 instances traitées (voir la colonne 2 de la table 4.7) avec un temps moyen de résolution de 262.79 secondes (voir la dernière ligne de la table 4.7)

4.5 Conclusion

Dans ce chapitre, nous avons proposé une méthode coopérative pour le problème de découpe contraint à deux dimensions à deux niveaux. Cette méthode est basée sur : une stratégie de recherche, une procédure complémentaire et une borne supérieure.
La stratégie de recherche utilise une recherche en faisceaux qui implémente une fonction d'évaluation. La procédure complémentaire est basée sur une génération de bandes qui améliore la qualité des solutions obtenues. Enfin, la borne supérieure est utilisée pour guider la recherche grâce à

l'intervention dans la fonction d'évaluation et la réduction de l'espace de recherche. Les performances de cette méthode ont été évaluées sur un ensemble d'instances et montre l'efficacité de cette méthode par rapport à d'autres méthodes de la littérature.

Chapitre 5

Recherche par faisceaux pour le problème de chargement/remplissage [5]

Contenu

Encouragé par les résultats des chapitres précédents, nous nous sommes intéressé au problème de chargement/remplissage. Ce problème a beaucoup d'applications dans la gestion d'entrepôts et le stockage en générale. Après une présentation du problème et quelques domaines d'application, nous proposons une adaptation de la recherche par faisceaux pour ce problème. Nous présentons ensuite, l'algorithme de résolution approchée que nous proposons, avant de finir par la présentation des résultats de l'algorithme sur un ensemble d'instances de la littérature.

5.1 La gestion des entrepôts

La gestion des stocks d'un entrepôt dit "ordinaire" est une tâche difficile, lorsqu'il s'agit d'ordonner et/ou d'assembler des milliers d'objets. La plupart des entreprises gèrent leurs stocks par secteur de produits, sans porter attention à l'organisation interne de ces derniers. Cependant, le conditionnement automatisé et optimisé des stocks permettrait un gain de place et d'argent pour

5. *Le contenu de ce chapitre a été présenté lors du 7ème congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision, Roadef 2008 [51]*

ces entreprises, qui ont toujours plus de produits à stocker. La gestion des stocks dans le domaine de la grande distribution est un problème fréquent et grandissant. Leur surface de vente moyenne augmente chaque année et implique un stockage de la marchandise de plus en plus important et contraignant.

D'une façon générale, les problèmes de stockages dans des entreprises sont aussi cruciaux dans le domaine de la production. Le stockage des matières premières et des produits crées est obligatoire, si l'entreprise veut satisfaire au mieux sa clientèle.

D'autres fonctionnalités peuvent apparaître dans la gestion des stocks, comme la notion de comptabilités des produits. Par exemple, des produits qui nécessitent des températures différentes, des produits fragiles, objets lourds, etc. Le rangement/placement/remplissage de la marchandise devrait aussi prendre en compte certaines contraintes dues à l'expédition des produits. Une autre contrainte qui peut apparaître lors du placement réside dans l'ordre de placement, par exemple, imposer un ordre sur le premier objet à placer avant d'autres objets. Pour ce dernier, la contrainte d'accessibilité aux objets prioritaires surgit et il faudrait donc la prendre en compte d'une façon dynamique. La figure ci-dessous décrit le processus que l'on doit respecter en partant du premier jour de la réception des objets au dernier jour représentant l'expédition.

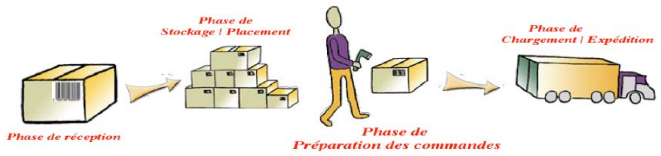


FIGURE 5.1 – Illustration de la gestion et du suivi des produits (objets).

Notons que d'autres contraintes peuvent se greffer lorsque la notion des entrepôts mutualisés est évoquée. En effet, une première contrainte est directement liée la topologie des entrepôts. Cette dernière nécessite une schématisation très précise de la structure de chacun des entrepôts, de leur capacité ainsi que des informations sur les produits pouvant être supportés par chacun des entrepôts. De telles contraintes nous incitent à élaborer des stratégies optimales permettant de répondre à l'attente de chacune des entreprises.

5.2 Chargement / remplissage

Le problème de chargement est connu sous le nom du " Bin-Packing ou Pallet-Loading " (Voir Dyckhoff [17], Sweeney et Paternoster [65]). Il s'agit de l'une des versions la plus utilisée en industrie. Dans des versions plus complexes, le processus séquentiel suivant doit être respecté : (i)

le remplissage d'objets dans des conteneurs, (ii) le chargement des conteneurs dans des camions et finalement (ii) la tournée engagée et le gestion du retour. Chacun des deux premiers points (i) et (ii) du processus est lié à certaines contraintes réelles qui peuvent facilement perturber la faisabilité du processus. Parmi les contraintes les plus rencontrées dans la pratique nous citons les contraintes de capacité, les contraintes de comptabilité ainsi que les contraintes de précédences.

Les contraintes de capacité : en général, chaque camion ou conteneur possède une capacité bien spécifique. Il est donc prudent d'effectuer une affectation réalisable permettant de respecter le tonnage des camions ainsi que la capacité maximale associée aux conteneurs.

Les contraintes de comptabilités : il s'agit de la gestion des conflits entre objets. Il est souvent nécessaire de séparer certains objets afin de garder la qualité des autres objets Un premier exemple rencontré dans la distribution apparaît lorsque l'on transporte des produits de jardinage ; à cause de certains gerbages, certains produits ne peuvent être transportés avec d'autres.

Un deuxième exemple, qui est bien connu dans la distribution, est lié aux objets fragiles. Dans certains cas, il est recommandé de placer des objets fragiles au-dessus des objets moins fragiles. Finalement, un troisième exemple liée aux objets de groupage : un ensemble d'objets ne peuvent être transportés séparément (une armoire est composée de plusieurs objets qui forment un groupe ; ce dernier groupe doit être chargé dans le même conteneur et donc le même camion).

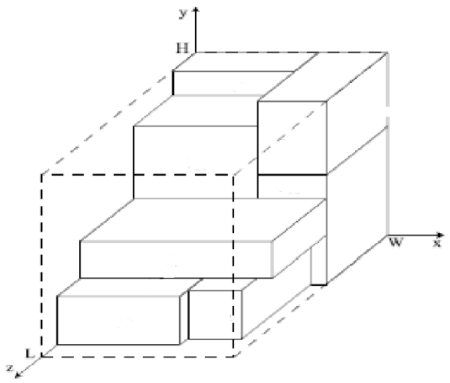


FIGURE 5.2 – Représentation d'un placement en trois dimensions (chargement ou remplissage maximum).

Les contraintes de précédences : Ces contraintes sont très spécifiques et elles apparaissent principalement lors du déchargement ou chargement adaptatif.
En effet, un véhicule peut décharger à un point A un nombre de conteneurs, il est donc préférable (et même recommandé) d'affecter un premier ordre de placement dans un véhicule pour une bonne organisation. Un véhicule peut décharger et charger d'autres conteneurs à un autre point B. Dans ce cas, il est recommandé d'affecter un deuxième ordre liant les conteneurs existants ainsi que les conteneurs positionnés au point B. Pour ce dernier cas, on parlera d'un chargement adaptatif.

La figure 5.2 illustre un problème de chargement (ou de remplissage) en trois dimensions ; il s'agit d'un problème sous sa forme standard.

5.3 L'étude du problème de chargement/remplissage

L'optimisation efficace des capacités (ou volumes) obtenue est considérée comme un point très important dans les problèmes de planification, de transport et de la production. Une des raisons principales est due à la croissance de la demande du marché, qui induit une augmentation forte du transport.

Dans de nombreux cas le chargement efficace de véhicules (voitures ou camions) peut être modélisé comme un problème de placement en trois dimensions, ce qu'on appelle aussi le problème de chargement d'un container (voir Bischoff et Marriott [9]). Dans d'autres situations (par exemple le cas de la production), le problème est catégorisée comme un problème de découpe (ou assemblage −packing−) des pièces dans une plus grande palette. Cette dernière peut être modélisée sous forme d'un problème de découpe/packing en trois dimensions. L'un des premiers exemple caractérisant cette problématique a été traitée par Gilmore et Gomory [62] (découpe de blocs de graphite).

Nous avons étudié le problème de placement/chargement en trois dimensions. Ce problème consiste à déterminer un plan de chargement de n objets de dimensions $(l_i, w_i, h_i),\ i = 1, \ldots, n$, dans un container (ou un récipient) de dimensions (L, W, H), où L (resp. l_i) dénote la longueur, W (resp. w_i) la largeur et H (resp. h_i) la profondeur. Évidemment, le problème peut être aussi ramener vers la gestion d'un entrepôt si l'on cherche à optimiser l'emplacement.

De plus, dans notre étude nous nous intéressons à maximiser le remplissage (placement) où chaque objet $i,\ i = 1, \ldots, n$, est caractérisé par son profit c_i (qui peut représenter le volume ou bien un volume pondéré). Dans la suite, on notera le problème traité par 3DBP, une référence pour le problème "Three-Dimensional Bin Packing".

Souvent en programmation mathématique la résolution d'un problème P peut se ramener à la résolution d'une série de m problèmes $P_1, \ldots, P_m$ plus ou moins facile à résoudre, comparés au problème original P.

Hifi dans [35], a suivi cette démarche. En effet, un plan de chargement, pour le problème 3DBP, peut être construit par la résolution d'une série de problèmes de placement en deux dimensions et par combinaison de ces deniers solutions pour produire une solution admissible (et parfois optimale). C'est une méthode de décomposition que nous détaillons dans les sections suivantes. Au fait cette approche, basée sur une série de modèles mathématiques, est principalement composée de quatre phases.

1. ***Phase 1 : Construction des z-piles***

   Comme il s'agit d'un container en trois dimensions, nous considérons donc que l'étude s'effectue sur un espace en trois dimensions, que l'on notera (x, y, z) −que l'on notera aussi $(\alpha, \beta, \gamma) \leq (L, W, H)$−. A partir de cet espace, on peut définir une z-pile (x-pile et y-pile, respectivement) comme étant la plaque rectangulaire de dimension (L, W) ((L, H) et (W, H), respectivement) et de profondeur $z \in \{h_1, \ldots, h_n\}$ (largeur $y \in \{w_1, \ldots, w_n\}$ et longueur $x \in \{l_1, \ldots, l_n\}$, respectivement).

   Soit S l'ensemble des objets à placer et $p \leq n$ le nombre de profondeurs distinctes supposées ordonnées comme suit : $h_1 < h_2 < \ldots < h_p$. Soit w_k une largeur du sous-objet (α, β, h_j) pour une profondeur donnée $h_j,\ j = 1, \ldots, p$. Considérons les r différentes largeurs $w_k \in \{w_1, \ldots, w_r\}$, satisfaisant l'ordre suivant : $w_1 < w_2 < \ldots < w_r$.

Notons $S_{(\alpha,w_k,h_j)} \subseteq S$ l'ensemble des sous-objets tel que $S_{(\alpha,w_k,h_j)} := \Big\{ i \in S \mid l_i \leq \alpha,\ w_i \leq w_k \leq \beta,\ h_i \leq h_j \Big\}$. Pour chaque couple $(w_k, h_j) \leq (\alpha, \gamma) \leq (L, H)$ est associé un problème de knapsack $(K^{w_k}_{\alpha,h_j})$ donné par :

$$(K^{w_k}_{\alpha,h_j}) \begin{cases} f^{z-pile}_{(\alpha,w_k,h_j)} = \max & \displaystyle\sum_{i \in S_{(\alpha,w_k,h_j)}} c_i x_i \\ \text{s.c.} & \displaystyle\sum_{i \in S_{(\alpha,w_k,h_j)}} l_i x_i \leq \alpha \\ & x_i \in \mathbb{N},\ i \in S_{(\alpha,w_k,h_j)}, \end{cases}$$

où x_i est le nombre d'occurrences de l'objet i dans la z-pile de dimensions (α, w_k, h_j), c_i dénote le profit associé à l'élément $i \in S_{(\alpha,w_k,h_j)}$ et $f^{z-pile}_{(\alpha,w_k,h_j)}$ est la valeur économique de la solution de la z-pile de dimensions (α, w_k, h_j), pour $k = 1, \ldots, r$. Ce programme mathématique peut être résolu par deux types d'approches. La première approche consiste à appliquer une procédure de séparation et d'évaluation et la deuxième approche utilise la programmation dynamique (voir Kellerer *et al.* [18]). Dans notre étude, vu la particularité du problème ainsi que les méthodes que nous avons proposé pour le résoudre, nous avons utilisé la deuxième approche (programmation dynamique).

2. ***Phase 2 : Combinaison et utilisation des z-piles***

   Cette phase permet, pour une profondeur fixe h_j, d'optimiser le sous-objet (α, β, h_j), pour $j = 1, \ldots, p$. Pour chacun de ces sous-objets, nous résolvons un autre programme mathématique $(K^{z-pile}_{(\alpha,\beta,h_j)})$ donné par :

$$(K^{z-pile}_{(\alpha,\beta,h_j)}) \begin{cases} \xi^{z-pile}_{(\alpha,\beta,h_j)} = \max & \displaystyle\sum_{k=1}^{r} f^{z-pile}_{(\alpha,w_k,h_j)} y_k \\ \text{s.c.} & \displaystyle\sum_{k=1}^{r} w_k y_k \leq \beta,\ y_k \in \mathbb{N}, \end{cases}$$

   où $y_k,\ k = 1, \ldots, r$, est le nombre d'occurrences de la z-pile de dimensions (α, w_k, h_j) dans (α, β, h_j) et $\xi^{z-pile}_{(\alpha,\beta,h_j)}$ est la solution produite pour $(\alpha, \beta, h_j),\ j = 1, \ldots, p$.

3. ***Phase 3 : Construction d'un plan de chargement pour le sous-objet*** (α, β, γ)

   Cette partie constitue la phase de construction qui permet de produire une solution par utilisation des z-piles. Le principe de la méthode utilisée consiste à explorer deux chemins au plus, sur plusieurs chemins possibles. A chaque étape du processus de construction, l'algorithme considère un sous-ensemble de z-piles. Le choix de ces z-piles est réalisé par application d'un mécanisme d'évaluation globale.

   Dans notre cas, une première étape, par application de la programmation dynamique, nous conduit vers un plan de chargement admissible. Ce plan peut être obtenu par la résolution du modèle mathématique suivant :

$$(K^{z-pile}_{(\alpha,\beta,\gamma)}) \begin{cases} \Phi^{z-pile}_{(\alpha,\beta,\gamma)} = \max & \displaystyle\sum_{j=1}^{p} \xi^{z-pile}_{(\alpha,\beta,h_j)} z_j \\ \text{s.c.} & \displaystyle\sum_{j=1}^{p} h_j z_j \leq \gamma,\ z_j \in \mathbb{N}, \end{cases}$$

où $z_j,\ j = 1, \ldots, p$, est le nombre d'occurrences de la z-pile de dimensions (α, β, h_j) dans (α, β, γ) et $\Phi^{z-pile}_{(\alpha,\beta,\gamma)}$ est la solution produite pour (α, β, γ). Notons que dans notre cas, il suffit de poser $(\alpha, \beta, \gamma) = (L, W, H)$ et par application de la programmation dynamique, nous obtenons un plan de chargement pour le problème 3DBP [1].

5.4 Recherche par faisceaux appliquée au problème 3DBP

L'adaptation de la recherche par faisceaux pour le problème 3DBP nécessite la définition des nœuds de l'arborescence et du mécanisme de branchement. Dans ce contexte, nous proposons de décomposer le problème 3DBP en une série de problèmes de 2DBP (Bin Packing en deux Dimensions). Au fait, chaque problème 2DBP est associé à une profondeur fixée $\gamma,\ \gamma \in \{h_1, \ldots, h_p\}$, où $p \leq n$ dénote un nombre distinct de profondeurs de l'instance du problème 3DBP.

Par conséquent, pour une profondeur fixée γ, un nœud est représenté par une paire de sous container $(L, W - y, \gamma)$ et (L, y, γ), avec $y \leq W$ et $(L, W - y, \gamma)$ la solution admissible courante. $(L, W - y, \gamma)$ est la solution construite par combinaison des piles et (L, y, γ) le reste du volume du container initial. Dans ce cas, la racine de l'arborescence peut être représentée par les deux sous containers $(L, 0, \gamma)$ et (L, W, γ). Dans cette configuration aucune pile n'est encore placée dans le container initial.

Soit r, le nombre de largeurs distincts des objets de l'instance du 3DBP et p le plus grand indice de l'objet de largeur w_p, tel que $w_1 < w_2 < \ldots < w_p$. Alors, le branchement sur le nœud $u = ((L, W - y, \gamma), (L, y, \gamma)), y \leq W$ est équivalent à la séparation de la pile $(L, \beta, \gamma),\ \beta \leq y$, $\beta \in \{\overline{w}_1, \overline{w}_2, \ldots, \overline{w}_r\}$ du sous container (L, y, γ). On fait remarquer qu'il y aura au maximum r branchement sur le nœud u, où chaque branchement correspond à la séparation d'une pile $(L, \overline{w}_j, \gamma)$, $j \in J$. Chacun de ces nœuds correspond à deux sous containers $(L, W-y+\overline{w}_j, \gamma)$ et $(L, y-\overline{w}_j, \gamma), j \in J$.

Le processus de recherche peut être décrit comme suite :

- Sur le nœud racine, aucune pile n'est partagée. L'algorithme créer r piles générales différentes $(L, \overline{w}_j, \gamma), j \in J$ par la résolution de $KP^{\overline{w}_j}_{(L,\gamma)}$ (décrit auparavant dans la Section 1). Chaque pile $j \in J$ correspond à un nœud fils, à développer.
- Pour effectuer un branchement sur le nœud $\big((L, W - y, \gamma), (L, y, \gamma)\big)$ pendant le parcours, l'algorithme effectue les opérations suivantes :
  1. pour $j = 1, \ldots, r$, résoudre le programme mathématique, par la programmation dynamique, $KP^{\gamma}_{(L,\overline{w}_j)}$ (décrit auparavant dans la Section 1).
  2. pour chaque pile $(L, \overline{w}_j, \gamma),\ j = 1, \ldots, r$ et $\overline{w}_j \leq y$:
     (a) séparer la pile du sous container (L, y, γ);
     (b) créer les nouveaux nœuds $((L, W - y + \overline{w}_j, \gamma), (L, Y - \overline{w}_j, \gamma)),\ i = 1, \ldots, r$;
     (c) calculer une évaluation $\hat{Z}$ du nouveau nœud en résolvant le programme mathématique $(K^{z-pile}_{(L,\beta,\gamma)})$, décrit auparavant dans la Section 2, où β dénote les séparations des nouveaux nœuds crées.

1. *Cette approche, avec sa partie expérimentale sur des instances de la littérature, a été présentée lors du 7ème congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision, Roadef 2008 [51]*

A l'étape 2c, l'algorithme fait une évaluation des nœuds en utilisant une fonction d'évaluation bien définie. Notons aussi que la procédure précédemment décrite permet la construction d'un plan de chargement/remplissage pour un sous container de dimensions (L, W, γ), où $\gamma \in \{\bar{w}_1, \ldots, \bar{w}_p\}$. Par la suite, nous verrons comment à partir de ces sous containers, la résolution d'un programme mathématique induit un plan admissible pour le problème 3DBP.

Dans la suite, nous avons considéré deux types de fonctions : une première fonction d'évaluation locale, que l'on notera stratégie LBS (*Local Beam Search*), et une deuxième fonction d'évaluation globale, que l'on notera stratégie GBS (*Global Beam Search*).

5.4.1 Recherche par faisceaux − utilisation de la stratégie LBS

La stratégie LBS effectue un choix des $min\Big\{\beta, |M|\Big\}$ nœuds dont l'évaluation des solutions réalisables sont plus importantes. Ce choix correspond à une sélection sur les nœuds d'une liste M composée de nœuds caractérisants des solutions partielles pour le problème 3DBP. Afin d'évaluer le degré du choix du nœud $u = \big((L, W-y, \gamma), (L, y, \gamma)\big), y \leq W$ de M, LBS utilise un opérateur local qui s'appuie sur une évaluation local de fonction partielle $\hat{Z}_u^{Local}$, où $\hat{Z}_u^{Local}$ représente la somme des profits des objets constituant la solution partielle du sous container $(L, W-y, \gamma)$. $\hat{Z}_u^{Local} = \sum_{i \in I} c_i v_i$, où v_i est le nombre d'apparitions l'objet de type i dans $(L, W-y, \gamma)$.

Avant de décrire le pseudo code de l'algorithme utilisant la stratégie LBS pour le problème 3DBP, nous rappelons, dans un premier temps, le programme mathématique permettant de générer la solution admissible finale pour le 3DBP. Rappelons qu'à partir du container (L, W, H), nous pouvons avoir la partition suivante : $(L, W, h_1), \ldots, (L, W, h_p)$, où $p \leq n$ dénote le nombre des différentes profondeurs associées aux différents objets de l'instance du 3DBP. Maintenant supposons l'existence d'un algorithme permettant de produire la solution admissible pour le sous container (L, W, h_j) dont l'évaluation est $\xi^{z-pile}_{(L,W,h_j)}$. Alors, résoudre le programme mathématique ci-dessous permet de produire un plan admissible pour le problème 3DBP :

$$(K^{z-pile}_{(L,W,H)}) \begin{cases} \Phi^{z-pile}_{(L,W,H)} = \max & \sum_{j=1}^{p} \xi^{z-pile}_{(L,W,h_j)} z_j \\ \text{s.c.} & \sum_{j=1}^{p} h_j z_j \leq H,\ z_j \in \mathbb{N}, \end{cases}$$

où $z_j,\ j = 1, \ldots, p$, est le nombre d'occurrences de la z-pile de dimensions (L, W, h_j) dans le sous container (L, W, H) et $\Phi^{z-pile}_{(L,W,H)}$ est la solution produite pour le container initial (L, W, H).

Nous rappelons que nous avons supposé l'existence d'un algorithme permettant de renvoyer, à tout moment, une solution admissible pour un sous container de dimensions $(L, W, z),\ 0 \leq z \leq H$. En effet, dans ce qui suit, nous donnons (la figure 5.3) un pseudo code le l'algorithme utilisant la stratégie LBS.

5.4.2 Recherche par faisceaux − utilisation de la stratégie GBS

Cette fois, la stratégie GBS s'intéresse plutôt aux nœuds *élites*. Au lieu de considérer que l'évaluation partielle, cette stratégie étend le choix vers une estimation globale, qui semble plus naturelle et qui peut aussi contenir plus d'informations lors de la sélection d'un nœud. En effet, les nœuds élites sont considérés, dans notre cas, comme ceux dont l'évaluation globale −basée sur une

`Entrée :` Une instance du problème 3DBP et une profondeur $\gamma \leq H$ fixe.
`Sortie :` Une solution approchée du 3DBP de valeur $Best_{(L,W,\gamma)}$.

Initialisation

Poser $M = \emptyset$ et $N = \{u = ((L, 0, \gamma), (L, W, \gamma))\}$.

Phase Iterative

1. Pour chaque nœud u de N :
   (a) Brancher sur le nœud u et générer les nœuds fils de u en créant r piles générales différentes $(L, \overline{w}_j, \gamma), j \in J$ par la résolution du programme mathématique $(KP^{z-pile}_{(L,\overline{w}_j,\gamma)})$. En posant $\overline{w}_j = \overline{w}_r$ et par application de la programmation dynamique, les différentes r z-piles sont générées après la résolution du plus large problème $(KP^{z-pile}_{(L,\overline{w}_r,\gamma)})$.
   (b) Pour chaque z-pile $(L, \overline{w}_j, \gamma), j \in J$, générer le nœud $u_j = ((L, W - \overline{w}_j, \gamma), (L, \overline{w}_j, \gamma))$, puis calculer $\hat{Z}^{Local}_{u_j}$ l'évaluation partielle du sous container $(L, \overline{w}_j, \gamma)$ et insérer u_j dans la liste M.
2. Insérer dans N les $\min\{\beta, |M|\}$ nœuds de M ayant la plus grande évaluation $\hat{Z}^{Local}_u$.
3. Poser $M = \emptyset$.

Condition d'arrêt

Si $N = \emptyset$, alors sortir avec la meilleure solution admissible dont l'évaluation est représentée par $Best_{(L,W,\gamma)}$; sinon, aller à l'étape 1 de la *Phase Itérative*.

FIGURE 5.3 – Un algorithme adaptatif utilisant la stratégie LBS.

fonction d'évaluation estimant la borne supérieure du 3DBP en ce nœud− est de valeur maximale. La stratégie GBS utilise une évaluation $\hat{Z}^{Global}_u$ du nœud $u = ((L, W - y, \gamma), (L, y)), y \leq W$ de M qui est le résultat d'une combinaison entre :

(i) une évaluation partielle, notée $\hat{Z}^{Local}_u$, de la solution admissible local associée au sous container $(L, W - y, \gamma)$, et

(ii) une estimation de la solution complémentaire, notée U_u, qui représente une borne supérieure du sous container complémentaire (L, y, γ).

Comme l'évaluation globale est composée d'une évaluation partielle complétée par une borne supérieure complémentaire, nous allons donc résumer les bornes supérieures utilisées ainsi que l'algorithme utilisant la stratégie GBS.

5.4.2.1 Utilisation d'une borne supérieure

L'estimation de la valeur optimale pour le sous container de dimensions $u = (L, y, \gamma)$ peut être obtenue par la résolution du programme mathématique suivant :

$$U_u = \max\Big\{ \sum_{j \in S_u} c_j t_j \;\Big|\; \sum_{j \in S_u} (l_j w_j h_j) \times t_j \leq V_u, t_j \in \mathbb{N}, j \in S_u \Big\},$$

où S_u représente l'ensemble des objets réalisants $(l_j, w_j, h_j) \leq (l_u, w_u, h_u)$, V_u est le volume du sous container u, t_j dénote le nombre d'apparitions de l'objet $j \in S_u$ dans le sous container u, et U_u est la borne supérieure de u.

La résolution du programme mathématique (en posant $y = W$) par une procédure de la programmation dynamique, permet de calculer les bornes supérieures de tous les sous containers possibles (L, y, γ), avec $y = 1, \ldots, W$. Par conséquence, le calcul de U_u se fait une seule fois sur le nœud racine. Ainsi l'utilisation de son évaluation, pendant le développement de l'arborescence, peut être facilement récupérée en un temps constant. Ce principe permet de calculer à l'avance toutes les bornes supérieures des sous containers.

5.4.2.2 Réduction de l'espace de recherche

L'utilisation des méthodes de type séparation et évaluation nécessite parfois une génération importante de sous problèmes. Dans ces conditions et parce que l'estimation de l'optimum sur le sous-container complémentaire se fait sur un problème relaxé, nous proposons donc de réduire l'espace de recherche par application de la stratégie du *hill-climbing*. En effet, cette stratégie consiste à effectuer l'élagage d'un nouveau nœud généré $u = (L, W - y, \gamma)$ de l'ensemble M si et seulement si son évaluation globale reste plus petite ou égale à la meilleure évaluation admissible courante. Dans ce cas, l'élagage utilise la formule suivante sur chaque nouveau nœud généré :

$$\hat{Z}_u^{Global} = \hat{Z}_u^{Local} + U_u \leq Z^*,$$

avec Z^* la meilleure solution admissible courante. La formule précédente interprète simplement le rejet du nœud généré u si son évaluation globale ne peut conduire vers une meilleure solution admissible que celle réalisant Z^*.

5.4.2.3 Filtrage des nœuds

Vu le nombre important de nœuds générés à chaque itération de la procédure de construction, parfois il est difficile de générer un échantillon dont les évaluations globales sont très proches. Afin d'éviter ce cas cyclique et de tenter une bonne sélection des nœuds élites, nous avons considéré le choix des nœuds en deux phases. La première phase définit, sur l'ensemble M, le sous ensemble η des nœuds à choisir par application de la formule suivante :

$$\eta = \max_{u \in M} \left\{ \hat{Z}_u^{Global} \right\}.$$

Comme la première phase peut générer un nombre important de nœuds dont les évaluations sont très proches, nous avons introduit la deuxième phase qui calcul, pour chaque nœud $u \in M$, l'écart entre $\hat{Z}_u^{Local}$ et η Par conséquent, l'ensemble des nœuds élites est celui qui réalise : $\min\{\beta, |M|\}$ et dont l'écart entre $\hat{Z}_u^{Local}$ et η est la plus importante. Dans le cas où plusieurs nœuds de M produisent les mêmes écarts, la sélection favorise le nœud dont l'évaluation partielle $\hat{Z}_u^{Local}$ est la plus importante.

5.4.2.4 Solution initiale

D'une façon générale, l'utilisation d'une solution de départ pour une méthode par séparation et évaluation, reste un point très important. En effet, plus l'évaluation de la solution admissible est proche de l'estimation globale, plus l'efficacité de l'algorithme est meilleure. De cette façon, nous pouvons toujours espérer des convergences assez rapides pour ce type de méthodes.

`Entrée` : Une instance du problème 3DBP et une profondeur $\gamma \leq H$ fixe.
`Sortie` : Une solution approchée pour le 3DBP de valeur Z^*.

Initialisation

- Poser $M = \emptyset$ et $N = \{u = ((L, 0, \gamma), (L, W, \gamma))\}$.
- Appliquer l'algorithme LBS avec un largeur du faisceau égal à un et soit Z^* l'évaluation de la solution admissible obtenue.

Phase Iterative

1. Pour chaque nœud u de N :
   (a) Brancher sur le nœud u et générer les nœuds fils de u en créant r piles générales différentes $(L, \overline{w}_j, \gamma)$, $j \in J$, par la résolution du programme mathématique $KP^{z-pile}_{(L,\overline{w}_j,\gamma)}$. Poser $\bar{w}_j = \bar{w}_r$ et appliquer la programmation dynamique sur le programme mathématique $KP^{z-pile}_{(L,\overline{w}_r,\gamma)}$.
   (b) Pour chaque pile $(L, \overline{w}_j, \gamma), j \in J$, générer le nœud $u_j = \big((L, W - \overline{w}_j, \gamma), (L, \overline{w}_j, \gamma)\big)$, calculer $\hat{Z}^{Global}_{u_j} = \hat{Z}^{Local}_{u_j} + U_{u_j}$ et insérer u_j dans la liste M.
2. Calculer $\eta = \max_{u \in M} \{\hat{Z}^{Global}_u\}$.
3. Pour chaque nœud $u \in M$, calculer l'écart entre $\hat{Z}^{Local}_u$ et η.
4. Insérer dans l'ensemble N les $\min\{\beta, |M|\}$ de M ayant le plus grand écart.
5. Poser $M = \emptyset$.

Condition d'arrêt

Si N est réduit à l'ensemble vide, alors sortir avec la solution admissible dont l'évaluation est Z^*; sinon, aller à l'étape 1 de la *Phase Itérative*;

FIGURE 5.4 – Un algorithme adaptatif par application de la stratégie GBS.

Dans le cas de la méthode utilisant la stratégie GBS, un démarrage par une solution de départ reste important. D'une part, elle permet d'élaguer des sous solutions qui peuvent facilement conduire vers des solutions cycliques, et d'autre part, elle permet de guider la recherche vers les solutions admissibles dont l'évaluation partielle est la plus importante. Cette dernière permet donc de conduire vers une amélioration plus rapide de la solution de départ. Notons que dans notre cas, la méthode utilisant la stratégie GBS démarre avec la solution obtenue par la méthode LBS (section 5.4.1) avec un largeur réduit à un ; ce qui permet de simuler un algorithme glouton par résolution d'une série de programmes mathématiques plus ou moins faciles à résoudre.

La figure 5.4 résume le pseudo code de la méthode utilisant la stratégie GBS. Comme le montre cette figure, nous supposons que la donnée du problème est un sous container de dimen-

sions (L, W, γ) pour lequel il est appliqué l'algorithme de la figure 3.3. Par la suite, après application de la méthode sur chacune des profondeurs $\gamma \in \{h_1, \ldots, h_p\}$, nous aurons comme résultat une séries de couples (profondeur, évaluation). Avec ces derniers couples, nous construisons un programme mathématique que nous résolvons, comme nous l'avons déjà signalé dans la section 5.4.1, pour fournir un plan de remplissage admissible pour le problème 3DBP.

5.5 Résultats numériques

5.5.1 Les instances

Notre étude expérimentale a été réalisée sur un ensemble de 144 instances de la littérature (voir Hifi [35] et Morabito et Arenales [71]). Nous avons décomposé ces instances en plusieurs groupes selon leurs types (pondérées ou non pondérées).

Le premiers groupe (voir table 5.1) est un ensemble de 64 instances générées aléatoirement. Les 32 premières instances sont non pondérées et les 32 instances qui restent sont pondérées. Les instance non pondérées ont été généré par Hifi dans [35] comme suite :

- L (respectivement W et H) ont été généré dans l'intervalle $[25, 500]$.
- Le nombre de pièces à placer a été généré dans l'intervalle $[5, 50]$.
- La taille des pièces à été généré dans l'intervalle $[0.1\phi, 0.75\phi]$, où $\phi = L,\ W$ et H.

Les instances pondérées sont générées à partir des instances précédentes avec un profit c_i égale à $\lceil \phi l_i w_i h_i \rceil$. ϕ est généré aléatoirement dans l'intervalle $[0.1\% \times \pi_i, 4\% \times \pi_i]$ et $\pi_i = l_i w_i h_i$, pour $i = 1, \ldots, n$.

Le reste des instances (voir table 5.2) ont été obtenues de Morabito et Arenales [71]. Ces instances sont non pondérées.

5.5.2 L'objectif

L'objectif principal de l'étude expérimentale que nous avons mené été de voir la compatibilité de la recherche par faisceaux sur ce type de problèmes. Ainsi nous avons commencé par traiter les instances du premier groupe (les solutions optimales de ces instances sont connues) avec différentes valeurs de η. La résolution de ces instances nous a permit de mesurer les qualités des solutions produites par LGBS et GBS avec $\eta = 1$ et $\eta = 2$ et de mesurer le temps de la résolution.

Nous avons ensuite, traité le deuxième groupe d'instance (l'optimum de ces instances n'est pas connu) avec la meilleure largeur η déterminée sur le premier groupe. Nous avons, ensuite comparé les solutions obtenues aux solutions produites par l'algorithme A2-B (Hifi [35]).

5.5.3 Les résultats

5.5.3.1 Le premier groupe d'instances

La table 5.1 montre les résultats obtenus sur le premier groupe d'instances. La colonne 1 identifie l'instance traitée et la colonne 2 rapporte la solution optimale de chaque instance. Les colonnes 3 à 8 montrent les résultats de LBS. Pour chaque valeur de η ($\eta = 1$ et $\eta = 2$), la table 5.1

montre la solution obtenue (notée Sol.), la valeur de la déviation par rapport à l'optimum (Gap) et le temps de résolution en secondes (noté cpu). De la même manière, les colonnes 9 à 14, montrent les résultats de GBS avec $\eta = 1$ et $\eta = 2$. Pour chaque version, la table rapporte la valeur de la solution réalisable finale, la déviation par rapport à l'optimum et le temps de résolutions en secondes.
Les trois dernière lignes de la table 5.1 montrent, respectivement, la moyenne du temps de résolution (noté Av_{cpu}), la déviation moyenne par rapport à l'optimum (noté Av_{Gap}) et le nombre d'instances résolues à l'optimum (noté NB_{Opt}).

Les résultats obtenus sur le premier groupe d'instances et présentés dans la table 5.1 montrent que :

- LBS_1 est la version la plus rapide avec un temps de résolution moyen de 1.18. LBS_1 résout 13 instances à l'optimum et produit une déviation moyenne de -46524.5.
- GBS_1 est comparable à LBS_1. Il résout 13 instances à l'optimum dans un temps plus important (2.95 secondes). GBS_1 génère la même déviation moyenne que LBS_1 (-46524.5).
- LBS_2 est plus performant que LBS_1 et GBS_1, il arrive à résoudre à l'optimum 46 instances parmi les 64 traitées avec une déviation moyenne de -2393.125. Le temps de résolution de LBS_2 est plus grand que LBS_1 mais il est inférieur au temps consommé par GBS_1 (2.35 secondes).
- GBS_2 arrive à résoudre toutes les instances à l'optimum (64/64) dans un temps moyen de 11.20 secondes.

Ces résultats montrent que GBS_2 est performant pour ce groupe d'instances mais l'application de la stratégie globale avec une largeur de faisceau réduite, augmente les temps de résolution (calcul de la borne supérieure) sans améliorer forcement la qualité des solutions produites.

5.5.3.2 Le deuxième groupe d'instances

Les tables 5.2 et 5.3 montrent les résultats obtenus par LBS sur le deuxième groupe d'instances. La colonne 1 identifie l'instance traitée et la colonne 2 rapporte la solution optimale ou la meilleure solution de la littérature de chaque instance (les instances dont la solution optimale est connue sont signalées avec le symbole $*$). Les colonnes 3 et 4 montrent, respectivement, les résultats des algorithmes A1 et A2 (voir Hifi [35]). Pour chaque valeur de η ($\eta = 1$ et $\eta = 2$), la table 5.2 montre la solution obtenue (notée Sol.), la valeur de la déviation par rapport à l'optimum (Gap) et le temps de résolution en secondes (noté cpu). De la même manière, les colonnes 5 à 10 de la table 5.3, montrent les résultats de LBS avec $\eta = 3$ et $\eta = 4$. Pour chaque version, la table rapporte la valeur de la solution réalisable finale, la déviation par rapport à l'optimum et le temps de résolutions en secondes.
Les trois dernière lignes de chaque table montrent, respectivement, la moyenne du temps de résolution (noté Av_{cpu}), la déviation moyenne par rapport à l'optimum (noté Av_{Gap}) et le nombre d'instances résolues à l'optimum (noté NB_{Opt}).

Les tables 5.4 et 5.5 rapportent les résultats obtenus par GBS sur le deuxième groupe d'instances. La colonne 1 identifie l'instance traitée et la colonne 2 rapporte la solution optimale ou la meilleure solution de la littérature de chaque instance. Les colonnes 3 et 4 montrent, respec-

#Inst.	Opt	LBS $\eta=1$ Sol.	LBS $\eta=1$ Gap	LBS $\eta=1$ cpu	LBS $\eta=2$ Sol.	LBS $\eta=2$ Gap	LBS $\eta=2$ cpu	GBS $\eta=1$ Sol.	GBS $\eta=1$ Gap	GBS $\eta=1$ cpu	GBS $\eta=2$ Sol.	GBS $\eta=2$ Gap	GBS $\eta=2$ cpu
UUS-CL1	36975	36975	0	1.90	36975	0	3.11	36975	0	4.39	36975	0	12.21
UUS-CL2	102681	102207	-474	1.22	102681	0	2.86	102207	-474	2.17	102681	0	7.37
UUS-CL3	119608	115178	-4430	1.59	119608	0	2.65	115178	-4430	2.52	119608	0	7.56
UUS-CL4	445396	434848	-10548	1.65	444560	-836	2.61	434848	-10548	1.93	445396	0	12.41
UUS-CL5	1150170	1146510	-3660	1.27	1146510	-3660	2.12	1146510	-3660	3.65	1150170	0	12.16
UUS-CL6	939456	932160	-7296	0.87	939456	0	2.43	932160	-7296	1.03	939456	0	11.91
UUS-CL7	1455406	1434832	-20574	1.09	1452808	-2598	2.06	1434832	-20574	1.96	1455406	0	9.09
UUS-CL8	1034815	1034280	-535	0.58	1034815	0	1.86	1034280	-535	1.76	1034815	0	12.05
UUS-CL9	3181370	3131586	-49784	1.79	3167058	-14312	2.35	3131586	-49784	3.81	3181370	0	13.94
UUS-CL10	2002179	1966473	-35706	0.52	2002179	0	1.81	1966473	-35706	4.68	2002179	0	11.38
UUS-CL11	19844592	19288560	-556032	1.83	19755816	-88776	3.58	19288560	-556032	2.77	19844592	0	12.91
UUS-CL12	55391956	55391956	0	1.36	55391956	0	1.90	55391956	0	3.71	55391956	0	13.89
UUS-CL13	50565463	49916460	-649003	1.22	50565463	0	2.92	49916460	-649003	1.44	50565463	0	10.79
UUS-CL14	2085881	2072173	-13708	0.85	2085881	0	1.68	2072173	-13708	1.77	2085881	0	11.77
UUS-CL15	16705332	15954999	-750333	1.01	16705332	0	1.76	15954999	-750333	2.52	16705332	0	12.20
UUS-CL16	17856765	17516388	-340377	1.31	17856765	0	2.17	17516388	-340377	1.52	17856765	0	7.41
UUS-CL17	6139059	5954886	-184173	0.63	6139059	0	1.63	5954886	-184173	1.21	6139059	0	7.26
UUS-CL18	8678052	8645499	-32553	1.48	8678052	0	2.99	8645499	-32553	2.84	8678052	0	8.33
UUS-CL19	158714	158714	0	1.05	158714	0	2.33	158714	0	4.03	158714	0	10.59
UUS-CL20	263228	249732	-13496	1.29	263228	0	2.55	249732	-13496	1.42	263228	0	9.81
UUS-CL21	458712	443984	-14728	1.28	458712	0	2.02	443984	-14728	3.15	458712	0	9.81
UUS-CL22	729068	707511	-21557	1.56	729068	0	2.18	707511	-21557	1.03	729068	0	10.19
UUS-CL23	837573	821835	-15738	0.73	837573	0	2.02	821835	-15738	2.23	837573	0	12.51
UUS-CL24	953408	932064	-21344	1.31	953408	0	2.44	932064	-21344	2.22	953408	0	11.44
UUS-CL25	1480221	1432412	-47809	1.27	1480221	0	1.91	1432412	-47809	3.28	1480221	0	11.70
UUS-CL26	1870785	1851639	-19146	0.57	1857909	-12876	1.07	1851639	-19146	1.83	1870785	0	9.19
UUS-CL27	1728065	1691417	-36648	0.53	1728065	0	1.77	1691417	-36648	3.84	1728065	0	14.38
UUS-CL28	2160194	2160194	0	1.79	2160194	0	3.58	2160194	0	3.77	2160194	0	11.73
UUS-CL29	2199756	2187052	-12704	0.66	2199756	0	2.36	2187052	-12704	2.24	2199756	0	10.07
UUS-CL30	2215217	2193136	-22081	1.06	2215217	0	1.62	2193136	-22081	3.93	2215217	0	9.61
UUS-CL31	2228631	2195578	-33053	1.46	2220474	-8157	2.06	2195578	-33053	4.30	2228631	0	9.54
UUS-CL32	2237110	2200554	-36556	1.70	2220474	-16636	2.28	2200554	-36556	4.96	2237110	0	12.46
WUS-CL1	26298	26298	0	1.20	26298	0	2.70	26298	0	1.97	26298	0	10.13
WUS-CL2	3242	3077	-165	1.34	3242	0	2.86	3077	-165	2.86	3242	0	10.59
WUS-CL3	3131	3128	-3	1.82	3131	0	3.01	3128	-3	3.57	3131	0	10.75
WUS-CL4	12746	12746	0	0.73	12746	0	2.29	12746	0	2.20	12746	0	12.75
WUS-CL5	32250	32250	0	0.65	32250	0	1.23	32250	0	2.20	32250	0	9.82
WUS-CL6	32508	32508	0	1.88	32508	0	3.19	32508	0	1.00	32508	0	10.50
WUS-CL7	42414	41614	-800	1.16	41880	-534	2.24	41614	-800	2.13	42414	0	9.64
WUS-CL8	29247	29148	-99	1.62	29247	0	2.28	29148	-99	3.71	29247	0	12.92
WUS-CL9	117056	113821	-3235	1.43	115853	-1203	2.18	113821	-3235	4.20	117056	0	12.35
WUS-CL10	19260	18504	-756	0.63	19104	-156	2.33	18504	-756	4.16	19260	0	13.33
WUS-CL11	48919	47226	-1693	0.57	48919	0	1.26	47226	-1693	3.28	48919	0	11.86
WUS-CL12	136823	136372	-451	0.51	136726	-97	1.78	136372	-451	2.52	136823	0	7.70
WUS-CL13	125293	120926	-4367	1.01	125293	0	2.02	120926	-4367	3.57	125293	0	14.37
WUS-CL14	57578	57454	-124	1.28	57578	0	2.88	57454	-124	2.94	57578	0	13.36
WUS-CL15	42803	41315	-1488	1.03	41315	-1488	2.87	41315	-1488	2.48	42803	0	7.70
WUS-CL16	47174	47174	0	0.56	47174	0	1.83	47174	0	3.95	47174	0	11.81
WUS-CL17	16616	16616	0	1.67	16616	0	3.28	16616	0	4.54	16616	0	12.07
WUS-CL18	24288	24121	-167	1.00	24288	0	2.73	24121	-167	2.71	24288	0	11.31
WUS-CL19	4470	4470	0	0.89	4470	0	1.69	4470	0	4.62	4470	0	14.53
WUS-CL20	7539	7438	-101	0.85	7539	0	1.62	7438	-101	2.74	7539	0	8.86
WUS-CL21	13674	13326	-348	1.50	13674	0	2.50	13326	-348	3.82	13674	0	14.44
WUS-CL22	22337	21933	-404	0.72	22337	0	2.30	21933	-404	1.37	22337	0	8.27
WUS-CL23	25453	25412	-41	1.27	25453	0	3.12	25412	-41	4.94	25453	0	12.92
WUS-CL24	29065	27655	-1410	1.49	28470	-595	2.70	27655	-1410	2.52	29065	0	11.70
WUS-CL25	44600	43608	-992	0.62	44600	0	1.69	43608	-992	4.33	44600	0	11.67
WUS-CL26	59388	59388	0	1.62	59388	0	2.78	59388	0	2.55	59388	0	13.46
WUS-CL27	52524	51078	-1446	0.95	52524	0	2.17	51078	-1446	4.69	52524	0	13.16
WUS-CL28	63872	63872	0	1.32	63872	0	2.75	63872	0	1.35	63872	0	9.11
WUS-CL29	73321	71183	-2138	1.00	73321	0	2.01	71183	-2138	2.75	73321	0	12.52
WUS-CL30	82929	81831	-1098	1.72	82517	-412	3.53	81831	-1098	3.99	82929	0	12.62
WUS-CL31	82929	81831	-1098	1.55	82517	-412	2.73	81831	-1098	3.94	82929	0	11.90
WUS-CL32	82929	81831	-1098	1.64	82517	-412	2.94	81831	-1098	3.11	82929	0	10.92
Av_{cpu}				1.18			2.35			2.95			11.20
Av_{Gap}			-46524.5			-2393.125			-46524.5			0	
NB_{Opt}		13			46			13			64		

TABLE 5.1 – Résultats de LBS et GBS sur le premier groupe d'instances 3D.

#Inst.				LBS					
				$\eta = 1$			$\eta = 2$		
	Opt/Best	Sol.A1	Sol.A2	Sol.	Gap	cpu	Sol.	Gap	
s1-1	674874	665884	674874	665884	-8990	0.67	674874	0	4.10
s1-2	879456	764670	879456	764670	-114786	0.76	841920	-37536	3.45
s1-3	696346	696346	696346	696346	0	0.70	696346	0	6.53
s1-4	778540	778540	778540	778540	0	0.78	778540	0	5.31
s1-5	760320	760320	760320	760320	0	0.76	760320	0	3.70
s1-6	684000	684000	684000	684000	0	0.68	684000	0	3.22
s1-7	860832	860832	860832	860832	0	0.86	860832	0	4.12
s1-8	815984	744808	815984	744808	-71176	0.74	815984	0	8.09
s1-9	760242	731160	760242	731160	-29082	0.73	760242	0	4.61
s1-10	886968	886968	886968	886968	0	0.89	886968	0	4.38
s2-1	871904	840756	871904	840756	-31148	0.84	867256	-4648	7.80
s2-2	869904	869904	869904	869904	0	0.87	869904	0	3.10
s2-3	921801	904068	921801	904068	-17733	0.90	921801	0	2.23
s2-4	864036	864036	864036	864036	0	0.86	864036	0	10.32
s2-5	945856	945856	945856	945856	0	0.95	945856	0	2.07
s2-6	804056	804056	804056	804056	0	0.80	804056	0	9.64
s2-7	822000	822000	822000	822000	0	0.82	822000	0	10.49
s2-8	913050	913050	913050	913050	0	0.91	913050	0	2.88
s2-9	815112	815112	815112	815112	0	0.82	815112	0	7.68
s2-10	860360	860360	860360	860360	0	0.86	860360	0	3.86
s3-1	652356	652356	652356	652356	0	0.65	652356	0	10.94
s3-2	781800	668800	781800	668800	-113000	0.67	781800	0	2.08
s3-3	671756	671756	671756	671756	0	0.67	671756	0	3.33
s3-4	873408	873408	873408	873408	0	0.87	873408	0	9.04
s3-5	610362	610362	610362	610362	0	0.61	610362	0	5.09
s3-6	628024	613228	628024	613228	-14796	0.61	628024	0	3.79
s3-7	689386	689386	689386	689386	0	0.69	689386	0	6.55
s3-8	719292	719292	719292	719292	0	0.72	719292	0	1.87
s3-9	685952	680472	685952	680472	-5480	0.68	685952	0	3.35
s3-10	757484	754208	757484	754208	-3276	0.75	757484	0	7.09
s4-1	941611	941571	941571	941571	-40	0.94	941571	-40	6.13
s4-2	994680	985840	994680	985840	-8840	0.99	993445	-1235	8.52
s4-3	935834	908400	935834	908400	-27434	0.91	921472	-14362	8.50
s4-4	964951	964951	964951	964951	0	0.97	964951	0	11.33
s4-5	989232	973800	989232	973800	-15432	0.97	989232	0	4.64
s4-6	9793344	970068	979344	970068	-8823276	0.97	979344	-8814000	4.20
s4-7	982800	971964	982800	971964	-10836	0.97	982800	0	10.33
s4-8	994128	994128	994128	994128	0	0.99	994128	0	7.15
s4-9	975528	975528	975528	975528	0	0.98	975528	0	9.12
s4-10	964010	964010	964010	964010	0	0.96	964010	0	5.78
s5-1	927535995	875707050	927535995	875707050	-51828945	0.88	925844325	-1691670	2.80
s5-2	791047040	791047040	791047040	791047040	0	0.79	791047040	0	10.24
s5-3	910377398	881647112	910377398	881647112	-28730286	0.88	908602232	-1775166	8.89
s5-4	940778852	940778852	940778852	940778852	0	0.94	940778852	0	5.29
s5-5	915787200	911989512	915787200	911989512	-3797688	0.91	911989512	-3797688	2.28
s5-6	963108450	963108450	963108450	963108450	0	0.96	963108450	0	4.70
s5-7	923291784	923291784	923291784	923291784	0	0.92	923291784	0	10.62
s5-8	902788776	902788776	902788776	902788776	0	0.90	902788776	0	4.61
s5-9	968858510	941370090	968858510	941370090	-27488420	0.94	961128220	-7730290	5.50
s5-10	899738312	897397956	899738312	897397956	-2340356	0.90	897397956	-2340356	9.70
s6-1	942849066	926563320	942849066	926563320	-16285746	0.93	942849066	0	6.08
s6-2	978040000	978040000	978040000	978040000	0	0.98	978040000	0	7.36
s6-3	950359442	911602476	950359442	911602476	-38756966	0.91	950359442	0	4.83
s6-4	967049676	960380916	967049676	960380916	-6668760	0.96	962343300	-4706376	2.62
s6-5	931551291	924793251	931551291	924793251	-6758040	0.92	924793251	-6758040	4.73
s6-6	947620624	931404216	947620624	931404216	-16216408	0.93	946864248	-756376	5.22
s6-7	934394194	921150574	934394194	921150574	-13243620	0.92	925674374	-8719820	6.00
s6-8	955622192	941568480	955622192	941568480	-14053712	0.94	951178530	-4443662	3.34
s6-9	943481358	930553716	943481358	930553716	-12927642	0.93	943006354	-475004	4.43
s6-10	919206812	915519588	919206812	915519588	-3687224	0.92	915519588	-3687224	9.06
s7-1	972769756	959421760	972769756	959421760	-13347996	0.96	972769756	0	8.50
s7-2	967828720	949105792	967828720	949105792	-18722928	0.95	967828720	0	7.00
s7-3	979766032	968648632	979766032	968648632	-11117400	0.97	979766032	0	6.16
s7-4	955427687	944387560	955427687	944387560	-11040127	0.94	955427687	0	7.64
s7-5	966595002	959891760	966595002	959891760	-6703242	0.96	959891760	-6703242	5.96
s7-6	975298932	962195088	975298932	962195088	-13103844	0.96	975262854	-36078	2.26
s7-7	962651892	962651892	962651892	962651892	0	0.96	962651892	0	3.79
s7-8	970318920	959714760	970318920	959714760	-10604160	0.96	959714760	-10604160	6.75
s7-9	977788140	969662000	977788140	969662000	-8126140	0.97	969662000	-8126140	8.28
s7-10	954640294	954640294	954640294	954640294	0	0.95	954640294	0	3.68
s8-1	970553620	965572160	970553620	965572160	-4981460	0.97	968451600	-2102020	4.46
s8-2	983603448	969688700	983603448	969688700	-13914748	0.97	983603448	0	2.22
s8-3	976385802	967087360	976385802	967087360	-9298442	0.97	976385802	0	8.15
s8-4	968557842	962818210	968557842	962818210	-5739632	0.96	962818210	-5739632	5.50
s8-5	980177968	970571022	980177968	970571022	-9606946	0.97	980177968	0	9.49
s8-6	966197176	962895437	966197176	962895437	-3301739	0.96	962895437	-3301739	10.19
s8-7	980290128	974420748	980290128	974420748	-5869380	0.97	974420748	-5869380	8.71
s8-8	982244405	974992552	982244405	974992552	-7251853	0.98	979539513	-2704892	4.23
s8-9	977389640	977389640	977389640	977389640	0	0.98	977389640	0	6.34
s8-10	975962688	975962688	975962688	975962688	0	0.98	975962688	0	4.79
Av_{cpu}						0.88			5.98
Av_{Gap}					-5060114.69			-1261759.7	
NB_{Best}				0			0		

TABLE 5.2 – Résultats de LBS sur le deuxième groupe d'instances 3D -($\eta = 1$ et $\eta = 2$).

tivement, les résultats des algorithmes A1 et A2. Pour chaque valeur de η ($\eta = 1$ et $\eta = 2$), la table 5.4 montre la solution obtenue (notée Sol.), la valeur de la déviation par rapport à l'optimum (Gap) et le temps de résolution en secondes (noté cpu). De la même manière, les colonnes 5 à 10 de la table 5.5, montrent les résultats de GBS avec $\eta = 3$ et $\eta = 4$. Pour chaque version, la table rapporte la valeur de la solution réalisable finale, la déviation par rapport à l'optimum et le temps de résolutions en secondes.

Les trois dernière lignes de chaque table montrent, respectivement, la moyenne du temps de résolution (noté Av_{cpu}), la déviation moyenne par rapport à l'optimum (noté Av_{Gap}) et le nombre d'instances résolues à l'optimum (noté NB_{Opt}).

Les résultats obtenus sur le deuxième groupe d'instances, présentés dans les tables 5.2, 5.3, 5.4 et 5.5 montrent que :

#Inst.				LBS					
				$\eta = 3$			$\eta = 4$		
	Opt/Best	Sol.A1	Sol.A2	Sol.	Gap	cpu	Sol.	Gap	cpu
s1-1	674874	665884	674874	674874	0	15.32	674874	0	59.12
s1-2	879456	764670	879456	841920	-37536	24.70	879456	0	61.84
s1-3	696346	696346	696346	696346	0	30.38	696346	0	70.95
s1-4	778540	778540	778540	778540	0	18.55	778540	0	67.64
s1-5	760320	760320	760320	760320	0	24.58	760320	0	69.23
s1-6	684000	684000	684000	684000	0	22.39	684000	0	63.51
s1-7	860832	860832	860832	860832	0	26.85	860832	0	66.40
s1-8	815984	744808	815984	815984	0	32.32	815984	0	81.82
s1-9	760242	731160	760242	760242	0	17.99	760242	0	57.99
s1-10	886968	886968	886968	886968	0	15.83	886968	0	49.14
s2-1	871904	840756	871904	867256	-4648	31.32	871904	0	70.05
s2-2	869904	869904	869904	869904	0	23.38	869904	0	69.51
s2-3	921801	904068	921801	921801	0	20.49	921801	0	67.18
s2-4	864036	864036	864036	864036	0	25.57	864036	0	74.47
s2-5	945856	945856	945856	945856	0	16.20	945856	0	65.75
s2-6	804056	804056	804056	804056	0	36.02	804056	0	71.83
s2-7	822000	822000	822000	822000	0	36.88	822000	0	74.91
s2-8	913050	913050	913050	913050	0	25.21	913050	0	67.43
s2-9	815112	815112	815112	815112	0	29.29	815112	0	69.89
s2-10	860360	860360	860360	860360	0	26.10	860360	0	59.18
s3-1	652356	652356	652356	652356	0	33.05	652356	0	66.62
s3-2	781800	668800	781800	781800	0	16.61	781800	0	64.15
s3-3	671756	671756	671756	671756	0	18.70	671756	0	60.08
s3-4	873408	873408	873408	873408	0	22.65	873408	0	69.45
s3-5	610362	610362	610362	610362	0	25.00	610362	0	63.21
s3-6	628024	613228	628024	628024	0	26.21	628024	0	68.88
s3-7	689386	689386	689386	689386	0	21.63	689386	0	58.33
s3-8	719292	719292	719292	719292	0	25.15	719292	0	67.35
s3-9	685952	680472	685952	685952	0	19.49	685952	0	59.23
s3-10	757484	754208	757484	757484	0	33.15	757484	0	82.82
s4-1	941611	941571	941571	941571	-40	18.76	941571	-40	67.85
s4-2	994680	985840	994680	993445	-1235	24.95	994680	0	69.04
s4-3	935834	908400	935834	921472	-14362	23.55	935834	0	66.82
s4-4	964951	964951	964951	964951	0	30.60	964951	0	64.08
s4-5	989232	973800	989232	989232	0	31.06	989232	0	72.34
s4-6	9793344	970068	979344	979344	-8814000	19.24	979344	-8814000	60.40
s4-7	982800	971964	982800	982800	0	22.21	982800	0	72.81
s4-8	994128	994128	994128	994128	0	25.67	994128	0	66.32
s4-9	975528	975528	975528	975528	0	23.28	975528	0	70.87
s4-10	964010	964010	964010	964010	0	27.51	964010	0	68.62
s5-1	927535995	875707050	927535995	925844325	-1691670	15.50	927535995	0	62.29
s5-2	791047040	791047040	791047040	791047040	0	25.51	791047040	0	64.06
s5-3	910377398	881647112	910377398	908602232	-1775166	28.11	910377398	0	74.91
s5-4	940778852	940778852	940778852	940778852	0	18.92	940778852	0	55.06
s5-5	915787200	911989512	915787200	911989512	-3797688	16.51	915787200	0	52.65
s5-6	963108450	963108450	963108450	963108450	0	20.32	963108450	0	67.02
s5-7	923291784	923291784	923291784	923291784	0	34.71	923291784	0	81.68
s5-8	902788776	902788776	902788776	902788776	0	22.92	902788776	0	69.07
s5-9	968858510	941370090	968858510	961128220	-7730290	30.32	968858510	0	72.29
s5-10	899738312	897397956	899738312	897397956	-2340356	23.62	899738312	0	68.03
s6-1	942849066	926563320	942849066	942849066	0	24.72	942849066	0	60.19
s6-2	978040000	978040000	978040000	978040000	0	23.13	978040000	0	66.29
s6-3	950359442	911602476	950359442	950359442	0	27.35	950359442	0	68.49
s6-4	967049676	960380916	967049676	962343300	-4706376	15.34	967049676	0	58.59
s6-5	931551291	924793251	931551291	924793251	-6758040	22.38	931551291	0	56.29
s6-6	947620624	931404216	947620624	946864248	-756376	24.85	947620624	0	75.48
s6-7	934394194	921150574	934394194	925674374	-8719820	17.93	934394194	0	52.81
s6-8	955622192	941568480	955622192	951178530	-4443662	22.57	955622192	0	66.17
s6-9	943481358	930553716	943481358	943006354	-475004	22.22	943481358	0	67.28
s6-10	919206812	915519588	919206812	915519588	-3687224	33.04	919206812	0	69.42
s7-1	972769756	959421760	972769756	972769756	0	27.06	972769756	0	77.07
s7-2	967828720	949105792	967828720	967828720	0	32.98	967828720	0	77.51
s7-3	979766032	968648632	979766032	979766032	0	30.55	979766032	0	64.11
s7-4	955427687	944387560	955427687	955427687	0	32.63	955427687	0	67.93
s7-5	966595002	959891760	966595002	959891760	-6703242	18.15	966595002	0	66.13
s7-6	975298932	962195088	975298932	975262854	-36078	27.93	975298932	0	66.23
s7-7	962651892	962651892	962651892	962651892	0	21.84	962651892	0	70.10
s7-8	970318920	959714760	970318920	959714760	-10604160	31.58	970318920	0	70.88
s7-9	977788140	969662000	977788140	969662000	-8126140	29.74	977788140	0	73.94
s7-10	954640294	954640294	954640294	954640294	0	28.82	954640294	0	72.46
s8-1	970553620	965572160	970553620	968451600	-2102020	16.23	970553620	0	62.26
s8-2	983603448	969688700	983603448	983603448	0	25.71	983603448	0	72.37
s8-3	976385802	967087360	976385802	976385802	0	27.39	976385802	0	60.73
s8-4	968557842	962818210	968557842	962818210	-5739632	22.79	968557842	0	64.72
s8-5	980177968	970571022	980177968	980177968	0	27.81	980177968	0	68.27
s8-6	966197176	962895437	966197176	962895437	-3301739	28.59	966197176	0	68.99
s8-7	980290128	974420748	980290128	974420748	-5869380	24.14	980290128	0	73.60
s8-8	982244405	974992552	982244405	979539513	-2704892	17.33	982244405	0	57.23
s8-9	977389640	977389640	977389640	977389640	0	18.18	977389640	0	63.19
s8-10	975962688	975962688	975962688	975962688	0	26.47	975962688	0	70.40
Av_{cpu}						24.67			66.92
Av_{Gap}					-1261759.70			-110175.50	
NB_{Best}				0			0		

TABLE 5.3 – Résultats de LBS sur le deuxième groupe d'instances 3D -($\eta = 3$ et $\eta = 4$).

#Inst.				GBS					
				$\eta = 1$			$\eta = 2$		
	Opt/Best	Sol.A1	Sol.A2	Sol.	Gap	cpu	Sol.	Gap	cpu
s1-1	674874	665884	674874	674874	0	0.71	674874	0	4.19
s1-2	879456	764670	879456	841920	-37536	1.03	879456	0	4.59
s1-3	696346	696346	696346	696346	0	1.13	696346	0	3.06
s1-4	778540	778540	778540	778540	0	1.07	778540	0	4.78
s1-5	760320	760320	760320	760320	0	0.83	760320	0	3.85
s1-6	684000	684000	684000	684000	0	0.84	684000	0	3.94
s1-7	860832	860832	860832	860832	0	1.01	860832	0	3.77
s1-8	815984	744808	815984	815984	0	1.05	815984	0	3.94
s1-9	760242	731160	760242	760242	0	0.67	760242	0	4.00
s1-10	886968	886968	886968	886968	0	1.03	886968	0	4.73
s2-1	871904	840756	871904	867256	-4648	1.16	871904	0	3.30
s2-2	869904	869904	869904	869904	0	1.13	869904	0	4.48
s2-3	921801	904068	921801	921801	0	0.96	921801	0	3.79
s2-4	864036	864036	864036	864036	0	0.82	864036	0	2.86
s2-5	945856	945856	945856	945856	0	0.79	945856	0	3.38
s2-6	804056	804056	804056	804056	0	0.89	804056	0	3.04
s2-7	822000	822000	822000	822000	0	0.82	822000	0	4.36
s2-8	913050	913050	913050	913050	0	0.98	913050	0	3.26
s2-9	815112	815112	815112	815112	0	0.90	815112	0	2.51
s2-10	860360	860360	860360	860360	0	0.53	860360	0	4.52
s3-1	652356	652356	652356	652356	0	0.78	652356	0	3.02
s3-2	781800	668800	781800	781800	0	1.19	781800	0	3.36
s3-3	671756	671756	671756	671756	0	1.09	671756	0	3.29
s3-4	873408	873408	873408	873408	0	1.01	873408	0	3.31
s3-5	610362	610362	610362	610362	0	0.73	610362	0	4.24
s3-6	628024	613228	628024	628024	0	0.78	628024	0	3.19
s3-7	689386	689386	689386	689386	0	0.96	689386	0	4.92
s3-8	719292	719292	719292	719292	0	0.70	719292	0	3.04
s3-9	685952	680472	685952	685952	0	0.74	685952	0	3.50
s3-10	757484	754208	757484	757484	0	1.02	757484	0	4.06
s4-1	941611	941571	941571	941571	-40	0.95	941611	0	3.78
s4-2	994680	985840	994680	993445	-1235	0.59	994680	0	3.01
s4-3	935834	908400	935834	921472	-14362	0.66	935834	0	2.29
s4-4	964951	964951	964951	964951	0	0.73	964951	0	3.02
s4-5	989232	973800	989232	989232	0	0.98	989232	0	3.61
s4-6	9793344	970068	979344	979344	-8814000	0.90	9793344	0	3.93
s4-7	982800	971964	982800	982800	0	0.83	982800	0	2.40
s4-8	994128	994128	994128	994128	0	0.54	994128	0	4.20
s4-9	975528	975528	975528	975528	0	1.11	975528	0	3.75
s4-10	964010	964010	964010	964010	0	0.84	964010	0	3.03
s5-1	927535995	875707050	927535995	925844325	-1691670	0.73	927535995	0	3.91
s5-2	791047040	791047040	791047040	791047040	0	1.01	791047040	0	3.78
s5-3	910377398	881647112	910377398	908602232	-1775166	0.70	910377398	0	2.31
s5-4	940778852	940778852	940778852	940778852	0	0.93	940778852	0	4.94
s5-5	915787200	911989512	915787200	911989512	-3797688	0.75	915787200	0	4.78
s5-6	963108450	963108450	963108450	963108450	0	1.10	963108450	0	3.29
s5-7	923291784	923291784	923291784	923291784	0	1.01	923291784	0	3.46
s5-8	902788776	902788776	902788776	902788776	0	0.51	902788776	0	3.71
s5-9	968858510	941370090	968858510	961128220	-7730290	0.92	**968975430**	116920	4.12
s5-10	899738312	897397956	899738312	897397956	-2340356	0.64	**899909876**	171564	4.31
s6-1	942849066	926563320	942849066	942849066	0	0.65	942849066	0	2.66
s6-2	978040000	978040000	978040000	978040000	0	0.77	978040000	0	2.56
s6-3	950359442	911602476	950359442	950359442	0	1.11	950359442	0	4.25
s6-4	967049676	960380916	967049676	962343300	-4706376	1.19	967049676	0	4.87
s6-5	931551291	924793251	931551291	924793251	-6758040	0.61	**931596118**	44827	3.92
s6-6	947620624	931404216	947620624	946864248	-756376	0.55	**947620624**	0	3.74
s6-7	934394194	921150574	934394194	925674374	-8719820	0.99	**934400234**	6040	3.41
s6-8	955622192	941568480	955622192	951178530	-4443662	0.93	**955629758**	7566	2.56
s6-9	943481358	930553716	943481358	943006354	-475004	0.86	**943499680**	18322	3.77
s6-10	919206812	915519588	919206812	915519588	-3687224	0.68	**919219796**	12984	2.67
s7-1	972769756	959421760	972769756	972769756	0	0.83	972769756	0	3.90
s7-2	967828720	949105792	967828720	967828720	0	0.54	967828720	0	4.01
s7-3	979766032	968648632	979766032	979766032	0	1.03	979766032	0	3.43
s7-4	955427687	944387560	955427687	955427687	0	0.89	955427687	0	2.51
s7-5	966595002	959891760	966595002	959891760	-6703242	0.70	966595002	0	2.21
s7-6	975298932	962195088	975298932	975262854	-36078	0.85	975298932	0	4.84
s7-7	962651892	962651892	962651892	962651892	0	1.12	962651892	0	4.58
s7-8	970318920	959714760	970318920	959714760	-10604160	0.56	**970456430**	137510	4.27
s7-9	977788140	969662000	977788140	969662000	-8126140	1.01	977788140	0	5.09
s7-10	954640294	954640294	954640294	954640294	0	0.93	**954676430**	36136	4.28
s8-1	970553620	965572160	970553620	968451600	-2102020	0.84	970553620	0	2.43
s8-2	983603448	969688700	983603448	983603448	0	0.55	983603448	0	2.51
s8-3	976385802	967087360	976385802	976385802	0	0.78	976385802	0	2.44
s8-4	968557842	962818210	968557842	962818210	-5739632	0.51	**968576543**	18701	2.04
s8-5	980177968	970571022	980177968	980177968	0	0.99	980177968	0	4.39
s8-6	966197176	962895437	966197176	962895437	-3301739	0.55	966197176	0	4.22
s8-7	980290128	974420748	980290128	974420748	-5869380	1.02	980290128	0	3.48
s8-8	982244405	974992552	982244405	979539513	-2704892	1.00	**982244734**	329	4.34
s8-9	977389640	977389640	977389640	977389640	0	0.51	977389640	0	3.93
s8-10	975962688	975962688	975962688	975962688	0	1.06	975962688	0	10.69
Av_{cpu}						0.86			3.72
Av_{Gap}					-1261759.70			7136.23	
NB_{Best}				0			12		

TABLE 5.4 – Résultats de GBS sur le deuxième groupe d'instances 3D -($\eta = 1$ et $\eta = 2$).

#Inst.				GBS					
				$\eta = 3$			$\eta = 4$		
	Opt/Best	Sol.A1	Sol.A2	Sol.	Gap		Sol.	Gap	
s1-1	674874	665884	674874	674874	0	13.54	674874	0	48.46
s1-2	879456	764670	879456	879456	0	11.49	879456	0	41.85
s1-3	696346	696346	696346	696346	0	17.57	696346	0	39.98
s1-4	778540	778540	778540	778540	0	13.66	778540	0	47.40
s1-5	760320	760320	760320	760320	0	13.92	760320	0	39.11
s1-6	684000	684000	684000	684000	0	16.37	684000	0	44.34
s1-7	860832	860832	860832	860832	0	14.40	860832	0	35.78
s1-8	815984	744808	815984	815984	0	14.02	815984	0	41.32
s1-9	760242	731160	760242	760242	0	12.30	760242	0	37.64
s1-10	886968	886968	886968	886968	0	16.49	886968	0	50.22
s2-1	871904	840756	871904	871904	0	15.48	871904	0	48.10
s2-2	869904	869904	869904	869904	0	12.40	869904	0	37.91
s2-3	921801	904068	921801	921801	0	11.17	921801	0	33.69
s2-4	864036	864036	864036	864036	0	17.34	864036	0	47.02
s2-5	945856	945856	945856	945856	0	17.10	945856	0	46.03
s2-6	804056	804056	804056	804056	0	10.83	804056	0	37.19
s2-7	822000	822000	822000	822000	0	12.72	822000	0	45.40
s2-8	913050	913050	913050	913050	0	12.36	913050	0	44.51
s2-9	815112	815112	815112	815112	0	16.78	815112	0	40.76
s2-10	860360	860360	860360	860360	0	13.42	860360	0	40.29
s3-1	652356	652356	652356	652356	0	11.45	652356	0	44.51
s3-2	781800	668800	781800	781800	0	16.91	781800	0	50.57
s3-3	671756	671756	671756	671756	0	10.64	671756	0	33.43
s3-4	873408	873408	873408	873408	0	15.82	873408	0	38.27
s3-5	610362	610362	610362	610362	0	11.34	610362	0	38.52
s3-6	628024	613228	628024	628024	0	14.75	628024	0	41.69
s3-7	689386	689386	689386	689386	0	16.48	689386	0	50.35
s3-8	719292	719292	719292	719292	0	11.10	719292	0	33.43
s3-9	685952	680472	685952	685952	0	17.77	685952	0	42.52
s3-10	757484	754208	757484	757484	0	13.09	757484	0	33.65
s4-1	941611	941571	941571	941611	0	16.83	941611	0	44.96
s4-2	994680	985840	994680	994680	0	9.37	994680	0	29.97
s4-3	935834	908400	935834	935834	0	12.23	935834	0	37.82
s4-4	964951	964951	964951	964951	0	15.99	964951	0	42.61
s4-5	989232	973800	989232	989232	0	9.74	989232	0	41.82
s4-6	9793344	970068	979344	9793344	0	15.19	9793344	0	44.85
s4-7	982800	971964	982800	982800	0	10.84	982800	0	31.33
s4-8	994128	994128	994128	994128	0	14.76	994128	0	40.21
s4-9	975528	975528	975528	975528	0	13.52	975528	0	48.00
s4-10	964010	964010	964010	964010	0	11.06	964010	0	45.92
s5-1	927535995	875707050	927535995	927535995	0	12.59	927535995	0	41.17
s5-2	791047040	791047040	791047040	791047040	0	18.12	791047040	0	41.49
s5-3	910377398	881647112	910377398	910377398	0	15.06	910377398	0	37.37
s5-4	940778852	940778852	940778852	940778852	0	15.61	940778852	0	37.33
s5-5	915787200	911989512	915787200	915787200	0	14.20	915787200	0	41.93
s5-6	963108450	963108450	963108450	963108450	0	14.42	963108450	0	43.77
s5-7	923291784	923291784	923291784	923291784	0	13.94	923291784	0	37.64
s5-8	902788776	902788776	902788776	902788776	0	13.50	902788776	0	39.75
s5-9	968858510	941370090	968858510	968975430	116920	18.36	968975430	116920	43.19
s5-10	899738312	897397956	899738312	899909876	171564	11.70	899909876	171564	38.42
s6-1	942849066	926563320	942849066	942849066	0	13.50	942849066	0	36.29
s6-2	978040000	978040000	978040000	978040000	0	15.39	978040000	0	49.33
s6-3	950359442	911602476	950359442	950359442	0	12.39	950359442	0	38.91
s6-4	967049676	960380916	967049676	967049676	0	13.01	967049676	0	45.96
s6-5	931551291	924793251	931551291	931596118	44827	16.66	931596118	44827	42.33
s6-6	947620624	931404216	947620624	947620624	0	12.09	947620624	0	33.79
s6-7	934394194	921150574	934394194	934400234	6040	12.48	934400234	6040	36.70
s6-8	955622192	941568480	955622192	955629758	7566	14.45	955629758	7566	36.89
s6-9	943481358	930553716	943481358	943499680	18322	13.41	943499680	18322	37.80
s6-10	919206812	915519588	919206812	919219796	12984	15.52	919219796	12984	45.34
s7-1	972769756	959421760	972769756	972769756	0	10.90	972769756	0	34.46
s7-2	967828720	949105792	967828720	967828720	0	16.18	967828720	0	40.38
s7-3	979766032	968648632	979766032	979766032	0	14.13	979766032	0	34.62
s7-4	955427687	944387560	955427687	955427687	0	13.35	955427687	0	46.45
s7-5	966595002	959891760	966595002	966595002	0	14.98	966595002	0	37.70
s7-6	975298932	962195088	975298932	975298932	0	16.34	975298932	0	40.74
s7-7	962651892	962651892	962651892	962651892	0	19.30	962651892	0	52.47
s7-8	970318920	959714760	970318920	970456430	137510	16.69	970456430	137510	42.62
s7-9	977788140	969662000	977788140	977788140	0	11.45	977788140	0	41.04
s7-10	954640294	954640294	954640294	954676430	36136	14.92	954676430	36136	43.20
s8-1	970553620	965572160	970553620	970553620	0	13.97	970553620	0	37.81
s8-2	983603448	969688700	983603448	983603448	0	8.86	983603448	0	34.06
s8-3	976385802	967087360	976385802	976385802	0	13.12	976385802	0	38.99
s8-4	968557842	962818210	968557842	968576543	18701	15.18	968576543	18701	39.40
s8-5	980177968	970571022	980177968	980177968	0	11.32	980177968	0	33.48
s8-6	966197176	962895437	966197176	966197176	0	16.67	966197176	0	46.94
s8-7	980290128	974420748	980290128	980290128	0	9.59	980290128	0	42.51
s8-8	982244405	974992552	982244405	982244734	329	18.58	982244734	329	50.76
s8-9	977389640	977389640	977389640	977389640	0	17.00	977389640	0	40.52
s8-10	975962688	975962688	975962688	975962688	0	24.91	975962688	0	50.64
Av_{cpu}						14.20			41.19
Av_{Gap}					7136.23			7136.23	
NB_{Best}				12			12		

TABLE 5.5 – Résultats de GBS sur le deuxième groupe d'instances 3D -($\eta = 3$ et $\eta = 4$).

- Les différentes version de LBS (avec $\eta = 1$, $\eta = 2$, $\eta = 3$ et $\eta = 4$) n'améliorent aucune solution dans ce groupe. Le temps de traitement passe de 0.88 secondes pour LBS_1 à 5.98 secondes pour LBS_2 et de 24.67 secondes pour LBS_3 à 66.92 secondes dans le cas de LBS_4.
- La déviation moyenne par rapport aux meilleures solutions de la littérature, diminue au fur et à mesure de l'augmentation de la largeur du faisceau. Elle passe de -5060114.69 (LBS_1) à -1261759.7 (LBS_2) et de -1261759.7 (LBS_3) à -110175.5 (LBS_4).
- Nous remarquons que la qualité des solutions produites par (LBS_2) et (LBS_3) sont identiques et que LBS_3 consomme sensiblement plus que LBS_2 en temps de calcul.
- GBS_1 n'arrive pas à améliorer les meilleurs solutions de la littérature mais produisent des déviations nettement inférieures à $LBS_1(-1261759.7)$. Le temps de résolution de GBS_1 est un peut plus inférieur au temps consommé par LBS_1 à 0.86.
- GBS_2 améliore 12 instances parmi les 80 instances de ce groupe et génère une déviation moyenne de 7136.23. Son temps de résolution moyen reste très raisonnable à 3.72 secondes.
- GBS_3 et GBS_4 n'améliorent pas les résultats de GBS_2 et ils augmentent considérablement le temps de résolution qui passe à 14.20 pour GBS_3, puis à 41.19 secondes pour GBS_3.

5.6 Conclusion

Dans ce chapitre, nous avons proposé une résolution approchée du problème de découpe à trois dimensions avec une méthode de recherche par faisceaux. Cette méthode est une méthode de séparation et évaluation qui explore un sous-ensemble de nœuds à chaque niveau. Le choix des nœuds à explorer se fait en utilisant une fonction d'évaluation nommée aussi, politique de filtrage. La partie expérimentale a montré l'intérêt de la politique de filtrage et son influence directe sur la qualité des solutions réalisables produites.

Troisième partie

RÉSOLUTION SÉQUENTIELLE EXACTE

Chapitre 6

Algorithme exact pour le problème de découpe double contraint à deux dimensions [3]

Contenu

Dans ce chapitre, nous nous intéressons au problème de découpe double contraint à deux dimensions noté *DCTDC (Double-Constrained Two-Dimensional Guillotine Cutting Stock Problem).* Nous proposons, une résolution exacte par séparation et évaluation (branch and bound). Cet algorithme opère par des constructions par pièces et utilise des bornes supérieures et inférieures pour limiter l'espace de recherche.

On présente aussi dans ce chapitre, une nouvelle borne inférieure pour le DCTDC basée sur une relaxation du DCTDC.

3. *Le contenu de ce chapitre a fait l'objet d'une publication dans "Computational Optimization and Applications [41]" et d'une conférence internationale "The 3rd ESICUP Meeting, EURO Special Interest Group on Cutting and Packing [7]"*

6.1 Problème de découpe double contraint à deux dimensions

Comme nous l'avons définie dans le premier chapitre de cette thèse, une instance du problème de découpe double contraint à deux dimensions est définie par (R, S, c, a, b), où R représente le rectangle initial, S l'ensemble des pièces, c le vecteur profit associé aux pièces et $a = (a_i), i = 1, ..., n$ et $b = (b_i)i = 1, ..., n$ représentent respectivement, une borne inférieure et une borne supérieure sur le nombre d'occurrences de chaque type de pièces dans un plan de découpe réalisable.

1. Un plan de découpe réalisable est produit par des découpes guillotines successives.
2. Les pièces de l'ensemble S sont fixées (i.e. la pièce de longueur l et de hauteur h n'est pas la même que celle qui possède la longueur h et la hauteur l).
3. Les dimensions L, H, l_i et h_i, pour $i = 1, \ldots, n$, sont des entiers positifs.

Le vecteur de n-dimensions d'entiers non négatifs $(d_1, d_2, \ldots, d_n)$ est un plan de découpe pour le DCTDC s'il est possible de placer d_i pièces de type $i \in I$ dans le rectangle initial S sans débordement et s'il est possible de produire ces pièces par des découpes guillotines. Ce plan de découpe est réalisable si $\forall i \in i, a_i \leq d_i \leq b_i$.

La formulation du DCTDC [1] est la suivante :

$$(DCTDC) = \begin{cases} \text{maximiser} \sum_{i \in I} c_i d_i \\ \text{s.c.}(d_i, \ldots, d_n) \text{ est un plan de découpe réalisable} \\ a_i \leq d_i \leq b_i, i \in I. \end{cases}$$

Lorsque $a_i = 0, \forall i \in I$ le problème est noté CTDC [2] et lorsque en plus $b_i = +\infty, \forall i \in I$, ce problème devient UTDC [3].
On trouve aussi dans la littérature, une autre variante de ce problème, nommée problème de découpe avec satisfaction exacte des demandes. Cette variante est le DCTDC avec $a_i = b_i, \forall i \in I$.

6.2 Une solution initiale

DCTDC étant NP-complet, nous obtenons une solution initiale réalisable en utilisant une procédure d'approximation du CTDC (single constrained TDC).
En effet, un problème CTDC peut être obtenu à partir du DCTDC en ignorant les bornes inférieures de toutes les pièces. Dans le cas où, la solution produite par la résolution du CTDC est réalisable pour DCTDC, alors cette dernière est utilisée comme solution initiale pour l'algorithme exact, sinon, nous utilisons une procédure gloutonne pour restaurer la réalisabilité.

1. *Double-Constrained Two-Dimensional Guillotine Cutting Stock Problem*
2. *Single constrained Two-Dimensional Guillotine Cutting Stock Problem*
3. *Unconstrained Two-Dimensional Guillotine Cutting Stock Problem*

Dans la suite, nous commençons par décrire une procédure constructive qui résout le problème CTDC. Nous détaillerons, par la suite, les procédures GP et IGP qui permettent de restaurer la réalisabilité d'une solution CTDC pour DCTDC.

6.2.1 Procédure constructive pour une contrainte

Le problème CTDC est NP-difficile et sa résolution est très complexe. Pour cette raison nous faisons une résolution approximative en utilisant une approche constructive qui génère et combine des bandes. Le principe de cette méthode est décrit dans la figure 6.1.

1. Soit r le nombre de hauteurs distinctes des pièces telle que $\overline{w}_1 > \overline{w}_2 > ... > \overline{w}_r$, où $r \leq n$. $b_i^{'} = b_i$, $d_i^{'} = 0$, et $\sigma = 0$. σ représente le nombre de bandes dans la solution finale ;
2. Mettre $j = 1$ et $W^{'} = W$;
3. Générer une bande de hauteur $\overline{w}_j$ par la résolution du problème de sac-à-dos borné suivant :

$$KP^{hor}_{(L,\overline{w}_j)} = \begin{cases} f^{hor}_{\overline{w}_j}(L) = \max \sum_{i \in S_{\overline{w}_j}} c_i d_{ij} \\ \text{s.c.} \sum_{i \in S_{\overline{w}_j}} l_i d_{ij} \leq L \\ d_{ij} \leq b_i^{'}, d_{ij} \in N, i \in S_{L,\overline{w}_j}. \end{cases}$$

   où d_{ij} est le nombre d'apparitions de la pièce de type i dans la bande $(L, \overline{w}_j)$ sans violation de la borne supérieure $b_i^{'}$. $f^{hor}_{\overline{w}_j}(L)$ le profit de la bande et $S_{L,\overline{w}_j} = \left\{ i \in I, w_i \leq \overline{w}_j, l_i \leq L, b_i^{'} > 0 \right\}$, cela signifie que $S_{L,\overline{w}_j}$ est l'ensemble de pièces qui peuvent être placées dans la bandes $(L, \overline{w}_j)$ et dont la borne supérieure $b_i^{'}$ est positive ;
4. Mettre $d_i^{'} = d_i^{'} + d_{ij}, i \in I$;
5. Mettre à jour les restes des demandes en mettant $b_i^{'} = b_i^{'} - d_{ij}, i \in I$ et incrémenter le nombre courant de bandes $\sigma = \sigma + 1$;
6. Placer la bande courante dans le reste du rectangle $(L, W^{'})$ et réduire sa hauteur $W^{'} = W^{'} - \overline{w}_j$;
7. Si $\left\{ k \in I : w_k = \overline{w}_j, b_k^{'} > 0 \right\} = \emptyset$ alors
   - Si $\exists i \in I$, tel que $b_i^{'} > 0, W^{'} \geq w_i$, Mettre $j = i$ et aller à l'étape 3 ;
   - Sinon Arrêter ;

   Sinon aller à l'étape 3 ;

FIGURE 6.1 – Procédure constructive pour CTDC

L'exécution de l'étape 3 plusieurs fois, génère des bandes de hauteurs $\overline{w}_j, j \in 1, \ldots, r$. Évidemment, la structure de deux bandes de même hauteurs peut être différente.

6.2.2 Restauration de la réalisabilité

La solution $d^{'} = (d_1^{'}, \ldots, d_n^{'})$ produite par la procédure de la section précédente, est réalisable pour le problème CTDC, mais elle n'est pas nécessairement réalisable pour DCTDC. En effet, dans la solution produite, on peut trouver des pièces $i \in I$ dont la contrainte sur la borne inférieure n'est pas satisfaite ($d_i^{'} < a_i$). Pour restaurer la réalisabilité, nous appliquons la procédure gloutonne décrite dans la figure 6.2.

1. Soit σ, le nombre de bandes utilisées pour construire $d^{'}$;
2. Soit $S_\sigma = \{i \in I \| d_i^{'} < a_i\}$ et $\overline{S}_\sigma = \{i \in I \| d_i^{'} > a_i\}$ et $n_{strip} = 0$;
3. Soit $j = 1$;
4. Soit $J \subset \overline{S}_\sigma$ l'ensemble des pièces de la bande j de $d^{'}$ dont les contraintes a_i sont satisfaites.
5. $J = \{i \| i \in \overline{S}_\sigma$ et $d_{ij}^{'} > 0\}$;
   - (a) Supprimer de la bande j, toutes les pièces de J ; mettre à jour $d^{'}$ en mettant $d_i^{'} = d_i^{'} - d_{ij}^{'}$, pour $i \in J$;
   - (b) Décaler toutes les pièces de la bande j au coin inférieur droit de la bande (bottom-left) ;
   - (c) Définir l'espace restant dans la bande E ;
     Pour chaque type de pièce $i \in \overline{S}_\sigma$, placer dans E (en utilisant la politique BLP) le maximum de pièces possible, sans dépasser $a_i - d_i^{'}$ pièces ;
     Si aucune pièce de $\overline{S}_\sigma$ ne rentre dans E, utiliser des pièces de S_σ ;
   - (d) Mettre à jour S_σ et $\overline{S}_\sigma$;
   - (e) Incrémenter $n_{strip} = n_{strip} + 1$;
     Si $S_\sigma = \emptyset$, arrêter, la solution est réalisable ;
     Si $S_\sigma \neq \emptyset$ et $\overline{S}_\sigma = \emptyset$, arrêter, la solution est irréalisable ;
     Si $S_\sigma \neq \emptyset$, $\overline{S}_\sigma \neq \emptyset$ et $n_{strip} > \sigma$, arrêter, la solution est irréalisable ;
     Si $S_\sigma \neq \emptyset$, $\overline{S}_\sigma \neq \emptyset$ et $n_{strip} \leq \sigma$, aller à l'étape 3 ;

FIGURE 6.2 – Restauration de la réalisabilité par GP

L'étape 3 de la procédure GP restaure la réalisabilité en supprimant des pièces d'une bande et en les remplaçant par des pièces dont les bornes inférieures sont violées. Dans ce cas, si aucune pièce ne rentre dans l'espace libéré, alors GP utilise les autres pièces sans violer la contrainte sur les bornes supérieures.
La procédure GP produit une solution réalisable, si elle arrive à satisfaire les contraintes inférieures de toutes les pièces $S_\sigma = \emptyset$. Dans le cas contraire, GP s'arrête sans produire de solution réalisable.

6.2.3 Procédure de discrétisation

Afin d'améliorer les solutions obtenues par GP et dans le cas échéant, trouver une solution réalisable, nous avons introduit une deuxième étape.
Cette étape, crée un ensemble R en utilisant une procédure de discrétisation et combine les éléments de R dans une méthode constructive.

R est un ensemble fini de couples $\overline{R} = (\overline{R}_y, \overline{R}_{W-y}), y \in w_1, w \ldots, w_r$ qui représente deux sous-rectangles complémentaires dans le cas, d'une découpe horizontale.

Par symétrie, dans le cas d'une découpe verticale, l'ensemble $\overline{R}$ sera $\overline{R} = (\overline{R}_x, \overline{R}_{l-x}), y \in l_1, w \ldots, l_k$, où k est le nombre de largeurs distincts des pièces.

Initialisation

$Sol_H = -\infty$;

$Sol_V = -\infty$;

où Sol_H (Sol_V) la solution produite par GP ;

$Best = max\{0, Sol_H, Sol_V\}$; où $Best$ la meilleure solution courante ;

Amélioration

$\forall k \in (1, \ldots, r)$, faire :

$W_{rest} = W - w_k$ et $Stop = 0$;

Répéter

1. Remplir R_{yk} en utilisant GP horizontale ;
2. Remplir $R_{W_{rest}}$ en utilisant GP verticale ;
3. $Best = max\{Best, Best(R_{yk}, R_{W_{rest}})\}$, où $Best(R_{yk}, R_{W_{rest}})$ la valeur de la solution de $(R_{yk}, R_{W_{rest}})$;
4. Si $\exists (l_j, w_j) \subseteq (L, w)$, tel que $w \in w_1, \ldots, w_r$ et $w \leq W_rest$ alors $y_k = y_k + w_i$; Sinon $Stop = 1$;

jusqu'à $Stop = 1$

FIGURE 6.3 – Procédure de combinaison de bandes horizontales et/ou verticales : IGP

Le principe de résolution par programmation dynamique du sac-à-dos borné $KP^{hor}_{(L,W)}$, détaillé dans les chapitres précédents, nous permet non seulement de construire la bande (L, W), mais aussi toutes les bandes optimales (L, β), où $\beta \leq W$.

La deuxième phase que nous proposons (nommée IGP), met en œuvre ce principe pour obtenir les couples $(\overline{R}_y, \overline{R}_{W-y}) \in R$ dans le cas d'une découpe horizontale et les couples $(\overline{R}_x, \overline{R}_{l-x}) \in R$ dans le cas d'une découpe verticale.

IGP est une procédure constructive, qui démarre d'une solution initiale produite par la procédure GP et tente de l'améliorer en fixant une hauteur $y \in w_1, \ldots, w_r$ (ou une largeur $x \in l_1, \ldots, l_r$, dans le cas d'une découpe verticale) dans S, remplit le premier sous-rectangle (L, y) (ou (x, W) dans le cas d'une découpe verticale) et combine les bandes horizontales et verticales en utilisant GP. Les étapes $1 - 4$ de la procédure IGP décrite dans la figure 6.3 sont répétées jusqu'à ce que toutes les pièces soient placées ou jusqu'a ce qu'aucune pièce ne peut entrer dans la dernière bande créée.

6.3 Résolution exacte

Nous nous sommes intéressés, dans cette section à la résolution exacte du DCTDC. L'algorithme que nous proposons est une méthode de séparation et évaluation qui utilise la solution réalisable produite par la méthode décrite dans la section précédente.
Nous proposons aussi, des bornes supérieures pour réduire l'espace de recherche, accélérer et orienter

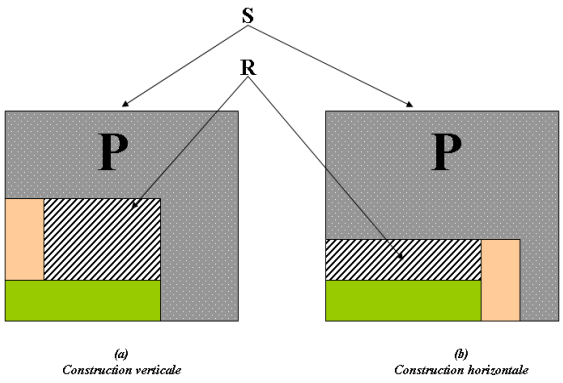


FIGURE 6.4 – Constructions horizontale et verticale.

la recherche de la solution optimale.

6.3.1 Principe

Chaque nœud de l'arborescence de recherche correspond à un sous-rectangle réalisable de S. Un sous-rectangle réalisable du rectangle initial S est obtenu par des constructions horizontales ou verticales.

6.3.1.1 Définitions

Une construction horizontale (notée $R_1|R_2$) de deux sous-rectangles $R_1 = (x_{R_1}, y_{R_1})$ et $R_2 = (x_{R_2}, y_{R_2})$ produit le sous-rectangle $R_3 = (x_{R_1} + x_{R_2}) \times \text{Max}(y_{R_1}, y_{R_2})$.

De la même manière, une construction verticale (notée $\frac{R_1}{R_2}$) de R_1 et R_2 produit le sous-rectangle $R_4 = \text{Max}(x_{R_1}, x_{R_2}) \times (y_{R_1} + y_{R_2})$. (La figure 6.4 illustre les constructions horizontale et verticale).

Un sous-rectangle interne $R = (x_R, y_R)$ est réalisable si :
- $\forall (l_i, w_i) \subseteq R, d_i^R \leq b_i$, où d_i^R est le nombre d'apparitions de la pièce du type i dans le rectangle R.
- R peut être placé dans le rectangle initiale S, autrement dit $(x_R, y_R) \leq (L, W)$
- $a_i \leq d_i^R + d_i^P \leq b_i, i \in I$ (Comme illustré dans la figure 6.4, $P = S/R$ et R est placé dans le coin inférieur droit de S.

6.3.1.2 Processus de développement

Pour chaque sous-rectangle interne R, l'algorithme calcule une borne supérieure. Soit $g(R)$ la valeur de la somme des profits des pièces contenues dans R et soit $h(R)$ le profit maximal qu'on peut obtenir en découpant la région $P = S/R$ sans violer les contraintes sur les demandes (les bornes a_i et b_i, $\forall i \in I$). Alors $f(R) = g(R) + h(R)$ représente une borne supérieure pour le profit maximal produit par n'importe quel plan de découpe incluant R.

Calculer $h(R)$ n'est pas une tâche facile, pour se faire, nous commençons par déterminer une estimation $h^{'}(R)$ de $h(R)$, ensuite on déduit à partir de $h^{'}(R)$, une estimation de la borne supérieure $f^{'}(R)$ de $f(R)$. Nous présentons dans ce qui suit, plusieurs approches d'estimation de $f(R)$.

Par ailleurs, à chaque construction, l'algorithme exact que nous proposons, branche sur le meilleur sous-rectangle interne réalisable, susceptible d'améliorer la solution courante.

Les étapes principales de l'algorithme sont détaillées dans la section 6.3.2. Les bornes supérieures dans la section 6.3.3, la stratégie de développement dans la section 6.3.4 , les structures de données dans la section 6.3.4.4 et finalement, la gestion des duplications et des plans de découpes non-réalisables sont décrits dans la section 6.3.4.3.

6.3.2 L'algorithme

L'algorithme exact (voir la figure 6.5) utilise deux listes : *Open* et *Clist*.
La liste *Open* contient initialement n éléments, chaque élément $R_i, i \in I$ correspond à un type de pièce i, cela veut dire que $(l_{R_i}, w_{R_i}) = (l_i, w_i), i \in I$.
Pour chaque élément R_i de la liste *Open*, l'algorithme associe trois valeurs : (i) la valeur $g(R_i)$, (ii) une estimation $g^{'}(R_i)$ et (iii) la valeur de la borne supérieure correspondant $f^{'}(R_i)$. L'algorithme associe à chaque élément R_i, un vecteur $d^{(R_i)}$ de dimension n. Chaque élément de ce vecteur (noté $d_k^{(R_i)}, k = 1, \ldots, n$) présente le nombre d'occurrences de la pièce de type k dans le sous-rectangle interne réalisable R_i.

Initialement, $g(R_i) = c_i$, $d_k^{(R_i)} = 1$ si $k = 1$ sinon $d_k^{(R_i)} = 0$.
La liste *Clist* est initialisée à l'ensemble vide. A chaque itération l'élément R de plus grande valeur $f^{'}(R)$ est déplacé de la liste *Open* à *Clist*.

Un nouvel ensemble Q est crée en combinant (en horizontal et en vertical) R avec les éléments de la liste *Clist* (en incluant le rectangle R). A chaque combinaison, la liste Q ne garde que les sous-rectangles réalisables q ($(l_q, w_q) \leq (L, W)$ et $d_k^{(q)} \leq b_k$, $k \in I$). La solution réalisable courante $Opt(best)$ est mise à jour par la valeur $g(R)$ sous deux conditions : (i) $d_k^{(q)} \geq a_k, k \in I$ et (ii) $g(q) > Opt(best)$. Finalement, l'algorithme s'arrête lorsque *Open* est vide ou si la valeur $f^{'}(R)$ est inférieure ou égale à la meilleure solution courante $Opt(best)$.

6.3.3 Bornes supérieures

Pour chaque sous-rectangle interne R, l'algorithme calcule une borne supérieure, nous proposons plusieurs manières de calculer une estimation de la borne supérieure $f^{'}(R)$ de S qui inclut le sous-rectangle $R = x_R \times y_R$.
La borne supérieure $f^{'}(R)$ est la somme de $g(R)$ et $h^{'}(R)$ ($f^{'}(R) = g(R) + h^{'}(R)$), où $h^{'}(R)$ est le profit maximum obtenu par la découpe de la région $P = S\mathcal{R}$.
Cette borne est obtenue par la résolution d'un problème UTDC, d'un problème de sac-à-dos borné unidimensionnel (SBK-single bounded knapsack) et d'une relaxation du SBK (RSBK).

`Entrée` : Instance DCTDC ;

`Sortie` : Une solution optimale *Best* de valeur *Opt* ;

Structure de données

- Open : Ensemble de sous-problèmes ;
- Clist : Liste des meilleurs sous-problèmes ;
- g,R : Des rectangles guillotines ;
- $g(R)$, $h^{'}(R)$, $f^{'}(R)$: La valeur du profit du rectangle R, la borne supérieure de la région P et la valeur de la borne supérieure du sous-problème contenant la configuration R ;
- *Best*, *Opt* : La structure de la meilleure solution courante et sa valeur ;

Résolution

- $Open = \{R_1, \ldots, R_n\}$;
- $Clist = \emptyset$;$Fin = 0$;
- Calculer $Upper(x_R, y_R)$ pour chaque $x_R = 1, \ldots, L$ et $y_R = 1, \ldots, W$;
- Lancer IGP afin de produire une solution de départ *best* de valeur $Opt(Best)$;

Répéter

Sélectionner de *Open* l'élément R ayant la plus grande valeur $f^{'}(R)$;

Si $(f^{'}(R) - Opt(Best) \leq 0)$ alors $fin = 1$;

Sinon

1. Transférer R de *Open* à *Clist* ;
2. Construire tous les rectangles guillotines Q tels que :
   (a) Q est une construction horizontale ou verticale de quelques sous-rectangles (incluant R) de *Clist* ;
   (b) $Q \leq (L, W)$;
   (c) $\forall q \in Q, d_k^q \leq b_k, k \in I$;
3. Pour chaque élément q de Q faire :
   (a) Si $(g(q) > Opt(Best))$ et $(d_k^q \geq a_k)$ alors $Best = q$ et $Opt(Best) = g(q)$;
   (b) Si $(f^{'}(q) > Opt(Best))$ alors $Open = Open \cup q$;
4. Supprimer tous les éléments R de *Open* ayant $f^{'}(R) \leq Open(Best)$ ou $h^{'}(R) = 0$;
5. Si $Open = \emptyset$ alors $fin = 1$;

jusqu'à $fin = 1$;

FIGURE 6.5 – Algorithme exact pour DCTDC

6.3.3.1 Une première borne supérieure

Une borne supérieure pour DCTDC est la solution obtenue par la résolution de UTDC (une version de DCTDC où les contraintes sur les demandes (a_i et b_i) sont ignorées).
Dans ce cas, $h^{'}(R)$ est estimé par :

$$\mathcal{U}_1(x_R, y_R) = \min\Big\{F(L, W - y_R) + F(L - x_R, W), F(L, W) - g(R)\Big\}, \tag{6.1}$$

où F est la valeur de la solution de la fonction de sac-à-dos à deux dimensions de Gilmore et Gomory [61].
$\mathcal{U}_1(x_R, y_R)$ peut ne pas être très fine car l'équation 6.1 prend en considération deux fois le rectangle $(L - x_R, W - y_R)$.

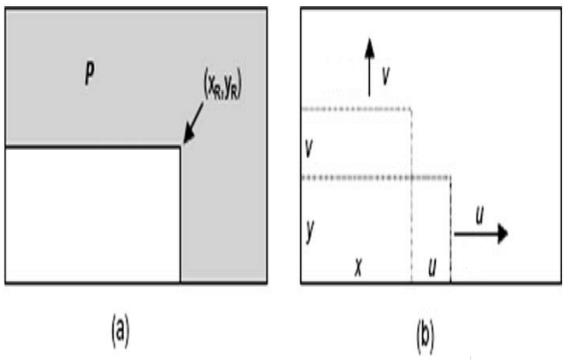


FIGURE 6.6 – La récursivité dans l'énumération de toutes les possibilités qui permettent à un sous-rectangle R de se placer dans le coin inférieur droit d'un plan de découpe guillotine.

6.3.3.2 Une deuxième borne supérieure

Une deuxième borne supérieure, $\mathcal{U}_2(x_R, y_R)$, est la valeur de la solution du problème de sac-à-dos borné unidimensionnel ($SBK_{(x_R,y_R)}$) suivant :

$$\mathcal{U}_2(x_R, y_R) = \max\Big\{ \sum_{i\in S_P^R} c_i z_i,\ \text{s.t.} \sum_{i\in S_P^R} (l_i w_i) z_i \leq (LW - x_R y_R),$$

$$z_i \leq \min\Big\{b_i, \lfloor \frac{LW - x_R y_R}{l_i w_i} \rfloor\Big\}, i \in S_P^R, z_i \in \mathcal{N}\Big\} \tag{6.2}$$

S_P^R est l'ensemble de pièces qui peuvent être placées dans la région P et z_i est le nombre d'occurrences de la pièce de type i dans P.
Au départ de l'algorithme exact, le problème $SBK_{(0,0)}$ est résolu par programmation dynamique, ainsi, les solutions de tous les $SBK_{(x,y)},\ x \leq L$ et $y \leq W$, comme celle de (x_R, y_R), sont disponibles au début de la résolution.

6.3.3.3 Amélioration de la borne supérieure courante

Nous avons amélioré la qualité des bornes $\mathcal{U}_1(x_R, y_R)$ et $\mathcal{U}_2(x_R, y_R)$ en utilisant deux méthodes :

(a) Application de l'algorithme récursive de Viswanathan et Bagchi's (voir Viswanathan et Bagchi [26]).

(B) Résolution d'un problème de sac-à-dos borné en utilisant une méthode gloutonne et une procédure polynômiale.

(a).Utilisation de la récursivité

La fonction récursive énumère toutes les possibilités qui permettent au sous-rectangle R de se placer dans le coin inférieur droit du plan de découpe guillotine, comme illustré dans la figure 6.6.(a).
nous appliquons la récursivité aux points qui se trouvent en dessous du grand rectangle S, lorsque nous avançons les points vers la droite ou en dessus de R comme illustré dans la figure 6.6.(b). Par conséquent, pour chaque sous-rectangle courant R, les solutions des problèmes UTDC et $SBK_{(...)}$ peuvent être combinées pour obtenir une borne supérieure plus fine.

$$\mathcal{U}_3(x_R, y_R) = \min\{\mathcal{H}(R), \mathcal{V}(R)\} \tag{6.3}$$

où

$$\mathcal{H}(R) = \max\{\mathcal{U}_3(x_R + u, y_R) + \min\Big\{\mathcal{U}_1(x_R + u, y_R), \mathcal{U}_2(u \times y_R)\Big\} \ : \ 1 \leq u \leq L - x_R\}, \tag{6.4}$$

$$\mathcal{V}(R) = \max\{\mathcal{U}_3(x_R, y_R + v) + \min\Big\{\mathcal{U}_1(x_R, y_R + v), \mathcal{U}_2(x_R \times v)\Big\} \ : \ 1 \leq v \leq W - y_R\}, \tag{6.5}$$

et $\mathcal{U}_3(L, W) = 0$.

(b).Utilisation d'un problème de sac-à-dos borné

L'équation 6.3 choisit la plus petite valeur de $\mathcal{U}_1(x_R, y_R)$ et $\mathcal{U}_2(x_R, y_R)$ obtenues respectivement de UTDC et SBK. Au lieu de résoudre $SBK_{(x_R, y_R)}$, on peut résoudre RSBK, une version relaxée de SBK, et utiliser sa solution $\mathcal{U}_4(x_R, y_R)$ comme une borne supérieure, comme suite :

$$\mathcal{U}_4(x_R, y_R) = \max\Big\{ \sum_{i \in S_P^R} c_i z_i, \text{ s.t. } \sum_{i \in S_P^R} (l_i w_i) z_i \leq T, \ a'_i \leq z_i \leq b'_i, \ z_i \geq 0, \ i \in S_P^R\Big\},$$

où les bornes sur les demande des pièces de type $i \in I$ sont relaxées à $(a'_i, b'_i) = (a_i - d_i^R, \min\{b_i, \lfloor \frac{T}{l_i w_i} \rfloor\})$.

La valeur T, dépend des pièces qui peuvent être placées dans P, elle est calculée comme suit :

$$T = \begin{cases} L(W - y_R) & \text{Si } \nexists \ k, \ k \in S_P^R \text{ tel que } l_k \leq L - x_R \\ W(L - x_R) & \text{Si } \nexists \ k, \ k \in S_P^R \text{ tel que } w_k \leq W - y_R \\ LW - x_R y_R & \text{Si } \exists \ (k \wedge k') \in S_P^R \text{ tel que} \\ & \qquad (w_k \leq W - y_R) \wedge (l_{k'} \leq L - x_R) \\ 0 & \text{sinon.} \end{cases} \tag{6.6}$$

La plus petite valeur entre $\mathcal{U}_3(x_R, y_R)$ et $\mathcal{U}_4(x_R, y_R)$, est une borne supérieure valide :

$$\mathcal{U}_5(x_R, y_R) = \min\Big\{\mathcal{U}_3(x_R, y_R), \mathcal{U}_4(x_R, y_R)\Big\}. \tag{6.7}$$

La comparaison de $\mathcal{U}_5(x_R, y_R)$ à $\mathcal{U}_1(x_R, y_R)$ avec une estimation basée sur $\mathcal{U}_2(x_R, y_R)$, donne une borne supérieure plus fine

$$\mathcal{U}_6(x_R, y_R) = \min\{\mathcal{U}_1(x_R, y_R), \mathcal{U}_2(L, W) - g(R), \mathcal{U}_5(x_R, y_R)\} \tag{6.8}$$

$$= \min\{\mathcal{U}_1(x_R, y_R), \mathcal{U}_2(L, W) - g(R), \mathcal{U}_3(x_R, y_R), \mathcal{U}_4(x_R, y_R)\}. \tag{6.9}$$

Afin d'accélérer le calcul de $\mathcal{U}_5(x_R, y_R)$, on utilise pendant le traitement de l'équation 6.6 un rectangle normalisé qui réduit la taille de T.

Par exemple, l'espace P de la figure 6.7.(a) peut être réduit à un espace normalisé $P1$ de la figure 6.7.(b) qui représente, l'union entre la bande horizontale $(L - x_R, W)$ et la bande verticale $(L, W - w_R)$.

L'espace normalisé $P1$ de la figure 6.7.(b) peut à son tour, être réduit à une bande normalisée de la figure 6.7.(c), lorsque la bande verticale est vide.

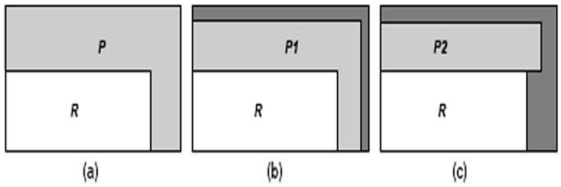


FIGURE 6.7 – **(a)** Estimation de l'espace P lorsque le rectangle R est placé au coin inférieur droit du rectangle initial S. **(b)** L'espace normalisé $P1$. **(c)** L'espace réduite $P2$ à l'application de l'équation 6.6.

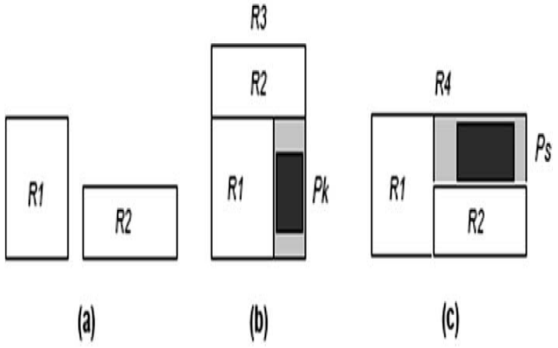


FIGURE 6.8 – Gestion des dominances entre plans de découpe pendant la construction.

6.3.4 Stratégie de développement

Les performances d'un algorithme "branch and bound" dépend très fortement de son implémentation. Utiliser des techniques avancées peut améliorer considérablement les performances. Ainsi, nous utilisons plusieurs techniques qui réduisent l'espace de recherche et troncaturent les plans de découpe (i) dominés (ii) dupliqués et (iii) non réalisables. Par ailleurs, nous utilisons des structures de données spécifiques pour les listes *Open* et *Clist*.

6.3.4.1 Gestion des dominances

On considère un plan $R1$ de la liste *Open* et un autre plan $R2$ de la liste *Clist*. $R3 = \frac{R1}{R2}$ est un plan dominé s'il existe une pièce k telle que $(l_k, w_k) \leq (l_{R3} - l_{R1}, w_{R1})$ (où $(l_k, w_k) \leq (l_{R3} - l_{R2}, w_{R2})$) et $b_k - d_k^{R3} \geq 1$, avec d_s^{R3} est le nombre d'occurrences de la pièce de type k dans $R3$.
De la même manière, $R4 = R1|R2$ est un plan dominé s'il existe une pièce s telle que $(l_s, w_s) \leq (l_{R2}, w_{R4} - w_{R2})$ (ou $(l_s, w_s) \leq (l_{R1}, w_{R4} - w_{R1})$) et $b_s - d_s^{R4} \geq 1$, avec d_s^{R4} le nombre d'occurrences de la pièce de type s dans $R4$.
Les plan $R3$ et $R4$ de la figure 6.8 sont des plans dominés.

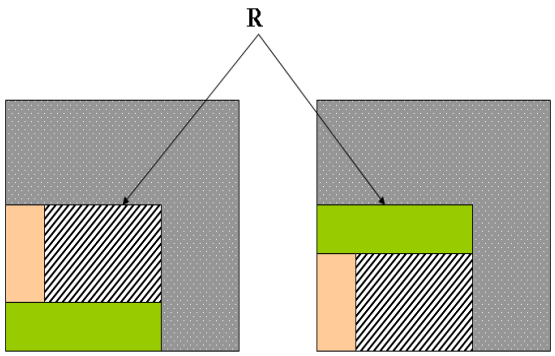


FIGURE 6.9 – Gestion des duplications des plans de découpe pendant la construction - Exemple de plans de découpe symétriques.

6.3.4.2 Gestion des duplications

Les plans de la figure 6.9 sont symétriques, ils ont la même taille et leur structure interne est identique (ils sont construit par les mêmes types de pièces), par conséquent, ils gérèrent le même profit. Dans une telle situation, ces deux plans sont dupliqués.
On peut détecter la duplication facilement comme suit :
Soit A un plan de la liste *Open* et B un autre plan de la liste *Clist*. La construction horizontale $A|B$ est une duplication d'un sous-rectangle interne s'il existe un autre sous-rectangle $C = (l_C, w_C)$ dans *Open* ou *Clist* tel que :

1. $(l_C, w_C) \leq (l_A + l_B, \max\{w_A, w_B\})$, et
2. $\forall i \in \{1, \ldots, n\},\ d_i^{A|B} \leq d_i^C$, où $d_i^{(\bullet)}$ le nombre d'occurrences de la pièce de type i dans le sous-rectangle interne courant.

6.3.4.3 Gestion de la non-réalisabilité

A chaque nœud de l'arborescence de recherche qui correspond à un sous-rectangle interne R de dimensions $x_R \times y_R$, un ensemble de bornes supérieures est calculé. Ces bornes servent à limiter l'espace de recherche et écarter les nœuds qui gérèrent des solutions non-réalisables. Parmi ces bornes, la borne $\mathcal{U}_4(x_R, y_R)$ est calculée par la résolution du problème $SKBK_{(x_R,y_R)}$.

Les deux propositions suivantes, sont utilisées pour écarter les nœuds non-réalisables lors de la combinaison en horizontal/vertical d'un élément de la liste *Open* et, un autre élément de la liste *Clist*.

Proposition 6.1. Une construction horizontale (resp. verticale) d'un plan A obtenue de *Open* et un plan B obtenue de *Clist* est ecartée si

$$\sum_{k \in P_R^P}^{n} l_k w_k a'_k > T,$$

où $a'_k = a_k - d_k^R$, et T est la capacité de $SKBK_{(x_R,y_R)}$ pour le rectangle courant R.

Preuve : Une construction horizontale de A et B produit un rectangle interne R de valeur $g(R)$ et une région occupée égale à $\sum_{k\in R} l_k w_k d_k^R$.

Soit S' l'ensemble de pièces tel que $a_i - d_i^R > 0$, et pour lequel T satisfait l'équation (6.6). Le plus petit espace utilisé par les pièces de S' est égale à $T' = \sum_{i\in S'} l_i w_i (a_i - d_i^R)$.
Si $T' \leq T$, chaque pièce de type $i \in S'$ satisfait sa contrainte de borne inférieure. Dans un autre sens, si $T' > T$, alors il existe un sous ensemble S'' tel que

$$T' = T + \sum_{k\in S''} l_k w_k (a_k - d_k^R).$$

Cela implique, qu'ils existent des pièces de type $k \in S''$ dont les contraintes de bornes inférieures ne sont pas satisfaites. Ainsi, le problème n'admet pas de solution réalisable, lorsque le plan R est construit. □

Proposition 6.2. Soit R une construction de deux plans A et B. R ne produit pas une solution réalisable pour le problème DCTDC si

$$\sum_{k\in I} l_k w_k a'_k > T.$$

Preuve : $d_k^R = 0$ si le type de pièce k ne satisfait pas l'équation 6.6. On distingue deux cas : (i) La capacité T est satisfaite par $BKSP$ et (ii) T n'est pas satisfaite.

Le cas(i) implique que l'ensemble S' qui représente les pièces qui peuvent remplir S_P^R satisfont les contraintes des bornes inférieures de ces pièces. S'il existe une pièce $t \notin S'$ tel que $a_t - d_t^R > 0$, alors sa contrainte de borne inférieure est violée, et la solution est non-réalisable.

Le cas (ii) implique que toutes les pièces de S' satisfont leurs contraintes de bornes supérieures respectives. Par conséquent, la solution construite par le positionnement de R dans le coin inférieur droit est non-réalisable si le problème $\overline{RSBK}_{(x_R,y_R)}$ ne sature pas la capacité $\overline{T} = T - \sum_{k\in S'} l_k w_k (a_k - d_k^R)$, où

$$(\overline{RSBK}_{(x_R,y_R)}) \begin{cases} \max & \sum_{i\in S''} l_i w_i z_i \\ \text{sc} & \sum_{i\in S''} (l_i w_i) z_i \leq \overline{T} \\ & z_i \leq a_i \\ & z_i \geq 0,\ i \in S'', \end{cases}$$

et S'' représente l'ensemble des pièces qui ne satisfont pas l'équation 6.6 et leurs $d_i^R = 0$.

Puisque, il existe au moins une pièce $t \in S''$ pour laquelle a_t est violée, alors la borne supérieure $g(R) + \mathcal{U}_4(x_R, y_R)$ est la valeur d'une solution non-réalisable du problème DCTDC. □

6.3.4.4 Structure de données

Afin d'améliorer les performances de l'algorithme, nous utilisons des structures de données particulières pour *Clist* qui représente les candidats à la solution optimale.

Nous ordonnons tous les plans de découpe (x_R, y_R) dans un ordre croissant de leurs largeurs et hauteurs. Cet ordre facilite la mise à jour de la liste Q qui contient toutes les constructions horizontales/verticales. Si R est l'élément courant sélectionné de la liste *Open* pour être déplacé vers *Clist*, alors l'algorithme construit les deux ensembles Q_{l_R} et Q_{w_R} qui contiennent les éléments candidats à être combinés avec R comme suit :

$$Q_{l_R} = \{q \mid l_q = l_R + l_p \leq L,\ p \in \text{Clist}\}$$

et

$$Q_{w_R} = \{q \mid w_q = w_R + w_p \leq W,\ p \in \text{Clist}\},$$

On remarque facilement que, $Q = Q_{l_R} \cup Q_{w_R}$.

6.4 Résultats numériques

Dans cette section, on évalue la performance des algorithmes proposés sur deux groupes d'instances. Nous avons mené les expérimentations en deux parties. Dans la première partie, on évalue la qualité des solutions approchées obtenues par GP et IGP. Dans la seconde partie, on évalue les performances de l'algorithme exact.
Toutes les approches ont été codées en C++ et testées sur un pentium centrino 2.8 GHZ et 512 Méga octets de mémoire centrale (Nous avons limité le temps exécution à 30 minutes).

6.4.1 Résultats du premier groupe d'instances

Le premier groupe d'instance, consiste en 24 instances de deux types :(i) 14 instances non-pondérées ($\forall i \in I,\ c_i = l_i \times w_i$), et 10 instances pondérées ($\exists i \in I,\ c_i \neq l_i \times w_i$).

6.4.1.1 Les instances

Les dimensions L et W du rectangle initial S ont été générées d'une manière aléatoire dans l'intervalle uniforme $[40, 150]$. Le nombre entier n qui correspond au nombre de pièces a été génère d'une manière aléatoire dans l'intervalle discret uniforme $[10, 30]$. Les dimensions l_i et w_i ont été générées respectivement d'une manière aléatoire dans les intervalles $[0.1L, 0.85L]$, et $[0.1W, 0.85W]$. Le profit c_i associé aux pièces est calculé par l'équation $c_i = \lceil \alpha l_i w_i \rceil$, où $\alpha = 1$ pour le cas non-pondéré. Par ailleurs, α a été génère aléatoirement dans l'intervalle $[0.25, 0.75]$ dans le cas pondéré.

Les bornes supérieures (respectivement inférieures) b_i (respectivement a_i), pour $i = 1, ..., n$, ont été calculées tel que $b_i = \min\{\alpha_1, \alpha_2\}$, où $\alpha_1 = \lfloor \frac{L}{l_i} \rfloor \lfloor \frac{W}{w_i} \rfloor$, α_2 a été génère aléatoirement dans l'intervalle entier $[a_i, 10]$,et a_i a été génère dans l'intervalle $[0, 4]$.

6.4.1.2 Performance de GP et IGP

Dans cette section, nous évaluons les performances de GP et IGP en comparant leurs solutions respectives $LB1$ et $LB2$ à la solution optimale Opt produite par l'algorithme exact.

La table 6.1 présente pour chaque instance, la solution optimale, Opt, $LB1$, $LB2$, un rapport d'approximation expérimentale $R_{LBi} = \frac{\text{LBi}}{Opt}$ avec $i = 1, 2$, le nombre de fois que la procédure de restauration de la réalisabilité est lancée. Res et le temps de résolution de IGP $T2$ mesuré en secondes. Le temps de résolution de GP n'est pas rapporté dans 6.1, car il est inférieur à 0.01 secondes. De la même manière, le temps nécessaire pour restaurer la réalisabilité n'est pas rapporté par 6.1 car il est de l'ordre de 0 secondes.

La qualité de la solution produite par GP est souvent loin de l'optimum. La colonne 3 de table 6.1 montre que le rapport moyen d'approximation est de 0.85 (resp. 0.93) pour les instances pondérées (resp. non-pondérées). Ce rapport est de 0.89 pour toutes les instances.

Le temps de résolution de GP est très court, il est inférieur à 0.01 secondes pour toutes les instances.
Le nombre de fois que la procédure de restauration de la réalisabilité est lancée, dépend généralement du nombre de combinaisons des hauteurs utilisées par GP ; la table 6.1 montre que cette procédure a été appliquée 168.58 fois en moyenne, pour toutes les instances.

Les colonnes 7 et 8 montrent que IGP améliore la qualité des solutions produites pas GP. La comparaison entre les colonnes 6 et 3 montre que le rapport d'approximation moyen passe de 0.89 à 0.95. Cette amélioration côute moins de 2 secondes en plus du temps de calcul en moyenne.

Le temps de résolution moyen des instances non-pondérées est plus important que les instances pondérées. Cela indique que GP et IGP se comportent mieux sur des problèmes de minimisation de chutes que sur les problèmes de maximisation de profits.

6.4.1.3 Performance de l'algorithme exact

Les performances de l'algorithme exact, en termes de nombre d'instances résolues et de temps de résolution, dépendent de la qualité de la solution de départ et de la stratégie de recherche implémentée.
On considère quatre versions de l'algorithme exact

1. La première version, ALGO1, utilise $\mathcal{U}_1(x_R, y_R)$, et $\mathcal{U}_2(x_R, y_R)$.
2. la deuxième version, ALGO2, utilise $\mathcal{U}_6(x_R, y_R)$, (La borne supérieure améliorée).
3. la troisième version, ALGO3, applique ALGO2 avec détection des dominances et des duplications.
4. La quatrième version, ALGO4, est une extension de ALGO3 qui inclut la gestion de la non-réalisabilité.

Algo1, comme illustré dans la table 6.2, résout 45.83% des instances. Son temps de résolution moyen est de 473 secondes. Le nombre de nœuds développés en moyenne est le plus grand des quatre versions (118485.73 nœuds).

Algo2, améliore la première version (Algo1) par la résolution de 66.67% des instances. Son temps de résolution moyen est de 333 secondes. Cela est dû principalement, au fait que Algo2

Instance				*Première borne inférieure*		*Deuxième borne inférieure*			
#Instance	a	b	Opt	LB1	R_{LB1}	LB2	R_{LB2}	Res	T_2
du1	7	22	14317	12934^{h}	0.90	$13490^{(h,v)}$	0.94	204	1.21
du2	3	34	14735	12404^{h}	0.84	$14145^{(h,v)}$	0.96	126	1.31
du3	6	35	14646	12563^{h}	0.86	$14010^{(h,v)}$	0.96	105	2.11
du4	4	45	7078	6664^{h}	0.94	$6932^{(h,v)}$	0.98	65	1.09
du5	5	40	14114	12047^{h}	0.85	$13478^{(h,v)}$	0.95	176	2.02
du6	10	46	15278	14647^{h}	0.96	$14647^{(h,v)}$	0.96	143	1.32
du7	12	46	15473	12146^{h}	0.78	$14064^{(h,v)}$	0.91	232	1.29
du8	13	49	5301	4404^{h}	0.83	$5301^{(h,v)}$	1.00	243	0.66
du9	7	48	12715	11474^{h}	0.90	$12340^{(h,h)}$	0.97	187	1.41
du10	7	49	14947	14016^{h}	0.94	$14602^{(h,v)}$	0.98	194	1.23
du11	6	71	15504	12454^{h}	0.80	$14785^{(h,v)}$	0.95	276	2.08
du12	12	71	18037	15029^{v}	0.83	$17049^{(v,h)}$	0.95	67	2.21
du13	6	65	20427	13801^{h}	0.68	$17353^{(h,v)}$	0.85	234	2.01
du14	11	69	16762	13596^{h}	0.81	$15058^{(h,v)}$	0.90	198	1.75
dw1	7	33	10296	9020^{h}	0.88	$9925^{(h,v)}$	0.96	201	1.1
dw2	5	34	10062	9219^{v}	0.92	$9442^{(v,v)}$	0.94	109	2.02
dw3	7	34	10312	9501^{v}	0.92	$9914^{(v,h)}$	0.96	232	1.54
dw4	4	33	8478	8314^{h}	0.98	$8384^{(v,h)}$	0.99	120	2.03
dw5	6	45	3660	3378^{v}	0.92	$3385^{(h,v)}$	0.92	43	2.01
dw6	9	55	8099	7524^{v}	0.93	$8028^{(h,v)}$	0.99	123	2.26
dw7	8	62	7171	6877^{v}	0.96	$7081^{(h,v)}$	0.99	143	2.15
dw8	7	52	9655	8675^{h}	0.90	$8739^{(h,v)}$	0.91	98	2.41
dw9	6	111	4550	4450^{h}	0.98	$4550^{(h,h)}$	1.00	105	2.34
dw10	8	111	8699	8360^{h}	0.96	$8414^{(h,v)}$	0.97	22	2.53
Av.N-P					0.85		0.95		1.55
Av.P					0.93		0.96		2.04
Av					0.89		0.95		1.75

TABLE 6.1 – Qualité des bornes inférieures produites par GP et IGP. Le symbole h (resp. v) signifie que la borne inférieure est obtenue par des bandes horizontales (resp. verticales). Le couple (h, v) signifie que la solution finale est obtenue en appliquant une construction horizontale complémentée par une construction verticale. La valeur a (resp. b) représente $\sum_{i=1}^{n} a_i$ (resp. $\sum_{i=1}^{n} b_i$).

développe moins de nœuds que Algo1 (100633.64 nœuds).

La détection des dominances et des duplications, améliore considérablement les performances. Ainsi, Algo3 résout pratiquement 3 fois plus d'instances qu'Algo1 et le double qu'Algo2. Il résout 83.33% des instances dans un temps réduit de 248 secondes.

L'implémentation de la gestion de la non-réalisabilité dans Algo4, permet de résoudre toutes les instances et de réduire le temps à 36 secondes.

Pour étudier l'impact de la solution de départ, nous avons lancé la quatrième version de l'algorithme exact (Algo4) (i) sans borne inférieure, (ii) avec la borne LB1 et (iii) avec la borne LB2.

La table 6.3 montre que Algo4 avec LB1 comme solution de départ est en moyenne plus performant que Algo4 sans solution de départ (4 instances non résolues contre 8 instances non résolues). L'utilisation de LB2 comme solution de départ, améliore considérablement les performances d'Algo4. Non seulement toutes les instances sont résolues, mais le temps de résolution a diminué de 226.9 secondes à 135.37.

6.4.2 Résultats du deuxième groupe d'instances

Dans cette section, nous évaluons les performances de IGP et de l'algorithme exact sur 40 instances de la littérature (Hadjiconstantinou et Christofides [58], Fekete et Schepers [77]). Nous

		Algo1		*Algo2*		*Algo3*		*Algo4*	
#Instance	Opt	*nœud*	*cpu*	*nœud*	*cpu*	*nœud*	*cpu*	*nœud*	*cpu*
du1	14317	199878	∘	185961	∘	159941	1304.17	16583	20.03
du2	14735	66843	∘	341241	∘	233401	∘	73153	420.66
du3	14646	76853	567.28	76853	567.28	76476	451.98	12474	7.14
du4	7078	342087	∘	231661	∘	101282	1204.64	66811	324.31
du5	14114	345085	∘	343191	∘	343088	∘	89125	1234.51
du6	15278	47710	157.45	29953	148.56	29953	148.56	6766	1.52
du7	15473	206809	473.72	206753	471.72	191386	468.55	36863	103.91
du8	5301	70292	83.72	67292	56.72	62192	47.72	12160	9.19
du9	12715	340219	∘	345256	∘	105439	1345.32	40350	124.48
du10	14947	341857	∘	343887	∘	343522	∘	43573	157.24
du11	15504	145141	∘	145121	1420.18	99580	98.12	28247	53.09
du12	18037	131642	∘	131642	1675.73	118529	104.16	26075	43.45
du13	20427	230205	∘	220205	1100.34	219322	556.5	52253	192.41
du14	16762	271554	920.24	253718	801.98	267262	326.08	50045	173.45
dw1	10296	185808	733.14	182957	643.95	175735	545.36	30339	79.08
dw2	10062	6677	13.31	4626	8.67	2540	3.67	456	0.14
dw3	10312	345790	∘	330864	∘	230547	1022.28	24154	44.98
dw4	8478	88624	295.38	48050	183.44	29618	84.56	6481	2.12
dw5	3660	161704	∘	121070	1127.06	119577	1022.91	32008	149.72
dw6	8099	78402	275.28	75246	274.5	50728	165.69	5921	2.61
dw7	7171	110448	∘	187198	∘	189056	∘	28168	68.09
dw8	9655	53027	1237.14	51564	241	50484	238.31	5186	2.25
dw9	4550	349237	∘	78095	395.26	23165	135.94	8593	13.56
dw10	8699	217587	445.58	109958	265.72	87579	247.3	17284	20.95
Av.P		118485.73	472.93	100633.64	333.05	93086.64	247.98	16725.00	36.58
Av.N-P		105020.83	499.97	78733.50	269.55	66114.00	214.15	10944.50	17.86
Av		118485.73	472.93	100633.64	333.05	93086.64	247.98	16725.00	36.58

TABLE 6.2 – Impact de la stratégie d'implémentation sur l'approche exacte. Le symbole ∘ signifie que l'algorithme ne trouve pas la solution optimale après 30 minutes d'exécution.

		Algo4-Sans LB		Algo4-Avec LB1		Algo4-Avec LB2	
#Instance	Opt	nœud	cpu	nœud	cpu	nœud	cpu
du1	14317	49457	376.75	25471	145.56	16583	20.03
du2	14735	10017	∘	10009	∘	73153	420.66
du3	14646	43876	165.43	12599	54.57	12474	7.14
du4	7078	117412	∘	81939	∘	66811	324.31
du5	14114	160032	∘	161957	∘	89125	1234.51
du6	15278	11028	19.47	7864	8.76	6766	1.52
du7	15473	40120	492.20	40110	463.32	36863	103.91
du8	5301	16160	44.76	16160	40.76	12160	9.19
du9	12715	96520	∘	65109	1202.76	40350	124.48
du10	14947	130883	∘	101481	∘	43573	157.24
du11	15504	46210	404.51	36209	346.08	28247	53.09
du12	18037	38876	276.78	28280	202.00	26075	43.45
du13	20427	75432	987.61	52289	775.80	52253	192.41
du14	16762	90063	921.54	50738	721.12	50045	173.45
dw1	10296	62348	427.86	41791	388.65	30339	79.08
dw2	10062	15808	24.95	7534	9.32	456	0.14
dw3	10312	89785	∘	44914	654.64	24154	44.98
dw4	8478	16523	66.80	9090	12.68	6481	2.12
dw5	3660	103505	∘	42368	706.76	32008	149.72
dw6	8099	11215	22.64	10631	20.36	5921	2.61
dw7	7171	40278	642.92	29315	294.32	28168	68.09
dw8	9655	15284	19.65	8223	12.34	5186	2.25
dw9	4550	183978	∘	25243	294.32	8593	13.56
dw10	8699	37789	453.20	21121	134.76	17284	20.95
Av.P		45691.33	409.89	29968.89	306.44	26829.56	67.13
Av.N-P		28463.57	236.86	18243.57	124.63	13405.00	25.03
Av		38154.19	334.19	24839.06	226.90	20956.31	48.71

TABLE 6.3 – Impact de la solution initiale sur les performances des quatre versions de l'algorithme exact.

avons, en premier lieu, adapté ces instances, conçues initialement pour le problème de découpe à deux dimensions orthogonale (ce problème ne tient pas compte de la contrainte guillotine) pour

CDTDC. Puis, nous avons évalué les performances des algorithmes proposés sur les instances adaptées.

6.4.2.1 Les instances

Ce groupe est composé de douze instances du type *gcut* (gcut1,. . . , gcut12), six instances du type *hadchr* (extraites de (Hadjiconstantinou et Christofides [58], sept instances du type *ngcutcon* (ngcutcon11,. . . , ngcutcon17), dix instances de type *ngcutfs* et cinq instances du type *okp* (extraites de Fekete et Schepers [77].

6.4.2.2 Étude expérimentale

Évidemment, comme ces instances ne sont pas conçues initialement pour DCTDC, leurs solutions optimales pour ce problème ne sont pas connues.
Cependant, nous avons résolu ces instances avec $a_i = 0, i \in I$, et nous avons comparé, pour chaque instance, la solution optimale obtenue OptCG avec la solution du problème général OptCNG (sans tenir compte de la contrainte guillotine). Pour les instances dont OptCNG n'est pas connue, nous avons calculé une borne supérieure en utilisant un problème de sac-à-dos (voir chapitre 3). La comparaison de OptCG et OptCNG indique l'influence de la contrainte guillotine sur la solution optimale de chaque instance.

En second lieu, nous avons lancé l'algorithme exact sur chaque instance avec a_i généré aléatoirement dans l'intervalle $[0, b_i], i \in I$. La comparaison entre la solution optimale obtenue dans ce cas OptCGD à OptCG, permet de mesurer l'effet de la contrainte des bornes inférieures des pièces a_i sur la valeur de la solution optimale.

Finalement, nous avons évalué l'effet de la solution de départ sur les performances de l'algorithme exact en termes de temps de calcul et de nombre de nœuds développés. Nous avons évalué la qualité de la solution obtenue par IGP en la comparant à OptCGD et enfin, nous avons lancé l'algorithme Algo4 avec et sans la solution obtenue par IGP, dans le but de comparer leurs performances.

6.4.2.3 Résultats

Les colonnes $2-4$ de la table 6.4 donnent OptCNG, OptCG et le pourcentage de l'écart entre OptCG et OptCNG. Dans 14 cas sur 40, $OptCG = OptCHG$, cela indique que dans ces 14 cas, la structure d'une des solutions optimales du problème générale, respecte la contrainte guillotine. Cependant, l'écart moyen entre OptCG et OptCNG pour toutes les instances est de 2.45. Cela indique, que la contrainte guillotine réduit la valeur de la solution optimale. En effet, $OptCG \leq OptCHG$ pour toutes les instances.

La colonne 5 de la table 6.4 montre OptCGD obtenue après l'introduction de la contrainte des bornes inférieures, alors que la colonne 6 donne le pourcentage de l'écart entre la solution que nous obtenons par apport à OptCGD (6.17% en moyenne). Ce pourcentage monte à 25.90% pour hadchr7 et descend à 0 pour les instances hadchr2, ngcutcon13 et okp4. Pour ces trois instances, la

			%Dev			%Dev				*Algorithme* Algo4					
										Sans LB2		Avec LB2			
#Instance	OptCNG	OptCG	NG vs G	OptCGD	G vs GD	*LB2*	Res	*cpu*	*nœud*	*cpu*	*nœud*	*cpu*
gcut1	48368	48368	0.00	40625	16.01	40625	44	0.45	50	0.80	22	0.68
gcut2	59798	59307	0.82	57996	2.21	54450	53	0.86	558	2.01	172	1.82
gcut3	61275	60241	1.69	58677	2.60	58438	128	1.23	1784	3.91	485	2.94
gcut4	61380	60942	0.71	58866	3.41	58004	150	2.54	7509	28.35	1973	9.47
gcut5	195582	195582	0.00	183339	6.26	167594	218	1.43	435	3.60	89	2.86
gcut6	236305	236305	0.00	189391	19.85	178443	354	1.65	377	7.20	302	6.80
gcut7	240143	238974	0.49	217082	9.16	201445	443	2.76	741	10.64	483	9.70
gcut8	245758	245758	0.00	239195	2.67	232764	362	1.04	7292	37.40	2254	19.97
gcut9	939600	919476	2.14	874589	4.88	858697	492	2.02	124	13.90	77	14.20
gcut10	937349	903435	3.62	790985	12.45	774901	275	3.05	787	24.02	220	21.70
gcut11	969709	955389	1.48	919069	3.80	890177	301	3.32	2247	37.80	973	32.50
gcut12	979521	970744	0.90	869700	10.41	861170	295	3.67	7298	46.98	2564	36.72
hadchr2	430	430	0.00	430	0.00	430	19	0.01	34	0.01	5	0.01
hadchr3	1178	1178	0.00	963	18.25	914	18	0.01	77	0.05	21	0.02
hadchr5	289	289	0.00	275	4.84	256	29	0.01	425	0.08	307	0.05
hadchr6	924	924	0.00	732	20.78	720	34	0.01	1099	0.53	1049	0.49
hadchr7	1865	1865	0.00	1382	25.90	1382	56	0.01	1233	0.41	1142	0.39
hadchr11	1270	1270	0.00	1188	6.46	1147	27	0.02	863	0.46	751	0.37
ngcutcon11	1864	1688	9.44	1518	10.07	1518	41	0.01	530	0.11	227	0.08
ngcutcon12	2012	1865	7.31	1672	10.35	1672	65	0.05	13331	20.81	3456	7.85
ngcutcon13	1347	1178	12.55	1178	0.00	1178	17	0.01	94	0.09	26	0.05
ngcutcon14	1547	1270	17.91	1216	4.25	1200	28	0.01	742	0.26	489	0.20
ngcutcon15	2800	2721	2.82	2700	0.77	2614	6	0.01	429	0.15	89	0.06
ngcutcon16	2020	1860	7.92	1700	8.60	1620	34	0.53	5432	13.70	2179	5.10
ngcutcon17	29133	27589	5.30	25377	8.02	23609	81	3.24	84264	○	43836	823.52
ngcutfs1.1	29955	27968	6.63	26542	5.10	25204	123	0.85	12355	85.70	7053	34.60
ngcutfs1.2	30000	29181	2.73	28807	1.28	27377	6	1.04	9500	42.80	6965	36.80
ngcutfs1.3	30000	29865	0.45	29830	0.12	28754	110	1.65	7845	7.70	3601	4.90
ngcutfs1.4	30000	29964	0.12	29628	1.12	29361	15	2.41	98181	○	29864	956.70
ngcutfs1.5	30000	29961	0.13	29891	0.23	29454	104	3.65	107235	1785.98	76984	895.06
ngcutfs2.1	29973	27659	7.72	26731	3.36	25324	98	1.11	86364	○	59034	1094.56
ngcutfs2.2	30000	28984	3.39	28327	2.27	28093	123	1.21	61066	○	33379	807.15
ngcutfs2.3	30000	29847	0.51	29325	1.75	29100	176	1.89	28206	336.25	7579	85.74
ngcutfs2.4	30000	29833	0.56	29122	2.38	27595	282	3.01	98098	○	46366	1320.35
ngcutfs2.5	30000	29970	0.10	29535	1.45	29408	0	4.56	74984	○	51195	1333.68
okp1	27718	27589	0.47	26372	4.41	25963	0	0.09	11775	34.04	5731	16.90
okp2	22502	22502	0.00	21916	2.60	20641	0	1.31	19188	247.64	11776	144.22
okp3	24019	24019	0.00	23740	1.16	23655	75	0.85	12354	82.65	8984	61.52
okp4	32893	32893	0.00	32893	0.00	30554	124	1.13	123453	987.83	76876	443.76
okp5	27923	27923	0.00	25835	7.48	25298	107	1.54	14371	79.83	6557	19.70
Av			2.45		6.17			1.36	22568.25	115.99	12378.38	56.39

TABLE 6.4 – Étude comparative entre la qualité de la solution guillotine et non-guillotine de la dernière version de l'algorithme exact avec et sans solution de départ.

structure de la solution qui correspond à OptCG satisfait les contraintes sur les bornes inférieures des pièces.

La colonne 7 de la table 6.4 montre LB2 qui correspond aux solutions obtenues par IGP. La colonne 8 montre le nombre de fois où la procédure de restauration de la réalisabilité a été lancée. La comparaison des solutions obtenues par IGP par rapport à OptCGD montre que le pourcentage de l'écart moyen est de 2.97%. Cependant cet écart monte à 8.59% pour gcut5 et à 7.2% pour gcut7. Pour certaines instances, cet écart est égal à 0%. L'écart moyen de IGP reste très réduit et son temps de résolution est satisfaisant (1.36 secondes en moyenne ; le plus long temps observé est

de 4.56 secondes) ce qui justifie son utilisation comme borne inférieure.

Les colonnes $10-11$ (Resp.$12-13$) montrent le nombre de nœuds développés et le temps de résolution de Algo4 sans (Resp. avec) la solution de départ obtenue par IGP. Lorsque LB2 n'est pas utilisé, Algo4 ne trouve pas l'optimum de 6 instances sur les 40 traitées. Par contre, Algo4 arrive à résoudre toutes les instances en utilisant LB2. De plus, le nombre de nœuds développés par Algo4 diminue considérablement (de 33.31%), lorsque LB2 est utilisée, ainsi que le temps de résolution. L'explication est simple ; l'utilisation de LB2 permet de troncaturer des nœuds inutiles et de réduire, ainsi, l'espace de recherche. En effet, On remarque que pour les instances résolues pendant les 30 minutes, l'utilisation de LB2 a permis de réduire le nombre de nœuds développés (de 22568.25 à 12378.38 nœuds en moyenne) et d'accélérer la résolution (de 115.95 à 56.39 secondes en moyenne). Cela, démontre l'importance et l'intérêt de la solution de départ dans l'algorithme exact.

6.5 Conclusion

Dans ce chapitre, nous avons étudié le problème de découpe double contraint à deux dimensions. Dans ce problème chaque type de pièce est limité par deux contraintes ; une borne supérieure qui limite le nombre d'apparitions de la pièce dans le plan de découpe réalisable final et une autre borne inférieure, qui impose un nombre d'apparitions de la pièce dans le même plan de découpe.
Nous avons résolu ce problème par un algorithme de séparation et évaluation et des nouvelles bornes supérieures et inférieures efficaces. La borne inférieure est obtenue, par relaxation du problème étudié, en un problème de découpe contraint à deux dimensions. Par ailleurs, la borne supérieure est obtenue par la résolution de plusieurs variétés de problèmes de sac-à-dos.
L'approche proposée résout des problèmes de petites et de moyennes tailles dans un temps très raisonnable. Ces bonnes performances sont dûes à l'implémentation de structures de données adaptées, d'une gestion des duplications, des dominances et de la non-réalisabilité. En effet, ces techniques permettent de réduire considérablement l'espace de recherche et d'améliorer, ainsi, les performances de l'algorithme.

Quatrième partie

RÉSOLUTION PARALLÈLE APPROCHÉE

Chapitre 7

Parallélisme et problèmes combinatoires

Contenu

Les applications informatiques industrielles, nécessitent de plus en plus de puissance de calcul. Pour obtenir des niveaux de performance élevés, on peut construire des systèmes informatiques dans lesquels plusieurs unités de calcul coopèrent pour résoudre le même problème. Cette solution permet d'avancer plus vite dans la résolution, mais aussi de travailler sur des problèmes de plus grande taille. L'évolution des machines parallèles a suivi l'évolution des processeurs et des réseaux. Des processeurs de plus en plus puissants sont apparus et ont permis de construire des stations de travail avec des puissances équivalentes à celles des premiers supercalculateurs et, l'apparition des réseaux à haut débit a permis d'interconnecter des dizaines, voire des centaines d'ordinateurs qui échangent des informations à vitesse très élevée, ce qui a permis de voir une nouvelle génération de machines parallèles a faible coût qui utilisent des composants standards tant au niveau des systèmes d'exploitation que du réseau d'interconnexion et des processeurs.

Dans ce chapitre, nous présentons quelques définitions et notions préliminaires du parallélisme et tout particulièrement l'utilisation du parallélisme dans la résolution de problèmes combinatoires. Nous commençons dans la section 7.1 par donner quelques notions sur les architectures des machines parallèles. Nous présentons ensuite, dans la section 7.2, quelques approches parallèles et quelques mesures de performances d'algorithmes ou d'approches parallèles dans la section 7.3. Dans la section 7.4, on présente quelques notions et travaux liés aux algorithmes branch and bound parallèles. On termine ce chapitre, par la section 7.5, qui présente deux bibliothèques de parallélisme, très utilisées dans la programmation parallèle.

7.1 Architecture des machines parallèles

Les ordinateurs parallèles sont construits autour d'un ensemble de plusieurs nœuds interconnectés. Chaque nœud correspond à un ou plusieurs processeurs de calcul qui peut ou peuvent communiquer avec les autres processeurs. Le processeur est relié à un espace mémoire qui peut être partagé avec les autres nœuds. Les machines parallèles ont été classées selon leur mode de fonctionnement. Flynn [53] distingue quatre types de machines selon le nombre de flot d'instructions et de données :

- Les machines SISD (Single Instruction Single Data) : Une seule instruction s'exécute sur une seule donnée à la fois. Ce type d'architecture correspond à une machine monoprocesseur.
- Les machines SIMD (Single Instruction Multiple Data) : Les processeurs exécutent la même instruction sur des données différentes. Ce type de machine est utilisé pour des calculs spécialisés. Elles peuvent contenir plusieurs centaines de processeurs grâce à la simplicité de leur architecture.
- Les machines MISD (Multiple Instruction Single Data) : plusieurs instructions s'exécutent sur une seule donnée.
- Les machines MIMD (Multiple Instruction Single Data) : Ce sont les machines les plus courantes aujourd'hui. Ce type d'architecture représente un système de processeurs autonomes où chacun exécute son propre flot d'instructions sur des flots de données différentes.

Ducan [69] a étendu cette classification selon le mode de synchronisation des nœuds :

- Les machines synchrones : Une horloge globale synchronise les différents nœuds du système.
- Les machines asynchrones : Chaque machine est autonome et possède une horloge locale qui lui est propre.

Les machines SIMD sont synchrones par définition. La majorité des machines MIMD sont asynchrones.

Un autre point qui différencie les ordinateurs parallèles, est l'organisation de la mémoire. Elle peut être partagée par tous les processeurs ou distribuée.

7.1.1 Les architectures à mémoire partagée

Les processeurs écrivent et lisent dans le même espace d'adressage mémoire (figure 7.1.a). Toute opération effectuée sur la mémoire par un processeur est immédiatement visible par l'ensemble des autres processeurs.

Cependant, l'accès à la mémoire constitue un goulot d'étranglement sur ce type de machine, si le nombre de processeurs devient important. Ce nombre se limite à une dizaine de processeurs sur les ordinateurs à mémoire partagée.

7.1.2 Les architectures à mémoire distribuée

Chaque processeur possède sa propre zone mémoire. Les différents nœuds de calcul sont reliés entre eux par un réseau d'interconnexion (figure 7.1.b). Les nœuds de calcul communiquent entre eux par des échanges de messages à travers le réseau de communication. Le temps de communication d'un message peut être modélisé par une fonction de la forme :

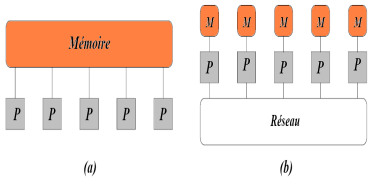


FIGURE 7.1 – (a) Machine à mémoire partagée (b) Machine à mémoire distribuée.

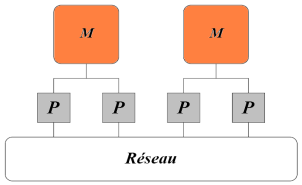


FIGURE 7.2 – Architecture mixte.

Latence + taille du message/bande passante

où la latence de communication est le temps de démarrage de la communication. La bande passante du réseau est la quantité de données qui peut transiter sur un lien par unité de temps.

Les constructeurs ont essayé plusieurs topologies de réseau pour connecter les nœuds de calcul : hypercube pour le CM2 et le iPSC, grilles pour le Paragon, tore pour le Cray T3E, fat-tree pour le CM5, réseau multi-étages pour le SP4...etc

Dans les anciennes générations de machines, la topologie du réseau est un élément critique qu'il faut prendre en considération lors de la programmation d'une application. La latence de communication entre deux nœuds, peut fortement varier selon qu'ils soient voisins ou non. Il est donc, à la charge du programmeur, de découper et de placer ses données de façon, à ne pas dégrader les performances espérées par la parallélisation de l'application à cause des performances du réseau d'interconnexion.

La nouvelle génération de machines MIMD utilise des réseaux d'interconnexion qui assurent des latences de communication constantes entre processeurs. La plupart des machines utilisent des modes de routage de type dérivés du wormhole. Dans ce mode, acheminer un message consiste à établir un chemin dans le réseau entre le nœud source et le nœud destination. Le message est ensuite découpé en petits paquets qui sont envoyés les uns à la suite des autres sur le chemin fixé. Ce mode de routage rend les communications peu dépendantes de la topologie du réseau.

Pour améliorer les performances de communications certaines machines offrent la possibilité de recouvrir les calculs et les communications comme dans le IBM SP4. Les nœuds de calculs des machines possèdent un microprocesseur spécialisé dans la communication qui accède directement à la mémoire du nœud. Il existe aujourd'hui des machines avec des architectures mixtes : des machines à mémoire distribuée dont les nœuds de calculs sont des ordinateurs parallèles à mémoire partagée (figure 7.2). La machine SGI-Origin 2000 en est un exemple. Ce type d'architecture est amené à se développer car il permet d'exploiter plusieurs grains de parallélisme. Notons aussi le

développement des machines à mémoire partagée virtuelle, ainsi que la multiplication des réseaux de stations, que l'on appelle aussi grappe de stations, et qui présentent l'avantage d'être flexibles, modulaires et d'un coût économique relativement peu élevé.

7.2 Approche de parallélisation

Dans la section précédente nous avons rappelé les architectures des machines parallèles. Dans cette section nous présenterons les différentes approches de parallélisation d'une application, ainsi que les problèmes qui peuvent se poser.

Il n'existe pas d'approche type lors de la parallélisation d'une application. Ceci dépend fortement de la nature du problème. Il est souvent nécessaire de construire un algorithme parallèle très différent de l'algorithme séquentiel. Pendant la procédure de parallélisation, il faut identifier les tâches qui peuvent s'exécuter en parallèle. Il faut ensuite choisir l'ordre d'exécution de chacune des tâches et construire un graphe de dépendance (graphe orienté sans cycle de dépendance entre les tâches).

7.2.1 Parallélisme de données

Il consiste à effectuer la même opération en parallèle sur un ensemble de données réparties sur les processeurs. Le programme parallèle se résume en une succession de phases de calcul et de phases de communication. La phase de communication permet de mettre à jour des données distantes ou à redistribuer des données pour améliorer l'équilibrage de charge entre les processeurs.

7.2.2 Parallélisme de contrôle

Le parallélisme de contrôle ou fonctionnel consiste à découper une application en plusieurs tâches qui peuvent s'exécuter en parallèle. L'algorithme parallèle peut être décrit sous la forme d'un graphe orienté sans cycle de dépendance entre les tâches : une tâche ne peut être effectuée qu'après la fin de l'exécution de toutes les tâches qui la précèdent dans le graphe de dépendance. Dans ce paradigme de parallélisme, ce sont les traitements qui sont placés sur un processeur. Les données sont utlilisées à distance ou déplacées en tant que paramètre du traitement.

7.2.3 Grain de parallélisme

Le grain de parallélisme peut être défini comme la taille des tâches de calcul sur un nœud. Le choix de la granularité est un point essentiel pendant la description d'un algorithme parallèle. Dans le parallélisme de données, le grain de parallélisme est déterminé par la finesse de la découpe des données. Dans le parallélisme de contrôle le grain de parallélisme est déterminé par le rapport entre le temps d'exécution des tâches et la largeur du graphe des tâches, c'est à dire le nombre maximum de tâches qui peuvent s'exécuter en parallèle. On augmente le grain de parallélisme en regroupant plusieurs tâches. On obtient alors un parallélisme de gros grain. Puis on diminue la granularité en découpant les tâches en sous-tâches plus petites. On obtient alors un parallélisme de grain fin. Le choix de granularité peut être fait en fonction des caractéristiques d'une machine cible particulière. Par exemple, les machines parallèles à mémoire partagée sont bien adaptées à un

parallélisme de grain fin grâce à la localité des données, mais elles ne peuvent entièrement l'exploiter à cause du nombre limité de processeurs. Pour les machines à mémoire distribuée, c'est la latence de communication et la bande passante de communication qui influent sur le grain de parallélisme souhaitable. Ainsi un grain trop fin peut entraîner trop de communication entre les processeurs. Alors qu'un parallélisme de gros grain rend les calculs trop longs devant des communications négligeables et diminue la concurrence de calcul entre les processeurs ce qui entraîne un déséquilibre de charge entre les processeurs.

Un parallélisme efficace doit faire un compromis entre le grain de parallélisme et le surcoût de la gestion du parallélisme qu'il induit. Il est, donc, parfois utile de pouvoir contrôler dynamiquement le grain de parallélisme durant l'exécution.

7.2.4 Les applications parallèles irrégulières

On peut classer les applications parallèles selon leurs comportement. Une application parallèle est dite régulière si on peut prédire à l'avance le graphe des tâches et leurs durées. En revanche, une application parallèle est dite irrégulière si on ne peut pas prédire le comportement de l'application indépendamment de l'instance du problème qu'on cherche à résoudre. On peut avoir plusieurs causes de cette irrégularité :

- La durée de calcul dans une tâche n'est pas connue et varie fortement d'une tâche à l'autre.
- On ne connait pas précisément les dépendances entre les tâches. Il est impossible de construire le graphe des tâches a priori.
- La taille des données manipulées lors des communications varie fortement ou n'est pas connue d'avance.

La parallélisation des applications irrégulières se heurte à la difficulté de prévoir même partiellement le graphe des tâches et l'estimation des coûts de calcul d'une tâche en fonction de ses entrées. Ainsi, il est difficile de bien choisir d'une part le grain de parallélisme afin d'avoir un bon équilibrage de charge, et d'autre part le placement optimal des tâches afin de minimiser les communications. Les applications irrégulières peuvent provenir de plusieurs domaines tels que l'optimisation combinatoire, la dynamique moléculaire, la météorologie...etc.

7.2.5 Ordonnancement

l'ordonnancement des tâches consiste à attribuer une date de début d'exécution à chaque tâche. l'ordonnancement dépend du nombre de processeurs utilisés et du graphe de précédence des tâches. Pour les applications régulières, il est facile d'ordonnancer les tâches puisque le graphe de précédence est connu à l'avance. Pour les applications irrégulières le graphe de précédence est connu au cours de l'exécution. On ordonnance les tâches au fur et à mesure de leur création. On utilise donc des méthodes d'ordonnancement en ligne plus compliquées à mettre en oeuvre.

7.2.6 Régulation de charge

L'un des points importants de la parallélisation d'une application est le choix d'un bon placement des tâches sur les processeurs minimisant les communications. Ainsi on place les tâches de manière à ce que l'accès aux données soit le plus local possible pour ne pas surcharger le réseau de

communication. On place en général sur le même processeur les tâches qui ont beaucoup de données à échanger. De plus, afin de maximiser la concurrence et donc de mieux équilibrer la charge de calcul entre les processeurs, le placement doit garder le plus de tâches actives simultanément. En pratique, le placement et l'ordonnancement de tâches s'effectuent au même moment. Pour les applications régulières, on utilise des algorithmes d'ordonnancement et de placement statique. Le problème de placement peut être défini comme un problème combinatoire sur le graphe des tâches où les coûts des sommets et des arcs sont définis comme étant égaux respectivement à la durée de l'exécution de la tâche et au temps de communication des données. Pour les applications irrégulières, on utilise des algorithmes de placement dynamique spécialement adaptés au problème. Des mécanismes de régulation de charge dynamique sont aussi intégrés à l'application. Ces algorithmes comportent plusieurs phases :

- L'évaluation de la charge de chaque processeur. Elle est en général très difficile à définir pour les applications irrégulières, puisqu'on ne connaît pas à l'avance le coût d'une tâche.
- La détermination de la surcharge ou de la sous-charge d'un processeur.
- La sélection des tâches à migrer d'un processeur à un autre.
- La migration de tâches sélectionnées.

Un bon algorithme de régulation de charge ne doit pas augmenter considérablement les communications dues à la migration des tâches, et ne doit pas consommer beaucoup de temps de calcul durant les différentes étapes d'évaluation de la charge.

7.3 Mesure des performances

L'évaluation de performance d'un algorithme parallèle est importante pour connaître le gain obtenu par la parallélisation par rapport à l'algorithme séquentiel.

7.3.1 Accélération

l'accélération (Speed Up) est définie comme le rapport du temps d'exécution séquentiel T_{seq} sur le temps de l'exécution parallèle T_p sur p processeurs :

$$S(p) = \frac{T_{seq}}{T_p}$$

l'accélération idéale est égale à p. Si on considère la fonction accélération $S(p)$, l'algorithme parallèle est d'autant plus performant que la fonction $S(p)$ est proche de la droite $y = p$. On parlera dans ce cas d'accélération linéaire. Pour avoir de bonnes accélérations, il faut que le surcoût de la parallélisation en temps de calcul et surtout en volume de communication soit minimum. Il faut s'appliquer aussi à minimiser les temps d'inactivité des processeurs dûs à une mauvaise répartition de charge.
Notons qu'on peut introduire la notion d'accélération relative, par opposition à l'accélération absolue, qui ne prend pas en compte dans T_{seq} les parties intrinsèquement séquentielles de l'algorithme.

7.3.2 Efficacité

l'efficacité d'un algorithme parallèle est définie comme le rapport de l'accélération sur le nombre de processeurs.

$$E_p = \frac{S_p}{p} = \frac{T_{seq}}{p_{T_p}}$$

L'efficacité représente le rendement de l'algorithme parallèle et elle est souvent exprimée par un pourcentage qui représente l'utilisation moyenne des processeurs par rapport à une utilisation permanente durant toute la durée de l'exécution T_p. Pour approfondir toutes ces notions, le lecteur est encouragé à consulter [31] et [11].

7.4 Parallélisation du Branch and Bound

Dans cette section nous présentons un aperçu non exhaustif des études de parallélisation de l'algorithme du Branch and Bound dédié à la résolution des problèmes d'optimisation combinatoire. Nous rapporterons quelques exemples d'expérimentations et de plateformes de programmation de Branch and Bound parallèle.

7.4.1 L'algorithme du Branch and Bound

La méthode du Branch and Bound est l'un des plus puissants algorithmes pour résoudre des problèmes NP-difficiles. Pour la plupart de ces problèmes seulement des petites instances peuvent être résolues en un temps raisonnable sur des ordinateurs séquentiels. La parallélisation de cette méthode pour diminuer les temps de calcul a retenu l'attention de nombreux chercheurs dans les dix dernières années, en particulier pour la résolution de problèmes d'optimisation combinatoire. L'objectif de cette section est de donner un aperçu général de ces recherches.

Rappellons que la méthode du Branch and Bound consiste en une énumération implicite de l'ensemble des solutions d'un problème d'optimisation P : $min(max)_{x \in S} c(x)$, où $c(x)$ est une fonction coût et S est l'espace des solutions réalisables de P. Le principe de l'algorithme est de diviser l'ensemble des solutions en sous-ensembles. Cette opération s'appelle branchement. Elle consiste à construire un arbre $T = (N; E)$ où N représente l'ensemble des nœuds de l'arbre qui représentent les sous-ensembles obtenus par branchement. Un nœud est alors, sélectionné de la file des nœuds actifs puis évalué. La valeur du nœud représente une borne inférieure de la valeur de la solution optimale. Un nœud peut être ignoré, si son évaluation est supérieure à la meilleure solution connue, sinon le nœud est ajouté à la file des nœuds actifs. La stratégie de sélection du nœud définit la manière de parcourir l'arbre du Branch and Bound. Pour une bonne compréhension des phénomènes liés aux performances de l'algorithme du Branch and Bound, nous allons présenter les définitions de Mans et Roucairol dans [5] des différents sous-ensembles de nœuds inclus dans N :

C est l'ensemble de nœuds d'évaluation, inférieures à la valeur de la solution optimale.
I est l'ensemble des nœuds d'évaluation égales à la valeur de la solution optimale mais qui ne sont pas réalisables.
E est l'ensemble des nœuds évalués, supérieures à la solution optimale(à trancaturer).
O est l'ensemble des nœuds optimaux.

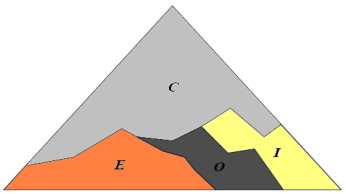


FIGURE 7.3 – Structure de l'arbre complet d'énumération du Branch and Bound.

7.4.2 Approche de parallélisation

On peut classer les algorithmes parallèles de Branch and Bound selon le grain de parallélisme utilisé. Il existe trois stratégies.

1. La première consiste à introduire le parallélisme pendant l'évaluation du nœud de l'arbre du Branch and Bound. Ce parallélisme n'influe pas sur la structure de l'algorithme du Branch and Bound. Il dépend fortement de la méthode d'évaluation d'un sous-problème.
2. La deuxième stratégie de parallélisation consiste à construire l'arbre du Branch and Bound en parallèle. Plusieurs nœuds de l'arbre du Branch and Bound sont évalués simultanément sur les processeurs de la machine parallèle.
3. Le dernier type de parallélisme consiste à construire plusieurs arbres du Branch and Bound en parallèle avec des stratégies différentes. Sur chaque processeur, on choisit une stratégie différente de branchement, de sélection ou d'élimination...etc. Les informations recueillies sur un processeur pendant la construction d'un arbre peuvent être utilisées sur d'autres processeurs.

Les trois stratégies de parallélisation peuvent être combinées soit d'une façon séquentielle, soit d'une façon hiérarchique.

7.4.3 Les anomalies de comportement des algorithmes parallèles de Branch and Bound

Il existe des études théoriques sur le comportement des algorithmes de Branch and Bound synchrones, à file d'attente unique. Dans ce type d'algorithme les sous-problèmes sont stockés dans une file d'attente unique. La borne supérieure est stockée dans une variable globale accessible par tous les processeurs. Initialement, le nœud racine est ajouté à la file F. A chaque itération, $min(|F|; p)$ nœuds de F sont choisis selon un certain critère de sélection. Chaque nœud sélectionné est traité par un processeur. Chaque processeur effectue l'opération de branchement, d'évaluation et le test d'élimination. A la fin de chaque itération la borne supérieure est mise à jour et les nouveaux nœuds générés sont ajoutés à la file F. l'algorithme s'arrête quand la file F est vide au début d'une itération.

Les stratégies de sélection peuvent être caractérisées par une fonction qui associe à chaque nœud une valeur. On sélectionne le nœud qui possède la plus petite valeur. La stratégie meilleure d'abord consiste à choisir le sommet qui possède la plus petite borne inférieure. Dans ce cas la valeur d'un nœud est égale à la valeur de sa borne inférieure. Cette même fonction retourne l'opposé de la profondeur d'un nœud quand le critère de sélection suit la stratégie profondeur d'abord. La

fonction de sélection joue un rôle central dans la compréhension des anomalies de comportement de certains algorithmes parallèles de Branch and Bound. On dégage deux types d'anomalies :

- Anomalie d'accélération : $S_p > p$ où p est le nombre de processeurs et S_p l'accélération associée à p.
- Anomalie préjudiciable de croissance : $T_{p_1} = T_{p_2} < 1$ avec $p_1 < p_2$ où T_{p_1} est le temps d'exécution sur p_1 processeurs, et T_{p_2} est le temps d'exécution sur p_2 processeurs.

Les anomalies de comportement des algorithmes parallèles de Branch and Bound utilisant la stratégie de sélection "meilleur d'abord" ont été étudiées par Lai et Sahni [79] et Lai et Sprague [78]. Ils ont montré que les anomalies d'accélération ne peuvent pas se produire si les valeurs de tous les nœuds internes à l'arbre du Branch and Bound sont différentes de la valeur de la solution optimale. Mans et Roucairol [5] ont proposé des règles de gestion des nœuds de mêmes priorités pour éliminer les anomalies. Quin et Deo [54] ont donné des bornes supérieures de l'accélération des algorithmes asynchrones à file d'attente unique utilisant le critère de sélection meilleur d'abord. Corrêra et Ferreira ont présenté un état de l'art de ses anomalies dans [68].

7.5 Les bibliothèques PVM et PMI

PVM [1] a pour but de faciliter l'écriture d'applications parallèles sur un réseau de stations de travail hétérogènes. L'utilisateur définit un ensemble de machines qui sera vu comme une seule grande machine multiprocesseur à mémoire distribuée. Le terme machine parallèle fait référence à cette machine multiprocesseur alors que le terme hôte se réfère à l'une des machines membre de l'ensemble. Cette approche permet le regroupement d'hôtes ayant des caractéristiques différentes (machine vectorielle, station graphique, station de travail, cluster, robot de stockage,... etc). Les communications entre les hôtes se font avec PVM qui prend en charge les conversions de formats nécessaires entre deux machines n'ayant pas les mêmes représentations internes des données. PVM est également conçue pour pouvoir fonctionner avec différents types de réseaux.

Le site netlib est un site où l'on peut récupérer la distribution PVM ainsi qu'un manuel on-line (décrivant les fonctions PVM), de même la home page de PVM développée par Oak Ridge National Laboratory (ORNL).
MPI [2] est une bibliothèque d'échanges de messages pour machines parallèles homogènes. Une application MPI est un ensemble de processus exécutant chacun son propre code et communiquant via des appels à des sous-programmes de la bibliothèque MPI. Le forum MPI est le site dédié à MPI, de même qu'argonne ou encore le site netlib.

7.6 Conclusion

Nous avons présenté dans ce chapitre quelques notions liées au parallélisme et aux algorithmes branch and bound parallèles génériques. En effet, nous avons montré que la conception de tels d'algorithmes n'est pas une tâche facile. Cette conception dépend fortement du problème combinatoire traité et surtout de l'architecture de la machine parallèle. Dans le chapitre suivant, nous présentons deux algorithmes parallèles pour le problème de découpe contraint à deux dimensions

1. *Parallel Virtual Machine*
2. *Message Passing Interface*

et à deux niveaux. La méthode parallèle proposée s'appuie sur le paradigme maître/esclave et les spécificités de la recherche par faisceaux. Nous appliquons les mêmes mécanismes d'évaluation et de réduction de l'espace de recherche que ceux présentés dans les chapitres 3 et 4 tous en exploitant le parallélisme, afin d'élargir l'espace exploré et diversifier la recherche d'une solution approchée.

Chapitre 8

Algorithmes parallèles pour le problème de découpe contraint à deux dimensions [4] [5]

Contenu

Dans ce chapitre, nous proposons deux méthodes de résolution parallèle approchées, pour le problème de découpe contraint à deux dimensions et à deux niveaux. Les deux méthodes parallèles proposées s'appuient sur le paradigme maître/esclave et les spécificités de la recherche par faisceaux. Nous appliquons les mêmes mécanismes d'évaluation et de réduction de l'espace de recherche que ceux présentés dans le chapitre 3 et 4 de cette thèse en exploitant le parallélisme afin d'élargir l'espace exploré et diversifier la recherche d'une solution approchée.

4. *Une description de la recherche par faisceaux parallèle a été soumise à "Informs Journal On Computing [49]" et a été présentée lors de la conférence internationale "International Conference on Metaheuristics and Nature Inspired computing,* META 2006 [45]"

5. *L'algorithme coopératif parallèle a été soumis à "Operations Research [50]" et a été présenté l'ors de la conférence internationale "International workshop on Operation Research,* IWOR 2008 [48]"

8.1 Recherche par faisceaux parallèle

8.1.1 Analyse de l'algorithme GBS

Dans cette section, nous faisons une brève description de l'algorithme séquentiel GBS, présenté dans le chapitre 3 (noté SGBS). Nous faisons un survol des principales phases de cet algorithme, avant de décrire les traitements qui peuvent se dérouler en parallèle. Les éléments les plus importants dans une méthode parallèle sont les structures de données, l'architecture fonctionnelle et le schéma de coopération entre les processeurs. Ainsi, nous donnerons une importance particulière à ces points en décrivant, comment les processeurs vont collaborer pour résoudre le problème de découpe à deux dimensions et quelles sont les différentes structures de données (liste de traitements et queues de communication) utilisées pour guider la recherche et exploiter les avantages de la recherche par faisceaux.

L'examen de l'algorithme séquentiel proposé dans le chapitre 3, révèle la possibilité d'explorer l'espace de recherche en parallèle lors de la résolution du problème FC2TDC. Ainsi, le processus de recherche passe la majorité du temps de résolution dans les phases suivantes :

1. La phase de sélection : Dans cette phase, l'algorithme sélectionne le meilleur nœud à développer parmi un ensemble de nœuds élites.
2. La phase d'extension : Dans cette phase, toutes les bandes générales sont créées pour chaque hauteur w_i. Avant de calculer, toutes les bornes supérieures en combinant la bande optimale courante avec les sous rectangles restants.
3. La phase de filtrage : Dans cette phase, un ensemble des nœuds élites, capables de générer des meilleures solutions sont retenus pour une autre phase de sélection.

L'algorithme parallèle proposé fonctionne suivant le paradigme maître/esclave. Le processeur maître guide la recherche en utilisant une liste globale de nœuds à explorer. Ce processeur utilise la stratégie du meilleur d'abord pour sélectionner les futures nœuds à développer.

Les processeurs esclaves, de leurs coté, développent les chemins de recherche et utilisent leurs propres listes de nœuds à développer (listes locales). Une mise à jour de la liste globale (la liste du processeur maître) est faite périodiquement. Cette mise à jour permet de diversifier la recherche et d'équilibrer la charge de résolution entre les processeurs esclaves.

8.1.2 Architecture de l'algorithme parallèle

Un des objectifs du parallélisme est la réduction du temps de calcul. Les phases de sélection et d'extension sont extrêmement difficiles à prédire et le traitement associé à ces deux phases est très variable. La phase de sélection est réalisée en utilisant une liste de nœuds ordonnés selon un critère spécifique. La phase d'extension est très gourmande en temps de calcul car (i) elle génère (pour chaque sélection) toutes les bandes générales, (ii) effectue une évaluation selon la borne supérieure et la solution réalisable courante, (iii) fait une mise à jour de la meilleure solution interne courante (meilleure solution générée par le processeur esclave).

Ces facteurs imposent l'utilisation d'un mécanisme de régulation de charge dynamique, afin de maintenir une charge de traitement équilibrée entre les processeurs esclaves et d'éviter la saturation du processeur maître.

L'algorithme parallèle proposé, fonctionne comme suite :

1. Une *liste globale B est gérée par le processeur maître ; un sous ensemble de nœuds élites est sélectionné, puis assigné à un processeur esclave.*
2. *Chaque processeur esclave utilise une* liste interne pour effectuer les phases de sélection, d'extension et de filtrage.
3. L'algorithme utilise un mécanisme de régulation de charge par seuil ξ. Le seuil limite la taille de la première liste interne de chaque processeur esclave. Si la taille d'une liste interne dépasse le seuil fixé, les éléments qui débordent sont insérés dans une liste secondaire et les nœuds sont transférés d'une manière asynchrone au processeur maître.
4. A chaque période, nommée T, une synchronisation des listes internes des processeurs esclaves et la liste globale du processeur maître, est réalisée. Cette synchronisation permet d'initialiser les listes internes associées aux processeurs esclaves et de redémarrer la résolution avec une nouvelle solution initiale (la meilleure solution réalisable courante).

Nous rappelons qu'à partir d'un nœud $\mu = \big((L, W - Y),\ (L, Y)\big)$ émerge $r,\ r \leq s$, différents nœuds $\big((L, W - Y + \omega),\ (L, Y - \omega)\big)$, où $\omega = \overline{w}_1, \ldots, \overline{w}_r$.

Pour chaque nœud généré ν, une borne inférieure partielle $\hat{z}_\nu^{Local}$ et une borne supérieure complémentaire $\mathcal{U}_\nu$ sont nécessaires pour évaluer le future chemin à développer. La phase de filtrage est appliquée, après l'évaluation.

8.1.3 Structure de données

Comme discuté précédemment, l'algorithme séquentiel combine les phases de sélection, d'extension et de filtrage pour chercher une solution finale. L'algorithme parallèle proposé explore un sous ensemble de nœuds prés sélectionnés suivant le paradigme maître/esclave.

Le processeur maître guide la recherche en utilisant une liste globale de nœuds à développer, spécialement à l'initialisation et lorsqu'un processeur esclave se libère ou ne peut plus supporter la charge de résolution (débordement de sa liste interne). Par ailleurs, chaque processeur esclave développe sa propre arborescence de recherche et fait des mises à jour périodiques (chaque période T) de la liste globale (processeur maître).

La recherche est guidée par une borne supérieure complémentaire. Cette borne supérieure permet de choisir le chemin à développer suivant une évaluation globale. On remarque qu'une itération de l'algorithme parallèle est similaire à une exploration d'une partie de l'espace de recherche (lorsqu'un processeur utilise sa propre liste interne) ou à un changement de chemin (lorsque le processeur maître, réaffecte de nouveaux nœuds à tous les processeurs esclaves).

Dans la suite, nous décrivons les structures de données de chaque type de processeur et leurs rôles dans la résolution.

8.1.3.1 Listes du processeur maître et des processeurs esclaves

Le *processeur maître* (voir. Figure 8.1) sauvegarde l'ensemble des nœuds dans une *liste globale*, notée B_{master}. Cette liste est utilisée, initialement, à générer le premier ensemble s de bandes associées aux hauteurs distincts des pièces $\overline{w}_j,\ j \in J$. Ces différentes bandes correspondent à s

solutions réalisables partielles, obtenues en appliquant la programmation dynamique au problème de sac à dos $BK^{hor}_{L,\overline{w}_s}$ (défini dans le chapitre 3), où s correspond à la plus grande hauteur $\overline{w}_s$, tel que $s = \max\{\overline{w}_s \mid \overline{w}_s \geq \overline{w}_j,\ j \in J\}$.

Après ce calcul, le processeur maître assigne les s bandes optimales générées aux processeurs esclaves disponibles. Naturellement, le processeur maître ne peut affecter que $\min\{s, \eta\}$ nœuds, où η dénote le nombre de processeurs esclaves disponibles. Les nœuds assignés aux processeurs esclaves, sont obtenus en utilisant une stratégie du meilleure d'abord basée sur une évaluation globale (Voir chapitre 3).

Chaque *processeur esclave* (voir. Figure 8.1) reçoit un nœud initial $\mu = \big((L, W-Y),\ (L, Y)\big)$ du processeur maître, où μ est caractérisé par sa hauteur résiduelle Y et le nombre $v_i,\ i = 1, \ldots, n$, d'apparitions de la i-ème type de pièce dans la solution réalisable partielle z^{Local}_{μ}. Chaque processeur esclave utilise *deux listes internes* : (i) la première liste, nommée B_{slave}, stocke les meilleurs nœuds crées pendant le développement (suivant la stratégie du meilleur d'abord), et (ii) la deuxième liste, notée $\overline{B}_{slave}$, sauvegarde le reste des nœuds, lorsque la taille de la première liste B_{slave} atteint le *seuil* maximum. Dans la suite, nous considérons que la taille maximum des listes B_{slave} est la même pour tous les processeurs esclaves $\xi = \xi_{slave},\ \text{for } slave = 1, \ldots, \eta$.

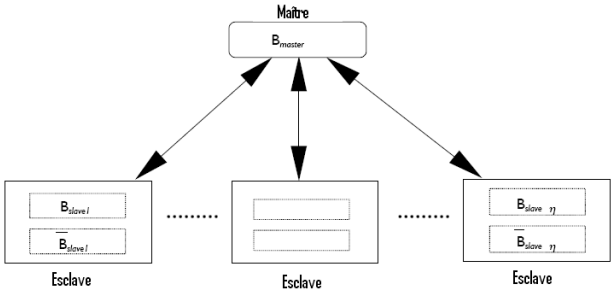


FIGURE 8.1 – Illustration de la liste globale et des deux listes internes des processeurs esclaves.

Lorsque un processeur esclave reçoit un nœud donné $\mu = \big((L, W-Y), (L, Y)\big)$, il commence son processus de calcul basé sur les étapes suivantes :

1. Calcul des demandes résiduelles b'_i pour chaque type de pièce $i \in I$, où $b'_i = b_i - v_i$ avec v_i le nombre d'apparitions de la pièce de type i dans $(L, W-Y)$.
2. Création des r $(r \leq s \leq n)$ bandes générales en résolvant $BK^{g}_{L,\overline{w}_r}$, où $r = \max\Big\{\overline{w}_r \mid \overline{w}_r \geq \overline{w}_j,\ j \in J\Big\}$ par programmation dynamique.
3. Pour chaque bande $(L, \overline{w}_j),\ \overline{w}_j \leq Y,\ j = 1, \ldots, r$, faire :
   (a) Placer la bande dans le sous rectangle courant (L, Y),
   (b) Obtenir un nouveau nœud $\nu = \big((L, W-Y+\overline{w}_j), (L, Y-\overline{w}_j)\big)$,
   (c) Calculer la valeur $\hat{z}_\nu$ du nouveau nœud ν.

Nous avons noté dans le chapitre 3, que l'algorithme séquentiel utilise une liste locale B_δ pour stocker tous les nœuds générés à partir d'un nœud donné μ. Dans l'algorithme parallèle, cette liste est remplacée par la liste $\overline{B}_{slave}$. Dans l'algorithme parallèle, tous les nœuds générés à partir de $\mu \in B_{slave}$ sont stockés, en premier lieu dans la liste secondaire $\overline{B}_{slave}$. Les meilleurs nœuds de $\overline{B}_{slave}$ sont ensuite, transférés vers B_{slave}. Les éléments $\overline{B}_{slave}$ sont finalement, transférés, vers la liste globale B_{master}.

Dans la phase d'extension et pour un nœud donné $\mu \in B_{slave}$, r nouveaux nœuds sont crées (r et le nombre de hauteurs distincts). Chaque nœuds dispose d'un indice d'évaluation. La phase de filtrage est appliquée afin de sélectionner les chemins les plus intéressants et qui correspondent aux nœuds élites. Ces nœuds réalisent $\min\{\delta, |\overline{B}_{slave}|\}$ les meilleurs écarts (comme défini dans la section 3.2.4.3 du chapitre 3). δ dénote la largeur du faisceau.

8.1.3.2 Stratégie de diversification

Comme discuté précédemment, chaque processeur esclave obtient les nœuds à développer de B_{slave}. Cette liste est modifiée par son processeur ôte lors de la phase d'extension et de filtrage. Néanmoins, un processeur esclave transfert (i) un sous ensemble de nœuds $\overline{B}_{slave}$ ou (ii) tous les nœuds $B_{slave} \cup \overline{B}_{slave}$ au processeur maître dans les cas suivants :

(1) Lorsque le seuil ξ de B_{slave} est atteint. Dans ce cas, le processeur maître reçoit les nœuds envoyés par l'esclave et retient ceux dont l'évaluation est supérieure à la meilleure solution réalisable courante $z^\star$.

(2) Lorsque la période de synchronisation T est écoulée. Dans ce cas, tous les nœuds de $B_{slave} \cup \overline{B}_{slave}$ sont transférés au processeur maître.

Dans le cas (1), l'algorithme utilise le seuil ξ afin d'équilibrer la charge entre les processeurs esclaves, Ce mécanisme permet de décharger un processeurs débordé (réduire la taille des listes internes), d'éliminer des chemins déjà explorés par d'autres processeurs (évaluation au niveau du processeur maître) et d'initialiser la résolution sur d'autres processeurs.

Dans le cas (2), l'algorithme essaie de diversifier la recherche d'une nouvelle solution réalisable. Par ailleurs, le processeur maître considère les étapes suivantes, avant d'initialiser les traitements sur les η processeurs esclaves :

(1) Il met un jour la meilleure solution réalisable obtenue par les processeurs esclaves et replace B_{master} par $B_{master} \cup \sum_{slave=1}^{\eta} B_{slave} \cup \sum_{slave=1}^{\eta} \overline{B}_{slave}$.

(2) Il applique une phase de filtrage pour réduire le nombre de nœuds dans B_{master}.

(3) Il sélectionne $\min\{\eta, |B_{master}|\}$ meilleurs nœuds en utilisant la stratégie du meilleur d'abord et affecte ces nœuds aux processeurs esclaves.

(4) Il supprime les nœuds affectés de la liste B_{master}.

`Entrée` : Instance FC2TDC.

`Sortie` : Solution de valeur $z^\star$.

1. *Phase d'initialisation*
   (a) $B_{master} = \left\{\mu = ((L,0),(L,W))\right\}$ et $z^\star = 0$.
   (b) Brancher sur μ pour générer toutes les bandes générales optimales associées à (L,W), par la résolution du problème $BK^{hor}_{L,\overline{w}_s}$, où $s = \max\{\overline{w}_s \mid \overline{w}_s \geq \overline{w}_j,\ j \in J\}$.
   (c) Pour μ, calculer la borne supérieure correspondant au sous rectangle $(L,Y),\ Y = 0,\ldots,W$, par la résolution le problème de sac à dos $UK_{(L,W)}$.
   (d) Appliquer la phase de filtrage sur les s nœuds crées et les insérer dans B_{master}.
   (e) Soit *Active* le vecteur associé aux processeurs esclaves (initialement $\forall k,\ Active_k$:=`false`), telle que $Active_k$:=`false`, $k = 1,\ldots,\eta$, si le k-ème processeur est disponible, `true` sinon.
   (f) Assigner les $\alpha = \min\left\{s,\ \max\{\delta,\eta\}\right\}$ nœuds de B_{master} aux processeurs esclaves disponibles, c'est à dire,
   $\forall k \in \{1,\ldots,\alpha\}$, `Appeler Slave_Algo`$(\mu_k, B_k, \overline{B}_k, T)$ et $Active_k$ =`true`.
2. *Phase de synchronisation*
   Cas 1. Un sous-ensemble de processeurs esclaves envoient des nœuds :
      (a) Le processeur maître reçoit les nœuds $\overline{B}_k,\ k = 1,\ldots,\eta'$ (où $\eta' \leq \eta$).
      (b) $B_{master} = B_{master} \cup_{k=1}^{\eta'} \overline{B}_k$.
      (c) Appliquer la phase de filtrage sur la liste B_{master}.
   Cas 2. Le k-ème processeur termine la résolution :
      (a) $Active_k$ =`false` et mettre à jour la meilleure solution réalisable courante $z^\star$.
      (b) Appliquer la phase de filtrage sur B_{master}.
      (c) Si $B_{master} \neq \emptyset$, soit $\mu \in B_{master}$ le meilleur nœud sélectionné par la stratégie du meilleur d'abord.
      Assigner μ au processeur esclave disponible, nommé ℓ, et mettre $Active_\ell$ =`true`.
   Cas 3. La période T a écoulée :
      (a) Le processeur maître reçoit les nœuds des $\eta' \leq \eta$ processeurs actifs.
      (b) $\forall k \in \{1,\ldots,\eta'\}, Active_k$ =`false`.
      (c) $B_{master} = B_{master} \cup_{k=1}^{\eta'} B_k \cup_{k=1}^{\eta'} \overline{B}_k$.
      (d) Mettre à jour la meilleure solution réalisable courante $z^\star$.
      (e) Appliquer la phase de filtrage sur la liste globale B_{master} tel que $\mu \in B_{master}$ si et seulement si, $z^\star < \hat{z}^{Local}_\mu + \mathcal{U}_\mu$.
      (f) Si $B_{master} \neq \emptyset$, alors répéter l'étape 1.(f) de la phase d'initialisation.
3. *Phase d'arrêt*
   Quitter avec la meilleur solution réalisable $z^\star$ lorsque (a) $B_{master} = \emptyset$, (b) $Active_k = false, \forall k \in \{1,\ldots,\eta\}$ et (c) toutes les queues de communication sont vides.

FIGURE 8.2 – Procédure globale du processeur maître dans PGBS : `MasterPGBS_Algo`.

8.1.4 L'algorithme

8.1.4.1 Le processeur maître

La figure 8.2 décrit les principales phases du traitement effectuées par le processeur maître, noté `MasterPGBS_Algo`. Il est composé de trois phases : L'initialisation, la synchronisation et la phase d'arrêt.

La *phase d'initialisation* sert à initialiser la liste globale B_{master} (Étape 1.(a)) par le premier nœud $\mu = \big((L,0),(L,W)\big)$ et la construction du premier ensemble s de bandes générales optimales. Ces bandes sont calculées par la résolution du problème $BK^{hor}_{L,\overline{w}_s}$ (Étape 1.(b)). La programmation dynamique permet d'obtenir la structure de toutes les bandes optimales de hauteurs $\overline{w}_j \leq \overline{w}_s,\ j \in J := \{1,\ldots,s\}$, avec leurs valeurs $f^{hor}_{\overline{w}_j}$.

Par ailleurs, dans la phase 1.(c), l'algorithme calcul la borne supérieure complémentaire $UK_{(L,W)}$; pour le couple $(\overline{w}_j, f^{hor}_{\overline{w}_j}),\ j \in J$. Cette borne supérieure est calculée par programmation dynamique qui permet d'avoir toutes les bornes supérieures $\mathcal{U}_{(L,Y)},\ Y = 0,\ldots,W,$. Dans la phase 1. (d), la phase de filtrage est appliquée sur l'ensemble s. Cette phase consiste à : (i) mettre à jour la meilleure solution réalisable courante $z^\star$ par la résolution de $\max\limits_{j\in J}\big\{\hat{z}^{Local}_j\big\}$, et (ii) modifier de la structure (combinaison de quelques bandes) de la borne supérieure $\mathcal{U}_{(L,Y)}$ dans le but de trouver une solution réalisable. Ceci est réalisé par la suppression des pièces qui induisent la non réalisabilité de la solution (la suppression des pièces est effectuée d'une manière séquentielle de 1 à n).

Dans l'étape 1. (e), un vecteur *Active* est utilisé pour contrôler et suivre l'état des processeurs esclaves. L'état de chaque processeur k est représenté par $Active_k$, qui est égale à `false` si le processeur est disponible, `true` sinon. Initialement, tous les processeurs sont disponibles.

Dans L'étape 1. (f), le processeur maître sélectionne les α meilleurs nœuds suivant la stratégie du meilleur d'abord. Ces nœuds réalisent la meilleure évaluation globale $\hat{z}^{Global}_j = \hat{z}^{Local}_j + \mathcal{U}_{(L,W-\overline{w}_j)},\ j \in J,$. Ces nœuds sont assignés aux processeurs esclaves,
(`Slave_Algo`$\big(\mu_k, B_k, \overline{B}_k, T\big),\ k = 1,\ldots,\alpha$, où μ_k est le k-ème nœud sélectionné de B_{master}, B_k et $\overline{B}_k$ dénote les listes internes du $k$ème processeur esclave.

La *phase de synchronisation* est composée de trois cas et le processeur maître réagit selon le cas à traiter. Dans le premier cas, un nombre, nommé $\eta' \leq \eta$, de processeurs esclaves envoie leurs nœuds résiduels (les nœuds résiduels sont choisis de la liste interne secondaire de chaque processeur esclave $\overline{B}_{slave}$). Le processeur maître reçoit les η' listes (étape 2.1. (a)) et les insert dans B_{master} (étape 2.1.(b)). Il applique, Alors, dans la l'étape 2.1. (c), une phase de filtrage qui élimine les duplications, ainsi que les nœuds dont l'évaluation globale est inférieure à la solution réalisable courante $z^\star$ (Naturellement, le processeur maître met à jour la meilleure solution réalisable courante, si possible).

Dans de second cas, un processeur esclave, nommé k, termine son traitement. Dans ce cas (étape 2.2. (a)), le processeur maître reçoit le contexte caractérisé par la meilleure solution produite et sa structure, il remplace l'état du processeur esclave concerné ($Active_k = false$) et met à jour la solution réalisable courante. Dans l'étape 2.2. (b), le processeur maître applique une phase de filtrage sur la liste globale ; Dans ce traitement, les nœuds ayant une évaluation globale inférieure à $z^\star$ sont supprimé de B_{master}. Après cette étape, le processeur maître sélectionne le meilleur nœud, nommé μ de la liste B_{master} et l'assigne à un processeur esclave disponible.

Dans le troisième cas, tous les processeurs esclaves sont arrêtés (fin de la période T). En premier lieu, dans l'étape 2.3. (a), le processeur maître reçoit tous les nœuds $\cup_{k=1}^{\eta'} B_k \cup_{k=1}^{\eta'} \overline{B}_k$ des processeurs esclaves actifs, nommés $\eta' \leq \eta$. En second lieu, il change (dans l'étape 2.3.(b)), l'état des η' processeurs esclaves de `true` à `false` et insert (étape 2.3.(c)) les nœuds $\cup_{k=1}^{\eta'} B_k \cup_{k=1}^{\eta'} \overline{B}_k$ dans B_{master}. De la même manière, dans l'étape 2.3. (d), le processeur maître met à jour la meilleure solution réalisable courante $z^\star$ et applique la phase de filtrage sur B_{master}. Finalement, dans l'étape 2.3.(e), il assigne les α meilleurs nœuds aux processeurs esclaves disponibles et $B_{master} \neq \emptyset$.

La *phase d'arrêt* sert à arrêter la résolution et l'obtention de la meilleure solution $z^\star$. Naturellement, le processeur maître arrête la résolution lorsque tous les éléments du vecteur *Active* sont à `false`. Cela signifie que tous les processeurs esclaves ont fini leurs résolutions et que toutes les queues de communications sont vides (les queues et les processus de communication sont décrits dans la section 8.1.4.3).

`Entrée` : Le contexte du nœud $\mu = ((L, W_\mu), (L, W - W_\mu))$ et la valeur de la période T.
`Sortie` : (i) La valeur de la meilleure solution locale $z_k^\star$ et les deux listes internes (B_k et $\overline{B}_k$), ou (ii) les nœuds de la liste secondaire $\overline{B}_k$.

1. *Phase d'initialisation*
   1. $B_k = \{\mu\},\ \overline{B}_k = \emptyset,\ z_k^\star = z^\star$ et soit $b'_i,\ i \in I$, les demandes résiduelles après filtrage des solutions partielles dans (L, W).
   2. Soit r le nombre de hauteurs distincts, tel que $\overline{w}_r \leq W - W_\mu$ et $\forall j \in \{1, \ldots, r\},\ \overline{w}_r \geq \overline{w}_j$. Créer les r bandes générales optimales associées à μ, par résolution (programmation dynamique) du problème $BK^{hor}_{L,\overline{w}_r}$ avec les valeurs des demandes $b'_i,\ i \in I$.
   3. Calculer les bornes supérieures complémentaires associées à $(L, Y),\ Y = 0, \ldots, W - W_\mu$, par résolution de UK_μ.
2. *Phase itérative*
   1. Sélectionner de B_k le nœud $\mu = ((L, W - Y), (L, Y))$ avec la meilleure évaluation globale $z_\mu^{Global} = \hat{z}_\mu^{Local} + \mathcal{U}_\mu$.
   2. Supprimer le nœud μ de B_k.
   3. Brancher sur μ et générer les bandes valides par résolution du problème $BK^{hor}_{L,\overline{w}_r}$, où $\overline{w}_r$ dénote la plus haute bande (avec le plus grand indexe) qui peut être placer dans le sous rectangle (L, Y).
   4. Pour chaque nouveau nœud $\nu = \Big((L, W - Y + \omega), (L, Y - \omega)\Big),\ \omega = \overline{w}_1, \ldots, \overline{w}_r$,
      (a) Effectuer une évaluation globale basée sur la borne inférieure $\hat{z}_\nu^{Local}$ et la borne supérieure complémentaire $\mathcal{U}_\nu$.
      (b) Mettre à jour la meilleure solution courante : $z_k^\star = \max\Big\{z^\star,\ \hat{z}_\nu^{Local}\Big\}$.
   5. Insérer les nœuds crées ν dans $\overline{B}_k$ si $z_k^\star < \mathcal{U}_\nu + \hat{z}_\nu^{Local}$ selon l'équation (4.1) ; écarter ν sinon.
   6. Appliquer la phase de filtrage sur les bandes générées ; soit $\overline{B}_k$ l'ensemble des nœuds élites ($\min\{\rho, |\overline{B}_k|\}$ meilleures nœuds de $\overline{B}_k$.)
   7. Si $|B_k \cup \overline{B}_k| > \xi$, alors copier les meilleurs $\xi - |B_k|$ nœuds de $\overline{B}_k$ à B_k et les supprimer de $\overline{B}_k$; mettre $B_k = B_k \cup \overline{B}_k$ sinon.
   8. Si $\overline{B}_k \neq \emptyset$, alors envoyer le contenu de $\overline{B}_k$ au processeur maître.
   9. Supprimer tous les éléments de $\overline{B}_k$.
3. *Arrêter ou répéter la recherche :*
   1. Si $B_k = \emptyset$ ou la période T a écoulée, alors
      (a) $B_k = \emptyset$: envoyer la meilleure solution locale obtenue $z_k^\star$ au processeur maître.
      (b) T a écoulée : envoyer la meilleure solution local obtenue $z_k^\star$ et les deux listes internes (B_k et $\overline{B}_k$).
   2. Sinon, répéter *étape itérative 2.*

FIGURE 8.3 – Description des étapes du k-ème processeur esclave dans PGBS : `SlavePGBS_Algo`.

8.1.4.2 Les processeurs esclaves

La figure 8.3 décrit les phases principales de l'algorithme, noté `Slave_Algo`. Cet algorithme est utilisé par chaque processeur esclave $k,\ k = 1, \ldots, \eta$, où η, dénote le nombre de processeurs esclaves disponibles. Naturellement, l'algorithme ressemble à la version séquentielle présentée dans le chapitre 3. Néanmoins quelques différences existent entre les deux versions. La figure 8.3 montre que l'algorithme est composé de trois phases principales :

Dans *la phase d'initialisation*, le processeur esclave reçoit le nœud μ. Il reçoit, également, les demandes résiduelles $b'_i,\ i \in I$. Dans l'étape 1.1, le nœud est stocké dans B_k et dans l'étape 1.2 ; les r nouvelles bandes générales sont crées pour chaque hauteur $\overline{w}_j,\ j \in J$ et leurs valeurs $f^{hor}_{\overline{w}_j},\ j \in J$ sont calculées par la résolution par programmation dynamique du problème :

$$(BK^{hor}_{L,\overline{w}_j}) \begin{cases} g^{hor}_{\overline{w}_j}(L) = \max & \displaystyle\sum_{i \in S_{L,\overline{w}_j}} c_i x_{ij} \\ \text{subject to} & \displaystyle\sum_{i \in S_{L,\overline{w}_j}} l_i x_{ij} \leq L \\ & x_{ij} \leq b'_i,\ x_{ij}\ \in N,\ i \in S_{L,\overline{w}_j}, \end{cases}$$

où $\overline{w}_r \leq W - W_\mu$ et $g^{hor}_{\overline{w}_j}(L)$ représentent les valeurs des bandes générales optimales de hauteurs $\overline{w}_j,\ j \in J := \{1, \ldots, r\}$. Par la suite, toutes les bornes supérieures complémentaires sont calculées par la résolution du problème suivant :

$$UK_{(L,W_\mu)} = \max \Big\{ \sum_{j \in J} g^g_{\overline{w}_j}(L) t_j \mid \sum_{j \in J} \overline{w}_j t_j \leq W - W_\mu, t_j \in \mathbf{N},\ j \in J \Big\}.$$

Dans *la phase itérative*, le processeur esclave commence par sélectionner le meilleur nœud (étape 2.1), nommé $\mu = ((L, W - Y), (L, Y))$, de B_k et le supprime de cette dernière (étape 2.2). Dans un deuxième temps (étape 2.3), r $(r \leq s \leq n)$ bandes générales sont créés par la résolution du problème $BK^g_{L,\overline{w}_r}$, où $r = \max \Big\{ \overline{w}_r \mid \overline{w}_r \geq \overline{w}_j,\ \overline{w}_j \leq W_\mu,\ j \in\ J \Big\}$. Dans l'étape 3, pour chaque nœud crée ν, les deux bornes inférieure ($\hat{z}^{Local}_\nu$) et supérieure complémentaire ($\mathcal{U}_\nu$) sont mises à jour afin d'effectuer une évaluation globale $\hat{z}^{Global}_\nu$. Dans l'étape 3.5, le processeur réalise une sélection des nœuds élites et les transfert à la deuxième liste interne $\overline{B}_k$. Dans l'étape 3.6, un filtrage est effectué sur $\overline{B}_k$ et les meilleurs $\min\{\delta, |\overline{B}_k|\}$ nœuds sont identifiés. Finalement, dans l'étape 3.7, il y'a une vérification de la taille de la liste interne $(B_k \cup \overline{B}_k)$. Si la taille de cette dernière, dépasse ξ, alors un sous ensemble d'éléments est transféré au processeur maître. Dans tous les cas, la deuxième liste interne $\overline{B}_k$ est vidée.

La phase d'arrêt est composée de deux parties. Dans la première partie (étape 3.1), le processeur esclave arrête la résolution, si la liste B_k est vide ou si la période de synchronisation T a écoulée. Dans le premier cas (liste B_k vide), le processeur esclave envoie la valeur de la solution réalisable locale courante au processeur maître. Si la période T est écoulée, le processeur esclave envoie, en plus de la valeur de la solution réalisable local courante, tous les nœuds des listes internes.

8.1.4.3 La communication entre les processeurs

La communication entre le processeur maître et les processeurs esclaves est réalisée à travers trois queues. Chaque queue est gérée par un processus indépendant du processus de résolution. Les informations sont exploitées selon la stratégie FIFO. Le rôle des trois queues est le suivant :

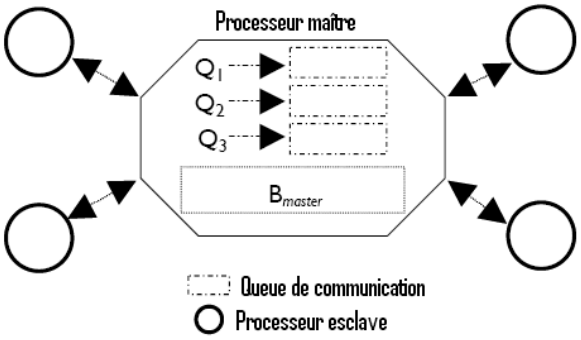


FIGURE 8.4 – Illustration des queues de communication utilisées par les processeurs.

Durant la résolution, la première queue (notée Q_1 dans la figure 8.4) est utilisée pour stocker les meilleures solutions réalisables locales transférées par les processeurs esclaves. Cette queue est utilisée dans la phase de synchronisation de `Algo_Master` (lorsque les processeurs esclaves envoient leurs solutions au processus maître). La deuxième queue (notée Q_2 dans la figure 8.4) permet de recevoir les nœuds résiduels envoyés par les processeurs esclaves (elle correspond au Cas 1 de l'étape de synchronisation dans `Algo_Master`). La dernière queue (notée Q_3) permet de recevoir les nœuds envoyés par les processeurs esclaves dans le cadre de la répartition de charge (nœuds qui saturent la première liste locale d'un ou de plusieurs processeurs esclaves). Cette queue est utilisée par le processeur maître afin de rediriger la résolution à d'autres processeurs esclaves disponibles est moins chargés.

Les traitements associés à ces trois queues, se font indépendamment de la résolution. Le traitement des différents cas associés à la synchronisation (cas 1 et 3) est réalisé en parallèle. Par ailleurs, après chaque période T (période de synchronisation), les deux queues Q_2 et Q_3 sont vidées.

8.1.5 Résultats numériques

Dans cette section, on évalue la performance de l'algorithme proposé (noté PGBS) sur deux groupes d'instances, classés dans la littérature comme larges et extra larges (ces instances sont décrites dans les parties expérimentale du chapitre 4). Nous avons codé l'algorithme PGBS en C et les communications entre les processeurs en pvm. Les résultats présentés sont obtenus sur un réseau de station Intel sous Linux Redhat.

Comme discuté dans le chapitre 3 de cette thèse, nous avons varié la valeur associée au paramètre δ (largeur du faisceau) dans l'intervalle $\{1, \ldots, 4\}$. Dans l'algorithme séquentiel SGBS, l'augmentation de la largeur du faisceau δ n'améliore pas forcement la qualité de la solution obtenue, mais augmente considérablement les temps de résolution. Dans la suite de cette section, nous testons l'efficacité de l'algorithme parallèle proposé sur les deux groupes d'instances et nous analysons le comportement de PGBS avec différentes valeurs du paramètre δ.

8.1.5.1 Premier groupe d'instances

Dans un premier temps, nous avons mené notre étude sur le premier groupe d'instances. Nous avons lancé deux résolutions pour chaque instance : la premier correspond à une première découpe horizontale et la deuxième à une première découpe verticale (les noms des instances sont préfixés par H ou V, selon la position de la première découpe). Pour la plupart des instances considérées, la solution optimale n'est pas connue, Nous avons, donc, comparé les solutions de PGBS aux meilleures solutions de la littérature (voir Hifi et al [42]). L'algorithme séquentiel GBS a des difficultés à résoudre quelques instances avec une largeur de faisceau élevé, par conséquence, nous avons limité le temps de résolution à 300 secondes pour ces instances. L'analyse des résultats obtenus est le suivant :

		SGBS				*PGBS* $\delta = 8$			
		$\delta = 4$		$\delta = 8$			$\eta = 6$	$\eta = 10$	$\eta = 20$
#Inst.	Opt/Best	Gap	cpu	Gap	cpu	Gap	cpu	cpu	cpu
nice25H*	860829	◦	0.05	◦	7.76	◦	0.02	0.01	0
nice50H	913738	◦	0.01	◦	23.87	◦	0.01	0.01	0
nice100H	911122	◦	0.05	-4646	–	**81**	0.04	0.03	0.01
nice200H	931143	◦	0.41	-11387	–	**8897**	0.21	0.11	0.01
nice500H	951279	◦	1.86	-15812	–	**2319**	1.01	0.09	0.04
path25H*	892765	◦	0.06	◦	17.76	◦	0.02	0.01	0.01
path50H*	750215	◦	0.05	◦	105.21	◦	0.02	0.01	0.01
path100H	888578	◦	0.08	33	–	**413**	0.03	0.01	0.01
path200H	889174	◦	1.59	-9253	–	**1049**	0.98	0.43	0.21
path500H	885112	◦	12.87	-14200	–	**2552**	8.96	4.03	2.01
nice25PH*	441993	◦	0.02	◦	5.65	◦	0.01	0	0
nice50PH	449292	◦	0.02	◦	21.87	◦	0.01	0	0
nice100PH	456306	◦	0.05	-7574	–	**403**	0.02	0.01	0.01
nice200PH	455727	◦	0.16	-26	–	**1161**	0.09	0.03	0.01
nice500PH	455036	◦	1.44	-19303	–	**3965**	1.03	0.75	0.61
nice1000PH	451805	◦	2.94	-21802	–	**7271**	1.05	0.56	0.31
path25PH*	464721	◦	0.01	◦	10.78	◦	0.01	0.01	0
path50PH	391626	◦	0.01	-885	89.09	◦	0.01	0.01	0
path100PH	456300	◦	0.09	-27502	–	**1697**	0.04	0.02	0
path200PH	452402	◦	1.34	-32698	–	**7454**	0.99	0.45	0.12
path500PH	434478	◦	40.52	-21747	–	**5689**	24.78	15.98	8.98
path1000PH	434966	◦	150.21	-10005	–	**5035**	78.98	41.87	17.98
nice25V*	834157	◦	0.01	◦	9.32	◦	0.01	0.01	0.01
nice50V	854425	◦	1.12	◦	22.98	◦	0.86	0.45	0.23
nice100V	911958	◦	3.41	-21387	–	◦	2.04	1.48	0.89
nice200V	935290	◦	4.55	-25384	–	**1400**	2.04	1.54	0.97
nice500V	960767	◦	10.54	-30896	–	**4837**	6.89	3.98	1.98
path25V*	699707	◦	0.01	◦	7.91	◦	0.01	0.00	0.00
path50V*	924908	◦	0.41	◦	29.07	◦	0.32	0.13	0.03
path100V*	754753	◦	3.43	-10007	–	◦	2.78	1.87	0.76
path200V	836022	◦	3.56	-20215	–	**732**	2.05	1.98	0.87
path500V	798544	◦	4.54	-30632	–	**1359**	2.45	1.42	0.72
nice25PV*	428662	◦	0.13	◦	10.87	◦	0.02	0.01	0.01
nice50PV	425921	◦	0.55	◦	31.67	◦	0.34	0.21	0.01
nice100PV	451851	◦	0.43	-22671	–	**3555**	0.21	0.11	0.01
nice200PV	454754	◦	1.23	-30790	–	**135**	1.07	0.98	0.42
nice500PV	457381	◦	10.41	-50080	–	◦	7.76	3.67	2.03
nice1000PV	459142	◦	3.44	-20000	–	◦	2.02	1.32	0.95
path25PV*	356088	◦	0.05	◦	12.67	◦	0.02	0.01	0.01
path50PV	487447	◦	0.11	◦	32.73	◦	0.04	0.02	0.01
path100PV	386033	◦	1.23	◦	–	◦	0.65	0.42	0.21
path200PV	418024	◦	2.34	-11368	–	**4681**	1.56	0.43	0.22
path500PV	368284	◦	5.2	-38295	–	**1097**	2.76	1.08	0.64
path1000PV	407097	◦	77.54	-29215	–	**2726**	40.76	22.76	13.76
Av_Gap		0.00		-12221.52		**1593.20**			
Av_cpu			7.91		200.89		4.43	2.46	1.25

TABLE 8.1 – Résultats de l'algorithme parallèle PGBS et de l'algorithme séquentiel SGBS. Le symbole "–" signifie que le temps de résolution de SGBS a dépassé 300 secondes.

La Table 8.1 rapporte les résultats obtenus sur le premier groupe d'instances. La colonne 1 montre le nom de l'instance et la colonne 2 contient la meilleure solution connue ou l'optimum s'il existe (les instances dont la solution optimale est connue, sont signalées par le symbole "$*$"). Les colonnes 3 et 5 montrent respectivement, la déviation, noté Gap et la solution obtenue par l'algorithme séquentiel SGBS avec $\delta = 4$ et $\delta = 8$. La colonne 4 et 6 donnent le temps de résolution de SGBS pour les deux valeurs de δ.

Les colonnes 7 à 10 montrent les résultats de l'algorithme parallèle PGBS. La colonne 7 montre la meilleure solution obtenue par PGBS alors que les colonnes 8, 9 et 10 montrent respectivement, le temps de résolution, la déviation (notée, Gap) et la différence entre la solution de la version considérée de l'algorithme et la meilleure solution connue (notée, Opt/Best) ; Dans ce cas, Gap > 0 signifie que la version considérée améliore la meilleure solution de la littérature.

Les deux dernières lignes de la table 8.1 donnent un résumé du contenu de celle-ci. Elles montrent, respectivement, la moyenne du Gap , noté Av_Gap et la moyenne des temps de résolution (Av_cpu), pour toutes les instances.

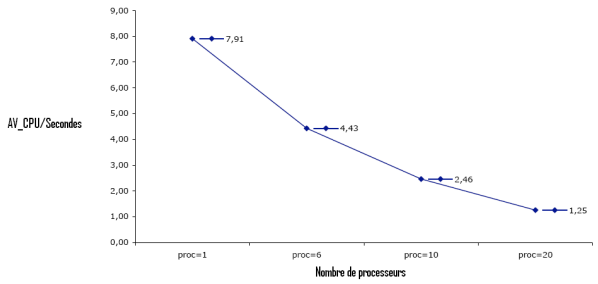


FIGURE 8.5 – Variation du temps de résolution par rapport au nombre de processeurs η.

La table 8.1 montre que :

1. $SGBS_4$ améliore les solutions de toutes les instances (44 instances). Son temps moyen de résolution est égale à 7.91 secondes.
2. $SGBS_8$ obtient des résultats médiocres par rapport à $SGBS_4$. En effet, 300 secondes ne sont pas suffisantes pour l'algorithme séquentiel avec une largeur de faisceau $\delta = 8$. Cette version de l'algorithme améliore la solution de 16 instances sur les 44 traitées (représentant un pourcentage de 36.36%) et dégrade les solutions des autres instances avec une moyenne de déviation de -12221.52. Par ailleurs, le temps de résolution augmente à 200.89 secondes en moyenne.
3. $PGBS_8$ est plus efficace que les deux versions de l'algorithme séquentiel SGBS. Il améliore les solutions obtenues pour 24 instances (54.54% des instances) et améliore les meilleures solutions connues de la littérature. On remarque également que parmi les instances non améliorées par $PGBS_8$, 11 ont des solutions optimales déjà connues.

   Soit $PGBS_8^{\eta}$ la version de l'algorithme parallèle avec η processeurs, $S(\eta)$ le speedup et $F(\eta)$ l'efficacité. Nous avons étudié l'impact du nombre de processeurs utilisés par PGBS sur sa

performance (l'analyse est illustrée dans la figure 8.5).

(a) Le temps moyen consommé par PGBS_8^6 est très inférieure des 7.91 secondes, consommées en moyenne par SGBS_4 et des 200.89 secondes consommées par SGBS_8. Selon le meilleur temps de résolution réalisé par SGBS_4 ; PGBS_8^6 réduit le temps de résolution de 44%. Par ailleurs, le speedup $S(6)$ et l'efficacité $F(6)$ sont respectivement égales à 1.79 et à 0.30.

(b) Le temps de résolution moyen de PGBS_8^{10} diminue de 44.47% par rapport à PGBS_8^6. Dans ce cas, le speedup $S(10)$ et l'efficacité $F(10)$ reviennent respectivement, égales à 3.21 et à 0.32.

(c) Le temps de résolution de PGBS_8^6 est plus intéressant. Il diminue de 2.46 secondes (temps de résolution moyen de PGBS_8^{10}) à 1.25 secondes. Dans ce cas, le gain en temps de calcul devient plus important par rapport à la meilleure version de l'algorithme séquentiel, il améliore de 84.20% le temps moyen de résolution avec un speedup $S(20)$ et une efficacité $F(20)$ égales, respectivement, à 6.32 et 0.32.

Selon la dernière analyse, Nous observons que $S(10)$ présente la moitié de $S(20)$ et les valeur $F(10)$ et $F(20)$ sont équivalentes.

	$T = 0.5$ Sec			$T = 1$ Sec			$T = 2$ Sec		
	$\eta = 6$	$\eta = 10$	$\eta = 20$	$\eta = 6$	$\eta = 10$	$\eta = 20$	$\eta = 6$	$\eta = 10$	$\eta = 20$
Av_Nj	8.41	3.84	1.64	3.73	2.09	0.93	1.80	0.86	0.30
Av_cpu	4.91	2.73	1.38	4.43	2.46	1.25	4.05	2.26	1.12
$\text{nb}_{Best/Opt}$	44	44	44	44	44	44	38	39	39

TABLE 8.2 – Impact de la variation de la période T sur les performances de l'algorithme parallèle PGBS.

Comme nous l'avons mentionné dans la section 8.1.3.2, PGBS utilise un paramètre T de synchronisation. Cette synchronisation peut être considérée comme une stratégie de diversification qui sert à explorer d'autres régions de l'espace de recherche. Dans ce sens, les résultats présentés dans la table 8.1 sont obtenues en fixant la valeur de la période T à 1 seconde. Ce choix a été réalisé après plusieurs tests. Les résultats obtenus en variant la valeur de la période T de 0.5 secondes à 2 secondes sont résumés dans la table 8.2.

La table 8.3 rapporte une comparaison détaillée des performances de PGBS, pour des périodes de synchronisation T égales à 0.5, 1 et 2. La table 8.2 résume les résultats obtenus.
La colonne 2 de la table 8.3 rapporte la meilleure solution (et la nouvelle solution) extraite de la table 8.1. Les colonne 3 à 8 montrent le résultat pour $T = 0.5$ avec différent nombre de processeurs : le temps de résolution (noté, cpu) et le nombre de sauts (noté Nj) qui correspond au nombre de synchronisations réalisées pendant la résolution. Les autres colonnes (de 9 à 20) montrent les même résultats pour les différentes valeurs de T.
Dans la table 8.2, la ligne 1 montre le nombre moyen de sauts (Av_Nj), la ligne 2 rapporte le temps moyen de résolution (Av_cpu) et la ligne 3 rapporte le nombre de fois ($\text{nb}_{Best/Opt}$) que PGBS a amélioré la meilleure solution connue (présentée dans la colonne 7 de la table 8.1).

Soit $\text{PGBS}_8^{\eta}(T)$ l'algorithme parallèle exécuté avec les paramètres η (nombre de processeurs esclaves), T (la valeur associée à la période de synchronisation) et la largeur du faisceau $\delta = 8$. L'analyse des résultats présentés dans la table 8.2 révèle que $\text{PGBS}_8^{\eta}(T = 1)$, pour $\eta = 6, 10, 20$, est

#Inst.	Best Sol. Tab. 8.1	$T = 0.5$ Sec						$T = 1$ Sec						$T = 2$ Sec					
		$\eta = 6$		$\eta = 10$		$\eta = 20$		$\eta = 6$		$\eta = 10$		$\eta = 20$		$\eta = 6$		$\eta = 10$		$\eta = 20$	
		cpu	Nj	cpu	Nj	cpu	Nj	cpu	Nj	cpu	Nj	cpu	Nj	cpu	Nj	cpu	Nj	cpu	Nj
nice25H	860829	0.02	0	0.01	0	0	0	0.02	0	0.01	0	0	0	0.02	0	0.01	0	0	0
nice50H	913738	0.01	0	0.01	0	0	0	0.01	0	0.01	0	0	0	0.01	0	0.01	0	0	0
nice100H	911203	0.04	0	0.03	0	0.01	0	0.04	0	0.03	0	0.01	0	0.04	0	0.03	0	0.01	0
nice200H	940040	0.21	0	0.11	0	0.01	0	0.21	0	0.11	0	0.01	0	0.21	0	0.11	0	0.01	0
nice500H	953598	1.98	2	0.09	0	0.04	0	1.01	0	0.09	0	0.04	0	1.01	0	0.09	0	0.04	0
path25H	892765	0.02	0	0.01	0	0.01	0	0.02	0	0.01	0	0.01	0	0.02	0	0.01	0	0.01	0
path50H	750215	0.02	0	0.01	0	0.01	0	0.02	0	0.01	0	0.01	0	0.02	0	0.01	0	0.01	0
path100H	888991	0.03	0	0.01	0	0.01	0	0.03	0	0.01	0	0.01	0	0.03	0	0.01	0	0.01	0
path200H	890223	0.98	0	0.43	0	0.21	0	0.98	0	0.43	0	0.21	0	0.98	0	0.43	0	0.21	0
path500H	887664	12.96	17	4.78	8	2.72	4	8.96	8	4.03	4	2.01	2	6,17	4	3.12	1	2.01	0
nice25PH	441993	0.01	0	0	0	0	0	0.01	0	0	0	0	0	0.01	0	0	0	0.00	0
nice50PH	449292	0.01	0	0	0	0	0	0.01	0	0	0	0	0	0.01	0	0	0	0	0
nice100PH	456709	0.02	0	0.01	0	0.01	0	0.02	0	0.01	0	0.01	0	0.02	0	0.01	0	0.01	0
nice200PH	456888	0.09	0	0.03	0	0.01	0	0.09	0	0.03	0	0.01	0	0.09	0	0.03	0	0.01	0
nice500PH	459001	1.36	2	0.89	1	0.61	0	1.03	0	0.75	0	0.61	0	1.03	0	0.75	0	0.61	0
nice1000PH	459076	1.58	2	0.56	0	0.31	0	1.05	0	0.56	0	0.31	0	1.05	0	0.56	0	0.31	0
path25PH	464721	0.01	0	0.01	0	0	0	0.01	0	0.01	0	0	0	0.01	0	0.01	0	0	0
path50PH	391626	0.01	0	0.01	0	0	0	0.01	0	0.01	0	0	0	0.01	0	0.01	0	0	0
path100PH	457997	0.04	0	0.02	0	0	0	0.04	0	0.02	0	0	0	0.04	0	0.02	0	0	0
path200PH	459856	0.99	0	0.45	0	0.12	0	0.99	0	0.45	0	0.12	0	0.99	0	0.45	0	0.12	0
path500PH	440167	27.18	49	17.48	34	9.38	16	24.78	22	15.98	13	8.98	8	20.88	10	14.13	7	8.01	3
path1000PH	440001	82.78	156	44.17	82	19.98	34	78.98	72	41.87	39	17.98	15	73.98	35	38.33	18	15.12	5
nice25V	834157	0.01	0	0.01	0	0.01	0	0.01	0	0.01	0	0.01	0	0.01	0	0.01	0	0.01	0
nice50V	854425	0.89	1	0.45	0	0.23	0	0.86	0	0.45	0	0.23	0	0.86	0	0.45	0	0.23	0
nice100V	911958	2.44	4	1.48	2	0.99	1	2.04	2	1.48	1	0.89	0	1.24	0	1.48	0	0.89	0
nice200V	936690	2.52	4	1.54	2	0.99	1	2.04	2	1.54	1	0.97	0	1.14	1	1.54	0	0.97	0
nice500V	965604	7.99	10	3.98	6	2.31	2	6.89	5	3.98	3	1.98	1	5.89	2	3.98	1	1.98	0
path25V	699707	0.01	0	0.00	0	0.00	0	0.01	0	0.00	0	0.00	0	0.01	0	0.00	0	0.00	0
path50V	924908	0.32	0	0.13	0	0.03	0	0.32	0	0.13	0	0.03	0	0.32	0	0.13	0	0.03	0
path100V	754753	2.78	4	2.15	4	0.99	1	2.78	2	1.87	1	0.76	0	2.18	1	1.87	0	0.76	0
path200V	836754	2.75	4	2.23	3	1.21	1	2.05	1	1.98	1	0.87	0	2.05	1	1.98	0	0.87	0
path500V	799903	2.55	4	1.62	2	0.99	1	2.45	2	1.42	1	0.72	0	2.45	1	1.42	0	0.72	0
nice25PV	428662	0.02	0	0.01	0	0.01	0	0.02	0	0.01	0	0.01	0	0.02	0	0.01	0	0.01	0
nice50PV	425921	0.34	0	0.21	0	0.01	0	0.34	0	0.21	0	0.01	0	0.34	0	0.21	0	0.01	0
nice100PV	455406	0.21	0	0.11	0	0.01	0	0.21	0	0.11	0	0.01	0	0.21	0	0.11	0	0.01	0
nice200PV	454889	1.27	2	0.98	1	0.42	0	1.07	1	0.98	1	0.42	0	1.07	0	0.98	0	0.42	0
nice500PV	457381	8.76	12	4.12	7	2.78	4	7.76	6	3.67	3	2.03	2	7.16	3	3.67	1	2.03	0
nice1000PV	459142	3.01	4	1.99	3	0.95	1	2.02	1	1.32	1	0.95	0	2.02	1	1.32	0	0.95	0
path25PV	356088	0.02	0	0.01	0	0.01	0	0.02	0	0.01	0	0.01	0	0.02	0	0.01	0	0.01	0
path50PV	487447	0.04	0	0.02	0	0.01	0	0.04	0	0.02	0	0.01	0	0.04	0	0.02	0	0.01	0
path100PV	386033	0.99	1	0.42	0	0.21	0	0.65	0	0.42	0	0.21	0	0.65	0	0.42	0	0.21	0
path200PV	422705	1.98	2	0.43	0	0.22	0	1.56	1	0.43	0	0.22	0	1.56	0	0.43	0	0.22	0
path500PV	369381	2.99	5	1.23	2	0.99	1	2.76	2	1.08	1	0.64	0	2.72	1	1.08	0	0.64	0
path1000PV	409823	43.76	85	27.76	12	13.76	5	40.76	37	22.76	22	13.76	13	39.56	19	20.16	10	11.76	5

TABLE 8.3 – Impact de la variation de la période T et du nombre de processeurs utilisés η sur les performances de l'algorithme parallèle PGBS.

plus performant que les autres versions de l'algorithme. par ailleurs, les deux versions $\text{PGBS}_8^\eta(0.5)$ et $\text{PGBS}_8^\eta(1)$ améliorent les solutions des 44 instances, mais $\text{PGBS}_8^\eta(1)$ est plus rapide que $\text{PGBS}_8^\eta(0.5)$. On remarque que , $\text{PGBS}_8^\eta(2)$ a amélioré 38 instances, et le temps de résolution est globalement très intéressant. On peut noter que $\text{PGBS}_8^\eta(1)$ reste favorable par rapport à $\text{PGBS}_8^\eta(2)$ car le rapport entre les qualités des solutions obtenues et le temps de résolution donne l'avantage à $\text{PGBS}_8^\eta(1)$;

dans ce sens, $\text{PGBS}_8^{\eta}(1)$ réalise un temps moyen de résolution qui varie de 0.0284 à 0.1007 secondes, alors que le temps de résolution de $\text{PGBS}_8^{\eta}(2)$ varie de 0.0287 à 0.1066.

8.1.5.2 Le deuxième groupe d'instances

Dans cette section, nous avons considéré un ensemble de 6 instances très larges (présenté dans la partie expérimentale du chapitre 4). Nous avons retenue la version parallèle PGBS avec $T = 1$. De la même manière que la section précédente, nous avons considéré deux positions pour chaque instance (horizontale et verticale). Les solution optimales de ces instances ne sont pas connues, par conséquence, nous avons comparé les solutions obtenues par PGBS aux meilleures bornes inférieures de la littérature (HESGA^b (voir [39]) et SGBS (voir chapitre 3). Par ailleurs, nous avons limité le temps de résolution à 3000 secondes pour le solver CPLEX et à 900 secondes pour SGBS (l'algorithme correspondant est arrêté s'il ne donne pas de solution réalisable dans la limite fixée).

L'étude comparative est résumée dans la table 8.4 lorsqu'on applique Cplex, HESGA^b, SGBS et PGBS. La colonne 2 contient les meilleures solutions réalisables obtenues par l'un des algorithmes considérés. La colonne 3 montre la déviation, notée Gap, réalisée par Cplex. La colonne 4 donne la déviation de HESGA^b. La colonne 5 et 6 montrent, respectivement, la déviation de la solution produite par SGBS lorsque $\delta = 2$ et le temps de résolution. La colonne 9 rapporte la déviation de PGBS et les trois dernière colonnes (colonnes 10, 11 et 12) présentent, respectivement, le temps nécessaire à PGBS avec 6, 10 et 20 processeurs pour trouver la solution présentée. Finalement, nous avons utilisé le symbole "$\circ$" lorsque l'algorithme obtient la meilleure solution de la littérature (cette valeur correspond à la meilleure solution obtenue par Cplex, HESGA^b et SGBS) et le symbole "$\diamond$" lorsque l'algorithme ne termine pas le résolution avant 900 secondes.

De la table 8.4, on peut observer que :

1. Cplex résout un nombre limité d'instances. Il obtient 2 solutions (WL1V et WL3V) sur les 12 instances traitées et réalise une déviation de -464.67. On rappel que ces solutions sont obtenues en limitant le temps de résolution à 3000 secondes.
2. SGBS_4 (avec une largeur de faisceau $\delta = 4$) produit des solutions moyennes à cause de la limitation du temps de résolution (300 secondes). L'algorithme génère une déviation moyenne de -218070.67. Ce résultat est dû essentiellement à la difficulté des instances traitées, par conséquence, l'algorithme nécessite plus de temps pour explorer l'espace de recherche.
3. SGBS_2 résout toutes les instances dans un temps plus intéressant que Cplex, HESGA^b et SGBS_4. Le temps de résolution est égale à 334.24 secondes (le plus petit temps de résolution).
4. PGBS résout plus efficacement que Cplex, HESGA^b et SGBS_2. Il améliore toutes les meilleures solutions de la littérature. Dans ce cas, la moyenne de la déviation est de 10917.33.

   Concernant le temps de résolution de PGBS, nous avons analysé le temps nécessaire pour produire une solution, selon le nombre de processeurs utilisés par PGBS :

   (a) Le temps de résolution moyen de $\text{PGBS}_4^6(1)$ est plus petit que 334.24 secondes (temps de résolution moyen de SGBS_2) et des 3000 secondes nécessaires au solver CPLEX. Par rapport aux temps obtenus par SGBS_2 ; $\text{PGBS}_4^6(1)$ réduit le temps moyen de résolution

de 34.65%. Le speedup $S(6)$ et l'efficacité $F(6)$ de $\text{PGBS}_4^6(1)$ sont, respectivement, égales à 1.53 et 0.26.

(b) $\text{PGBS}_4^{10}(1)$ réduit de 18.01% le temps de résolution de $\text{PGBS}_8^6(1)$. Le speedup $S(10)$ et l'efficacité $F(10)$ sont égales, respectivement, à 1.87 et 0.19.

(c) Le temps de résolution de $\text{PGBS}_4^{20}(1)$ est plus intéressant. Il réduit le temps de 179.09 secondes ($\text{PGBS}_4^{10}(1)$) à 156.33 secondes. On remarque que la diminution est plus importante par rapport à la meilleure version de l'algorithme séquentiel (Il là réduit de 53.23% en moyenne). Le speedup $S(20)$ et l'efficacité $F(20)$ de $\text{PGBS}_4^{20}(1)$ sont égales, respectivement à 2.14 et 0.11.

		Cplex	HESGA[b]	*SGBS*				*PGBS* $\delta = 4$					
				$\delta = 2$		$\delta = 4$		$\eta = 6$		$\eta = 10$		$\eta = 20$	
#Inst.	Best	Gap	Gap	Gap	cpu	Gap	cpu	Gap	cpu	cpu		cpu	
UL1H	887393	-10711	-294	○	138.56	-187359	◇	**1993**	99.89	81.65		67.76	
UL2H	906237	-5314		○	310.13	-216139	◇	**1629**	190.87	157.67		129.76	
UL3H	1017575	-8544		○	347.89	-18810	◇	**83417**	220.76	189.67		134.65	
WL1H	688840	-1054		○	410.09	-168781	◇	**5259**	250.87	206.76		187.23	
WL2H	782566	-4538		○	454.76	-125136	◇	**1562**	280.65	202.54		198.6	
WL3H	899890	-2517		○	488.76	-486401	◇	**477**	310.76	281.65		276.65	
UL1V	891516	-2754		○	120.87	-316084	◇	**8583**	99.87	82.89		66.21	
UL2V	908982	-2382		○	210.41	-230231	◇	**1039**	145.67	114.65		99.07	
UL3V	1022483	-3595	-3595	○	320.87	-124832	◇	**5008**	199.98	176.98		160.65	
WL1V	694657	○		○	340.89	-177948	◇	**5337**	210.65	186.71		151.65	
WL2V	783836	-1687	-1687	○	410.78	-293071	◇	**5985**	287.65	199.98		182.04	
WL3V	881931	○		○	456.81	-272056	◇	**10719**	323.54	267.89		221.67	
Av_Gap		-3591.33	-464.67	0.00		-218070.67		**10917.33**					
Av_cpu					334.24		900.00		218.43	179.09		156.33	

TABLE 8.4 – Performance de l'algorithme parallèle PGBS sur le deuxième groupe d'instances.

8.2 Algorithme coopératif parallèle

Nous avons proposé une application parallèle de la recherche par faisceaux pour résoudre le problème de découpe contraint à deux dimensions et à deux niveaux. L'algorithme proposé exploite plusieurs processeurs pour élargir l'espace de recherche et permet, ainsi, d'augmenter la largeur du faisceau. Par ailleurs, L'algorithme fonctionne selon le paradigme maître/esclave et chaque type de processeur utilise une stratégie du meilleur d'abord dans ces développements. L'algorithme utilise le processeur maître pour gérer les synchronisations et la diversification des chemins de recherche. Afin d'éviter la surcharge du processeur maître et la génération d'un goulot d'étranglement, chaque processeur esclave utilise ses propres listes internes afin de développer des chemins de recherche.
Le caractère irrégulier du problème traité, nous a poussé à définir une politique dynamique de régulation de charge. En effet, cette politique (régulation de charge par seuil) permet à l'algorithme d'exploiter efficacement les processeurs esclaves même si elle augmente le nombre de communications entre les processeurs. Les résultats obtenus sur deux groupes d'instante, réputés dans la littérature comme étant, très difficiles, a donné des résultats encourageants et montre l'intérêt de l'utilisation du parallélisme dans les méthodes approchées.

Encouragé par les résultats précédents, nous proposons dans la suite de ce chapitre, une version parallèle de l'algorithme coopératif présenté dans le chapitre 4. L'algorithme parallèle proposé s'appuie sur les mêmes mécanismes de synchronisation et d'équilibrage de charge précédents. Par ailleurs, nous augmentons les performances de l'algorithme séquentiel en calculant en parallèle sur chaque nœud de l'arbre de recherche, la borne supérieure et la solution complémentaire. Le calcul parallèle de la borne supérieure et de la solution complémentaire permet de réduire considérablement le temps du filtrage et le calcul de la borne supérieure sur chaque nœud permet d'orienter, plus finement, la recherche d'une solution réalisable.

8.2.1 Analyse de l'algorithme séquentiel CGBS

Dans cette section, nous faisons une description de l'algorithme séquentiel SCGBS, présenté dans le chapitre 4 (noté SCGBS). L'algorithme séquentiel combine l'algorithme GBS et la procédure de remplissage BFP.

Le principe de BFP est le suivant :
Soit w, $w \leq W$, la hauteur du sous rectangle courant (L, w), r, $r \leq n$ le nombre de bandes optimales de hauteur $\overline{w}_j \leq w$, $S_{\overline{w}_j}$ l'ensemble de pièces de la bande $(L, \overline{w}_j)$, où $\overline{w}_j \leq w$ et $f_{\overline{w}_j}(L)$ la valeur de la solution associée à la bande $(L, \overline{w}_j)$.

Le plan de découpe associé au sous rectangle (L, w) de profit $h_L(w)$ est la solution optimale

du programme linéaire suivant :

$$(IP_{L,w}) = \begin{cases} h_L(W) = \max \sum_{j=1}^{r} f_w(L) y_i \\ \text{s.c. } \sum_{j=1}^{r} \overline{w}_j y_j \leq w \\ \sum_{j=1}^{r} x_{ij} y_j \leq b_i, i \in I \\ y_j \leq a_j, y_j \in N, j = 1, ..., r, \end{cases}$$

où x_{ij} représente le nombre d'occurrences de la i ème pièce dans la j ème bande de dimension $(L, \overline{w}_j)$ et

$$a_j = min\Big\{ \left\lfloor \frac{\overline{w}_p}{\overline{w}_j} \right\rfloor, min_{i \in f_{\overline{w}_j}} \left\lfloor \frac{b_i}{x_{ij}} \right\rfloor \text{ avec } x_{ij} > 0 \Big\}$$

Dans l'algorithme séquentiel SCGBS, un nœud u est représenté par deux sous rectangles ; (i) $((L, W-y), (L, y))$, où $(L, W-y)$ est la solution réalisable locale et (ii) (L, y) le sous rectangle à compléter. En d'autres termes, le sous rectangle $(L, W-y)$ est déjà remplie par GBS alors que (L, y) reste à remplir par les prochains développements de GBS. L'algorithme utilise une fonction d'évaluation qui combine trois solutions :
Soit w une découpe horizontale sur le sous rectangle complémentaire (L, y), avec $w \leq Y$ et $w \in \overline{w}_1, ..., \overline{w}_n$. Soit $v = ((L, W-y+w), (L, Y-w))$ un des nœuds résultat de cette découpe. L'algorithme coopératif combine les trois points suivants sur le nœud v.

1. $\hat{Z}_v^{Local}$ la valeur de la solution réalisable du sous rectangle $(L, W-y+w)$.
2. $BFP_{(L,Y-w)}$ une solution réalisable complémentaire du rectangle initial S.
3. $\overline{U}_{(L,y)}$ une borne supérieure du sous rectangle complémentaire $(L, y-w)$.

L'examen de ces trois points permet de remarquer l'indépendance entre le calcul de la borne supérieure et l'obtention de la solution réalisable complémentaire. Le calcul en parallèle de ces deux points présente un axe d'amélioration des performances de l'algorithme séquentiel.

8.2.2 L'algorithme

8.2.2.1 Le processeur maître

La figure 8.6 décrit les principales phases du traitement effectuées par le processeur maître. Il est composé de trois phases :

La *phase d'initialisation* est améliorée par rapport à PGBS ; Dans la phase 1.(c), l'algorithme calcul la borne supérieure complémentaire $UK_{(L,W)}$; pour le couple $(\overline{w}_j, f_{\overline{w}_j}^{hor})$, $j \in J$. Cette borne supérieure est calculée par programmation dynamique. La programmation dynamique permet d'avoir toutes les bornes supérieures $\mathcal{U}_{(L,Y)}$, $Y = 0, \ldots, W$,. Le calcul de la borne inférieure complémentaire $LB_{(L,W)}$; est réalisé par la procédure *MFP* qui est lancée parallèlement au calcul de la borne supérieure. Les deux bornes sont utilisées par la suite dans la phase de filtrage.

Dans la phase 1. (d), le filtrage est appliqué sur l'ensemble s. Cette phase consiste à : mettre à jour la meilleure solution réalisable courante $z^\star$ par la résolution de $\max_{j \in J} \left\{ \hat{z}_j^{Local} \right\}$.

`Entrée :` Instance FC2TDC.

`Sortie :` Solution de valeur $z^\star$.

1. *Phase d'initialisation*
   (a) $B_{master} = \left\{\mu = ((L,0),(L,W))\right\}$ et $z^\star = 0$.
   (b) Brancher sur μ pour générer toutes les bandes générales optimales associées à (L,W), par la résolution du problème $BK^{hor}_{L,\overline{w}_s}$, où $s = \text{argmax}\{\overline{w}_s \mid \overline{w}_s \geq \overline{w}_j,\ j \in J\}$.
   (c) Pour μ, calculer la borne supérieure correspondant au sous rectangle $(L,Y),\ Y = 0,\ldots,W$, par la résolution le problème de sac à dos $UK_{(L,W)}$ et calculer, en parallèle, la borne inférieure complémentaire $LB_{(L,W)}$ (utilisation de la procédure MFP).
   (d) Appliquer la phase de filtrage sur les s nœuds crées et les insérer dans B_{master}.
   (e) Soit *Active* le vecteur associé aux processeurs esclaves (initialement $\forall k,\ Active_k$:=`false`), telle que $Active_k$:=`false`, $k = 1,\ldots,\eta$, si le k-ème processeur est disponible, `true` sinon.
   (f) Assigner les $\alpha := \min\left\{s,\ \max\{\delta,\eta\}\right\}$ nœuds de B_{master} aux processeurs esclaves disponibles, c'est à dire,
   $\forall k \in \{1,\ldots,\alpha\}$, `Appeler Slave_Algo`$(\mu_k, B_k, \overline{B}_k, T)$ et $Active_k$ =`true`.
2. *Phase de synchronisation*
   Cas 1. Un sous-ensemble de processeurs esclaves envoie des nœuds :
      (a) Le processeur maître reçoit les nœuds $\overline{B}_k,\ k = 1,\ldots,\eta'$ (où $\eta' \leq \eta$).
      (b) $B_{master} = B_{master} \cup_{k=1}^{\eta'} \overline{B}_k$.
      (c) Appliquer la phase de filtrage sur la liste B_{master}.
   Cas 2. Le k-ème processeur termine la résolution :
      (a) $Active_k$ =`false` et mettre à jour la meilleure solution réalisable courante $z^\star$.
      (b) Appliquer la phase de filtrage sur B_{master}.
      (c) Si $B_{master} \neq \emptyset$, soit $\mu \in B_{master}$ le meilleur nœud sélectionné par la stratégie du meilleur d'abord.
      Assigner μ au processeur esclave disponible, nommé ℓ, et mettre $Active_\ell$ =`true`.
   Cas 3. La période T a écoulée :
      (a) Le processeur maître reçoit les nœuds des $\eta' \leq \eta$ processeurs actifs.
      (b) $\forall k \in \{1,\ldots,\eta'\}, Active_k$ =`false`.
      (c) $B_{master} = B_{master} \cup_{k=1}^{\eta'} B_k \cup_{k=1}^{\eta'} \overline{B}_k$.
      (d) Mettre à jour la meilleure solution réalisable courante $z^\star$.
      (e) Appliquer la phase de filtrage sur la liste globale B_{master} tel que $\mu \in B_{master}$ si et seulement si, $z^\star < \hat{z}^{Local}_\mu + \mathcal{U}_\mu$.
      (f) Si $B_{master} \neq \emptyset$, alors répéter l'étape 1.(f) de la phase d'initialisation.
3. *Phase d'arrêt*
   Quitter avec la meilleur solution réalisable $z^\star$ lorsque (a) $B_{master} = \emptyset$, (b) $Active_k = false, \forall k \in \{1,\ldots,\eta\}$ et (c) tous les queues de communication sont vides.

FIGURE 8.6 – Procédure globale du processeur maître dans PCGBS : `MasterPCGBS_Algo`.

Les traitements relatifs à la synchronisation, la diversification et l'arrêt de la résolution restent identiques à *PGBS*.

8.2.2.2 Les processeurs esclaves

La figure 8.7 décrit des phases principales de la procédure améliorée, notée `Slave_PCGBS`, utilisée

`Entrée` : Le contexte du nœud $\mu = ((L, W_\mu), (L, W - W_\mu))$ et la valeur de la période T.

`Sortie` : (i) La valeur de la meilleure solution locale $z_k^\star$ et les deux listes internes (B_k et $\overline{B}_k$), ou (ii) les nœuds de la liste secondaire $\overline{B}_k$.

1. *Phase d'initialisation*
   1. $B_k = \{\mu\},\ \overline{B}_k = \emptyset,\ z_k^\star = z^\star$ et soit $b'_i,\ i \in I$, les demandes résiduelles après filtrage des solutions partielles dans (L, W).
   2. Soit r le nombre de hauteurs distincts, tel que $\overline{w}_r \leq W - W_\mu$ et $\forall j \in \{1, \ldots, r\},\ \overline{w}_r \geq \overline{w}_j$. Créer les r bandes générales optimales associées à μ, par résolution (programmation dynamique) du problème $BK^{hor}_{L,\overline{w}_r}$ avec les valeurs des demandes $b'_i,\ i \in I$.
   3. Calculer les bornes supérieures complémentaires associées à $(L, Y),\ Y = 0, \ldots, W - W_\mu$, par résolution de UK_μ.
2. *Phase itérative*
   1. Sélectionner de B_k le nœud $\mu = ((L, W - Y), (L, Y))$ avec la meilleure évaluation globale $z_\mu^{Global} = \hat{z}_\mu^{Local} + \mathcal{U}_\mu$.
   2. Supprimer le nœud μ de B_k.
   3. Brancher sur μ et générer les bandes valides par résolution du problème $BK^{hor}_{L,\overline{w}_r}$, où $\overline{w}_r$ dénote la plus haute bande (avec le plus grand indexe) qui peut être placer dans le sous rectangle (L, Y).
   4. Pour chaque nouveau nœud $\nu = \Big((L, W - Y + \omega), (L, Y - \omega)\Big),\ \omega = \overline{w}_1, \ldots, \overline{w}_r$,
      (a) Effectuer une évaluation globale basée sur la borne inférieure $\hat{z}_\nu^{Local}$ et la borne supérieure complémentaire $\mathcal{U}_\nu$ et en parallèle appliquer la solution complémentaire par `MFP de` $\Big((L, W - Y + \omega), (L, Y - \omega)\Big)$ `et mettre à jour la valeur du nœud` ν.

      `(b) Mettre à jour la meilleure solution courante :` $z_k^\star = \max\Big\{z^\star,\ \hat{z}_\nu^{Local}\Big\}$.
   5. Appliquer la phase de filtrage sur les bandes générées ; soit $\overline{B}_k$ l'ensemble des nœuds élites ($\min\{\rho, |\overline{B}_k|\}$ meilleure nœuds de $\overline{B}_k$.)
   6. Si $|B_k \cup \overline{B}_k| > \xi$, alors copier les meilleurs $\xi - |B_k|$ nœuds de $\overline{B}_k$ à B_k et les supprimer de $\overline{B}_k$; mettre $B_k = B_k \cup \overline{B}_k$ sinon.
   7. Si $\overline{B}_k \neq \emptyset$, alors envoyer le contenu de $\overline{B}_k$ au processeur maître.
   8. Supprimer tous les éléments de $\overline{B}_k$.
3. *Arrêter ou répéter la recherche :*
   1. Si $B_k = \emptyset$ ou la période T est écoulée, alors
      (a) $B_k = \emptyset$: envoyer la meilleure solution locale obtenue $z_k^\star$ au processeur maître.
      (b) T a écoulée : envoyer la meilleure solution local obtenue $z_k^\star$ et les deux listes internes (B_k et $\overline{B}_k$).
   2. Sinon, répéter *étape itérative 2.*

FIGURE 8.7 – Description des étapes du k-ème processeur esclave dans PCGBS `SlavePCGBS_Algo`.

par les processeurs esclaves $k,\ k = 1, \ldots, \eta$, où η présente le nombre de processeurs esclaves disponibles. L'algorithme ressemble à l'algorithme de recherche en faisceau parallèle avec quelques améliorations. Comme montré dans la figure 8.7, l'algorithme est composé de 3 étapes :

La *phase d'initialisation*, est identique à la phase d'initialisation de l'algorithme parallèle décrite

dans la section 8.1.4.2. En effet, la procédure MFP est inutile dans cette phase. L'évaluation des nœuds et le calcul de la solution complémentaire seront effectués dans la phase suivante.

La *phase itérative* commence (étape 2.1) par sélectionner le meilleur nœud, noté $\mu = ((L, W - Y), (L, Y))$, de la premier liste interne B_k et le supprime de celle-ci (étape 2.2). Dans l'étape 2.3, r $(r \leq s \leq n)$ bandes générales sont crées par la résolution du problème BK^g_{L,w_r}, où

$$r = \max\Big\{w_r \mid w_r \geq w_j,\ w_j \leq W_\mu,\ j \in\ J\Big\}.$$

Dans l'étape 2.4, l'algorithme calcul pour chaque bande (nœud) ν les bornes inférieure ($\hat{z}^{Local}_\nu$) et supérieure complémentaire ($\mathcal{U}_\nu$). Ces deux bornes sont calculées en parallèle. L'algorithme continu par évaluer les nœuds pour les futures développements ($\hat{z}^{Global}_\nu$;). La borne inférieure complémentaire ($\hat{z}^{Local}_\nu$) est obtenue par l'application de la procédure `MFP` décrite dans la figure 8.2.2.2.

`Entrée` : Un nœud local ν.
`Sortie` : Une solution d'approximation $\hat{z}^{Local}_\nu$.

1. Soit $\nu = \big[\big((L, W - Y), (L, Y)\big);\ v\big]$ le nœud à évaluer et la valeur de sa solution locale $\hat{z}^{Local}_\mu$.
   Poser $d^{rest} := v$ et $W^{rest} := Y$.
2. Répéter
   (a) Soit $\hat{z}$ La solution du problème de sac à dos suivant :

$$(BK^{hor}_{L,w_r}) \begin{cases} \hat{z} = \max & \displaystyle\sum_{i \in S_{L,w_r}} c_i x_{ij} \\ \text{subject to} & \displaystyle\sum_{i \in S_{L,w_r}} l_i x_{ij} \leq L \\ & x_{ij} \leq v_i,\ x_{ij}\ \in N,\ i \in S_{L,w_r}, \end{cases}$$

   Où w_r dénote la plus grande hauteur des pièces qui peuvent entrer dans la bande (L, w_r) et S_{L,w_j} l'ensemble des pièces qui peuvent entrer dans $(L, w_r),\ w_r \leq W^{rest}$.
   (b) Poser $\hat{z}^{Local}_\mu := \hat{z}^{Local}_\mu + \hat{z}$.
   (c) Mettre à jour le vecteurs des demandes d^{rest}.
   (d) Poser $W^{rest} := W^{rest} - w_r$.
   Jusqu'à ce que $(\forall i \in I,\ w_i > W^{rest})$ ou $(\forall i \in I,\ d^{rest}_i = 0)$.

FIGURE 8.8 – Procédure de remplissage modifiée : `MFP`.

Dans l'étape 2.5, l'algorithme réalise une sélection sur les nœuds crées et les transfert à la deuxième liste interne $\overline{B}_k$, et dans l'étape 2.6 le filtrage est effectué sur $\overline{B}_k$ et retourne les meilleurs $\min\{\delta, |\overline{B}_k|\}$ nœuds qui réalisent la meilleure évaluation globale (selon l'équation (4.2)). L'étape 2.7 vérifie le nombre de nœuds dans la première liste interne $(B_k \cup \overline{B}_k)$, cette étape concerne la régulation de charge entre les processeurs esclaves.

La phase d'arrêt est composée de deux parties. Dans la première partie (étape 3.1), le processeur esclave arrête la résolution, si la liste B_k est vide ou si la période de synchronisation T a écoulée.

Dans le premier cas (liste B_k vide), le processeur esclave envoie la valeur de la solution réalisable locale courante au processeur maître. Si la période T a écoulée, le processeur esclave envoi, en plus de la valeur de la solution réalisable local courante, tous les nœuds des listes internes.

8.2.2.3 La communication entre les processeurs

La communication entre le processeur maître et les processeurs esclaves est réalisée de la même manière que le premier algorithme présenté dans ce chapitre, cependant le calcul des deux bornes supérieure et inférieure complémentaires est réalisé en parallèle et le résultat est géré avec deux nouvelles queues de communications.

Durant la résolution locale sur chaque processeur esclave, deux procédures indépendantes, UB_{proc} et LB_{proc}, sont lancées en parallèle. La première est utilisée pour calculer la borne supérieure complémentaire et la deuxième pour obtenir la borne inférieure. Chaque procédure utilise deux queues de communication pour recevoir le sous problème à traiter et pour transmettre le résultat (voir la figure 8.9).

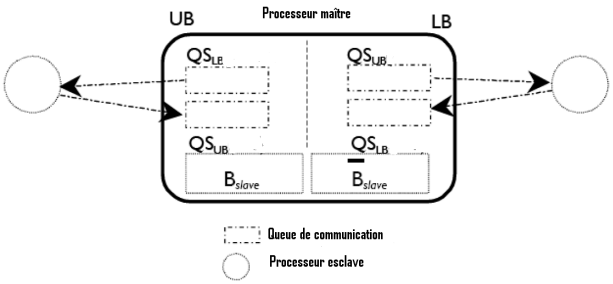


FIGURE 8.9 – Illustration du mécanisme de communication utilisé par les processeurs esclaves dans PCGBS.

8.2.3 Résultats numériques

Dans cette section, on évalue la performance de l'algorithme proposé (noté PCGBS) sur les deux groupes d'instances précédente. De la même manière que les résultats numériques précédents, nous avons varié la valeur associée au paramètre δ (largeur du faisceau) dans le but d'évaluer la qualité des solutions obtenues et la performance en terme de temps de résolutions.

8.2.3.1 Premier groupe d'instances

Dans un premier temps, nous avons mené notre étude sur le premier groupe d'instances. Nous avons lancé deux résolution pour chaque instance : la premier correspond à une première découpe horizontale et la deuxième à une première découpe verticale (les noms des instances est préfixé par

H ou V, selon la position de la première découpe). Pour la plupart des instances considérées, la solution optimale n'est pas connue, Nous avons, donc, comparé les solutions de PGBS aux meilleures solutions de la littérature (voir Hifi et al [42]) en limitant le temps de résolution de SCGBS à 300 secondes pour ces instances. L'analyse des résultats obtenus est le suivant :

La Table 8.5 rapporte les résultats obtenus sur le premier groupe d'instances. La colonne 1 montre le nom de l'instance et la colonne 2 contient la meilleure solution connue ou l'optimum s'il existe (les instances dont la solution optimale est connue, sont signalées par le symbole "$*$"). Les colonnes 3 et 4 montrent respectivement, la solution obtenue par l'algorithme séquentiel SCGBS avec $\delta = 4$ et le temps de résolution en secondes. La colonne 5 et 6 donnent la solution obtenue par SCGBS $\delta = 16$, ainsi que le temps d'obtention de cette solution s'il est inférieur à 300 secondes (Le temps est signalé par le symbole "$-$" dans le cas où la résolution a été arrêté dans les 300 secondes). La colonne 7 et 8 montrent les meilleurs résultats de l'algorithme parallèle PCGBS et le temps de résolution. Les deux dernières lignes de la table 8.5 montrent respectivement, le nombre d'instances dont la solution a été amélioré par l'algorithme correspondent (noté NB_{Best}) et le temps résolution moyen pour toutes les instances.
La table 8.5 montre que : SCGBS_{16} obtient des résultats médiocres par rapport à SCGBS_4. En effet, 300 secondes ne sont pas suffisantes pour l'algorithme séquentiel avec une largeur de faisceau $\delta = 16$ pour certaines instances. Cette version de l'algorithme n'améliore la solution d'aucune instance et fait augmenter le temps de résolution. Le temps de résolution augmente à 143.63 secondes en moyenne.

PCGBS_{16} est plus efficace que les deux versions de l'algorithme séquentiel SGBS. Il améliore les solutions obtenues pour 24 instances (54.54% des instances) et améliore les meilleures solutions connues de la littérature. On remarque également que parmi les instances non améliorées par PCGBS_{16}, 11 ont des solutions optimales déjà connues.

La Table 8.6 détaille les résultats de l'algorithme parallèle. En effet, les solutions obtenues par l'algorithme PCGBS sont différentes selon le nombre de processeurs utilisés dans la résolution. La colonne 1 montre le nom de l'instance et la colonne 2 contient la meilleure solution obtenue par SCGBS (les instances dont la solution optimale est connue, sont signalées par le symbole "$*$").
Les colonnes 3 et 4 montrent les résultats de PCGBS en utilisant 6 processeurs ($\eta = 6$), les colonnes 5 et 6 montrent les résultats de PCGBS avec 10 processeurs et en fin les colonnes 7 et 8 illustrent les résultats de PCGBS avec 20 processeurs.

Soit $\mathrm{PCGBS}_{16}^{\eta}$ la version de l'algorithme parallèle avec η processeurs, $S(\eta)$ le speedup et $F(\eta)$ l'efficacité. Nous avons étudié l'impact du nombre de processeurs utilisés par PGBS sur sa performance.

1. Le temps moyen consommé par PCGBS_{16}^{6} est inférieure des 143.63 secondes, consommées en moyenne par $\mathrm{SCGBS}_1 6$ mais il est nettement plus important des 3.58 secondes consommées par SCGBS_4. Selon le temps de résolution moyen réalisé par SGBS_4 et PGBS_{16}^{6} ; PGBS_{16}^{6} et 32 fois plus long $\mathrm{SCGBS}_1 6$. Cela indique bien, que la largeur du faisceau dans la résolution est l'élément qui influence le plus le temps de résolution.

	SCGBS					PCGBS $\delta = 16$	
		$\delta = 4$		$\delta = 16$		$\delta = 16$ et $\eta = 20$	
#Inst.	Opt/Best	Sol	cpu	Sol	cpu	Sol	cpu
nice25H*	860829	860829	0.15	860829	2.76	860829	1.01
nice50H	913738	913738	0.31	913738	6.21	913738	3.21
nice100H	911122	911122	0.95	911122	17.67	**911203**	8.98
nice200H	932686	932686	0.98	932686	39.67	940040	22.01
nice500H	952015	952015	0.89	952015	–	**953608**	141.98
path25H*	892765	892765	0.16	892765	1.16	892765	0.01
path50H	750215	750215	0.55	750215	3.55	750215	0.01
path100H	888578	888578	0.08	888578	9.99	**888991**	2.01
path200H	889174	889174	1.08	889174	–	**890223**	101.21
path500H	885112	885112	4.98	885112	–	**887667**	99.01
nice25PH*	441993	441993	0.02	441993	2.02	441993	0.21
nice50PH	453460	453460	0.02	453460	6.02	453460	1.99
nice100HP	459214	459214	0.03	459214	–	**459781**	130.76
nice200PH	455727	455727	0.12	455727	–	**456888**	89.87
nice500PH	455036	455036	1.04	455036	–	**459203**	141.76
nice1000PH	454089	454089	2.32	454089	–	**459098**	107.76
path25PH*	464721	464721	0.01	464721	1.02	464721	0.07
path50PH	391931	391931	0.01	391931	4.01	**391946**	0.87
path100PH	456300	456300	0.09	456300	17.89	**457997**	6.78
path200PH	452402	452402	1.04	452402	–	**459856**	130.98
path500PH	434478	434478	20.02	434478	–	**440221**	169.05
path1000PH	434966	434966	50.21	434966	–	**441242**	117.98
nice25V*	834157	834157	0.01	834157	4.01	834157	0.98
nice50V	868781	868781	0.67	868781	8.67	868781	3.23
nice100V	911958	911958	1.09	911958	39.75	911958	15.89
nice200V	935290	935290	3.89	935290	–	**936690**	91.97
nice500V	960767	960767	7.98	960767	–	**965604**	160.98
path25V*	699707	699707	0.01	699707	2.81	699707	0.01
path50V*	924908	924908	0.34	924908	4.34	924908	1.03
path100V*	754753	754753	2.98	754753	29.98	754753	0.76
path200V	836022	836022	2.99	836022	–	**836754**	140.87
path500V	808376	808376	3.98	808376	–	**808511**	199.72
nice25PV*	428662	428662	0.21	428662	3.22	428662	0.01
nice50PV	426269	426269	0.43	426269	19.43	426269	3.98
nice100PV	451851	451851	4.33	451851	64.33	**455406**	23.18
nice200PV	454754	454754	0.99	454754	–	**454889**	130.42
nice500PV	457381	457381	5.98	457381	–	**457403**	120.03
nice1000PV	459142	459142	2.78	459142	–	**459907**	160.95
path25PV*	356088	356088	0.04	356088	2.16	356088	0.01
path50PV*	487447	487447	0.01	487447	9.01	487447	2.01
path100PV*	386033	386033	0.98	386033	19.98	386033	6.21
path200PV	418024	418024	1.98	418024	–	**422706**	79.22
path500PV	381860	381860	3.76	381860	–	381860	88.98
path1000PV	425768	425768	26.98	425768	–	425768	201.76
Nb_Best						24	
Av_cpu			3.58		143.63		61.58

TABLE 8.5 – Résultats de l'algorithme parallèle PCGBS et de l'algorithme séquentiel SCGBS. Le symbole "–" signifie que le temps de résolution de SGBS a dépassé 300 secondes.

2. Le temps moyen consommé par PCGBS_{16}^{6} est inférieure au temps consommé en moyenne par $\text{SGBS}_1 6$. PCGBS_{16}^{6} améliore 24 des instances traités (sur 44 instances). Le speedup $S(6)$ et l'efficacité $F(6)$ sont respectivement égale à 1.25 et à 0.20.

3. Le temps de résolution moyen de PCGBS_{16}^{10} diminue de 75.9% par rapport à PCGBS_{16}^{6}. Dans ce cas, le speedup $S(10)$ et l'efficacité $F(10)$ reviennent respectivement, égales à 1.64 et à 0.16.

4. Le temps de résolution de PCGBS_{16}^{20} est plus intéressant. Il diminue de 25.48 secondes (temps de résolution moyen de $\text{PGBS}_1 6^{10}$) à 61.58 secondes. Dans ce cas, le gain en temps de calcul devient plus important par rapport à la meilleure version de l'algorithme séquentiel, il améliore de 57.12% le temps moyen de résolution avec un speedup $S(20)$ et une efficacité

$F(20)$ égales, respectivement, à 2.33 et 0.11.

		PCGBS					
		$\eta = 6$		$\eta = 10$		$\eta = 20$	
#Inst.	Sol. SCGBS	Sol	cpu	Sol	cpu	Sol	cpu
nice25H*	860829	–	2.02	–	1.56	–	1.01
nice50H	913738	913738	5.01	913738	4.01	913738	3.21
nice100H	911122	**911203**	12.99	911203	11.89	911203	8.98
nice200H	932686	**940040**	28.21	940040	27.11	940040	22.01
nice500H	952015	**953608**	205.81	953608	167.87	953608	141.98
path25H*	892765	–	1.02	–	0.07	–	0.01
path50H*	750215	–	2.02	–	1.01	–	0.01
path100H	888578	**888991**	3.03	888991	2.89	888991	2.01
path200H	889174	**890223**	199.09	890223	130.02	890223	101.21
path500H	885112	**887667**	201.96	887667	160.56	887667	99.01
nice25PH*	441993	–	1.01	–	0.52	–	0.21
nice50PH	453460	453460	4.01	453460	3.00	453460	1.99
nice100HP	459214	**459781**	231.89	459781	201.21	459781	130.76
nice200PH	455727	**456888**	199.98	456888	110.89	456888	89.87
nice500PH	455036	**459203**	233.38	459203	189.97	459203	141.76
nice1000PH	454089	**459098**	220.87	459098	180.86	459098	107.76
path25PH*	464721	–	0.09	–	0.08	–	0.07
path50PH	391931	**391946**	2.33	391946	1.87	391946	0.87
path100PH	456300	**457997**	11.76	457997	8.71	457997	6.78
path200PH	452402	**459856**	271.77	459856	199.08	459856	130.98
path500PH	434478	**440221**	299.89	440221	220.67	440221	169.05
path1000PH	434966	**440001**	268.98	**441242**	241.87	441242	117.98
nice25V*	834157	–	3.01	–	2.01	–	0.98
nice50V	868781	868781	6.86	868781	5.45	868781	3.23
nice100V	911958	911958	28.04	911958	22.48	911958	15.89
nice200V	935290	936690	222.04	936690	176.54	936690	91.97
nice500V	960767	**965604**	234.89	965604	201.98	965604	160.98
path25V*	699707	–	1.01	–	0.09	–	0.01
path50V*	924908	–	3.32	–	2.13	–	1.03
path100V	754753	754753	2.78	754753	1.87	754753	0.76
path200V	836022	**836754**	267.05	836754	199.98	836754	140.87
path500V	808376	808376	280.45	**808511**	240.42	808511	199.72
nice25PV*	428662	428662	2.02	428662	1.01	428662	0.01
nice50PV	426269	426269	6.34	426269	4.99	426269	3.98
nice100PV	451851	**455406**	40.21	455406	32.11	455406	23.18
nice200PV	454754	**454889**	286.07	454889	200.98	454889	130.42
nice500PV	457381	**457403**	244.76	457403	180.67	457403	120.03
nice1000PV	459142	**459201**	222.02	**459907**	203.32	459907	160.95
path25PV*	356088	–	2.02	–	1.01	–	0.01
path50PV*	487447	–	5.04	–	3.02	–	2.01
path100PV*	386033	–	12.65	–	9.42	–	6.21
path200PV	418024	**422706**	190.56	422706	110.43	422706	79.22
path500PV	381860	381860	280.76	381860	101.08	381860	88.98
path1000PV	425768	425768	291.76	425768	263.76	425768	201.76
Nb_Best			24		24		24
Av_cpu			114.56		87.06		61.58

TABLE 8.6 – Résultats de l'algorithme parallèle PCGBS avec 6, 10 et 20 processeurs. Le symbole "–" signifie que la solution optimale est obtenue par l'algorithme considéré. La valeur en italique présente une amélioration de la solution obtenue par SCGBS et le symbole ∘ signifie que l'algorithme PCGBS améliore la solution rapportée dans la troisième colonne.

Comme nous l'avons mentionné dans la section 8.1.3.2, PCGBS utilise un paramètre T de synchronisation. Cette synchronisation peut être considérée comme une stratégie de diversification qui sert à explorer d'autres régions de l'espace de recherche. Dans ce sens, les résultats présentés dans la table 8.5 sont obtenues en fixant la valeur de la période T à 1 secondes. Ce choix a été réalisé après plusieurs tests. Les résultats obtenus en variant la valeur de la période T de 1 secondes à 3 secondes sont résumés dans la table 8.8.

La table 8.7 montre les résultats de PCGBS en variant la valeur de la période T de 1 secondes à 3 secondes. La colonne 1 montre le nom de l'instance et la colonne 2 contient la meilleure

solution obtenue par PCGBS (les instances dont la solution optimale est connue, sont signalées par le symbole "$*$"). Pour chaque valeur de T, la table 8.7 rapporte, respectivement, les solutions obtenues et le temps de résolution de PCGBS_{16}^{6}, PCGBS_{16}^{10} et PCGBS_{16}^{20}.

La colonne 2 de la table 8.7 rapporte la meilleure solution (et la nouvelle solution) extraite de la table 8.5. Les colonne 3 à 8 montrent le résultat pour $T = 1$ avec différent nombre de processeurs : le temps de résolution (noté, cpu) et le nombre de sauts (noté Nj) qui correspond au nombre de synchronisations réalisées pendant la résolution. Les autres colonnes (de 9 à 20) montrent les mêmes résultats pour les différentes valeurs de T.

Dans la table 8.8, la ligne 1 montre le nombre moyen de sauts (Av_Nj), la ligne 2 rapporte le temps moyen de résolution (Av_cpu) et la ligne 3 rapporte le nombre de fois ($\text{nb}_{Best/Opt}$) que PCGBS a amélioré la meilleure solution connue (présentée dans la colonne 7 de la table 8.5).

Soit $\text{PCGBS}_{16}^{\eta}(T)$ l'algorithme parallèle exécuté avec les paramètres η (nombre de processeurs esclaves), T (la valeur associée à la période de synchronisation) et la largeur du faisceau $\delta = 16$. L'analyse des résultats présentés dans la table 8.8 révèle que $\text{PCGBS}_{16}^{\eta}(T = 1)$, pour $\eta = 6, 10, 20$, est plus performant que les autres versions de l'algorithme. par ailleurs, la version $\text{PCGBS}_{16}^{\eta}(2)$ améliore les solutions des 44 instances, mais $\text{PCGBS}_{16}^{\eta}(1)$ est plus rapide que $\text{PCGBS}_{16}^{\eta}(2)$. On remarque que , $\text{PCGBS}_{16}^{\eta}(3)$ a amélioré 38 instances, et le temps de résolution est globalement plus important (il varie de 118.69 à 172.76). On peut noter que $\text{PCGBS}_{16}^{\eta}(1)$ reste favorable par rapport à $\text{PCGBS}_{16}^{\eta}(2)$ car le rapport entre les qualités des solutions obtenues et le temps de résolution donne l'avantage à $\text{PCGBS}_{16}^{\eta}(1)$; dans ce sens, $\text{PGBS}_{16}^{\eta}(1)$ réalise un temps moyen de résolution qui varie de 61.58 à 114.56 secondes, alors que le temps de résolution de $\text{PGBS}_{16}^{\eta}(2)$ varie de 91.94 à 116.81.

8.2.3.2 Le deuxième groupe d'instances

Dans cette section, nous avons considéré un ensemble de 6 instances très larges. Nous avons retenue la version parallèle PGBS avec $T = 1$. De la même manière que la section précédente, nous avons considéré deux positions pour chaque instance (horizontale et verticale). Les solutions optimales de ces instances ne sont pas connues, par conséquence, nous avons comparé les solutions obtenues par PCGBS au meilleures bornes inférieures de la littérature (HESGA^{b} (voir [39]) et SCGBS (voir chapitre 4). Par ailleurs, nous avons limité le temps de résolution à 900 secondes pour SCGBS.

L'étude comparative est résumée dans la table 8.9 lorsqu'on applique SCGBS et PCGBS. La colonne 2 contient les meilleures solutions réalisables obtenues par l'algorithme séquentiel. La colonne 3 et 4 montre, respectivement, la solution réalisée par $\text{SCGBS}_{\delta} = 2$ et le temps de résolution. La colonne 5 et 6 donnent la solution obtenue par $\text{SCGBS}_{\delta} = 4$ et le temps de résolution. Les colonnes 7 à 11 montrent, respectivement, la solution produite par $\text{PCGBS}_{\delta} = 8$ et le temps de résolution lorsque $\eta = 6$, $\eta = 10$ et $\eta = 20$. La colonne 7 rapporte les solutions de l'algorithme parallèle, alors que la colonne 8 montre le gain, noté Gap de la solution produite par l'algorithme parallèle, par rapport à la meilleure solution de l'algorithme séquentielle.

#Inst.	Best	T=1 Sec						T=2 Sec						T=3 Sec					
		$\eta = 6$		$\eta = 10$		$\eta = 20$		$\eta = 6$		$\eta = 10$		$\eta = 20$		$\eta = 6$		$\eta = 10$		$\eta = 20$	
		N_j	cpu	N_j	cpu	N_j	cpu	N_j	cpu	N_j	cpu	N_j	cpu	N_j	cpu	N_j	cpu	N_j	cpu
nice25H*	860829	2	2.02	0	1.56	0	1.01	1	1.89	0	0.78	0	0.51	0	4.89	0	2.99	0	1.98
nice50H*	913738	6	5.01	5	4.01	3	3.21	3	9.51	2	4.01	1	2.61	1	7.81	0	4.01	0	2.61
nice100H	911203	16	12.99	13	11.89	11	8.98	6	14.99	5	10.95	4	8.49	4	13.99	2	12.95	1	10.00
nice200H	940040	32	28.21	28	27.11	25	22.01	13	29.19	11	22.56	10	11.01	10	32.19	9	39.56	5	19.01
nice500H	953598	251	205.81	205	167.87	173	141.98	103	228.91	84	199.94	71	140.99	90	240.81	80	240.94	50	176.99
path25H*	892765	0	1.02	0	0.07	0	0.01	0	0.51	0	0.04	0	0.01	0	0.51	0	1.99	0	1.98
path50H*	750215	0	2.02	0	1.01	0	0.01	0	1.01	0	0.51	0	0.01	0	1.01	0	1.98	0	1.65
path100H	888991	4	3.03	4	2.89	0	2.01	2	5.52	1	2.45	0	1.01	1	8.52	0	2.45	0	1.01
path200H	890223	243	199.09	159	130.02	123	101.21	100	217.55	65	180.01	51	103.61	89	238.55	55	190.01	43	160.61
path500H	887664	246	201.96	196	160.56	121	99.01	101	212.98	80	188.28	50	103.51	80	232.98	61	206.28	50	199.51
nice25PH*	441993	0	1.01	1	0.52	0	0.21	0	0.51	0	0.26	0	0.11	0	2.98	0	1.26	2	1.98
nice50PH*	453460	5	4.01	3	3.00	0	1.99	2	2.01	1	2.50	0	1.00	0	2.01	0	2.50	0	1.00
nice100PH	459214	283	231.89	245	201.21	160	130.76	116	235.95	101	205.61	65	149.38	90	290.95	66	340.61	55	199.38
nice200PH	456888	244	199.98	135	110.89	110	89.87	100	102.99	55	112.45	45	99.94	90	280.99	43	180.45	32	123.94
nice500PH	459001	285	233.38	232	189.97	173	141.76	117	268.69	95	204.99	71	144.88	100	320.69	80	299.99	56	190.88
nice1000PH	459076	269	220.87	221	180.86	131	107.76	110	240.44	90	199.43	54	116.88	91	290.44	80	320.43	67	206.88
path25PH*	464721	0	0.09	0	0.08	0	0.07	0	0.05	0	0.04	0	0.04	0	2.99	0	2.87	0	2.08
path50PH*	391931	3	2.33	0	1.87	1	0.87	1	3.17	0	0.94	0	0.44	0	3.17	0	2.99	0	2.04
path100PH	457997	13	11.76	8	8.71	5	6.78	5	11.88	3	7.36	2	6.39	2	1122.00	1	8.36	0	6.39
path200PH	459856	332	271.77	243	199.08	160	130.98	136	289.89	100	213.54	65	137.49	100	289.89	90	340.54	65	337.49
path500PH	440167	366	299.89	269	220.67	206	169.05	150	300.94	110	268.34	85	170.53	121	390.14	95	379.34	71	370.53
path1000PH	440001	328	268.98	295	241.87	144	117.98	134	300.49	121	275.94	59	179.99	117	376.18	112	399.94	89	379.99
nice25V*	834157	3	3.01	0	2.01	1	0.98	1	2.51	0	1.01	0	0.49	0	2.51	0	1.01	0	1.01
nice50V*	868781	6	6.86	7	5.45	3	3.23	2	5.43	3	6.73	1	2.62	0	3.93	1	8.73	0	6.62
nice100V	911958	34	28.04	27	22.48	15	15.89	14	29.02	11	23.24	6	13.95	9	32.08	8	33.24	4	23.95
nice200V	936690	271	222.04	215	176.54	112	91.97	111	228.02	88	198.27	46	99.99	95	239.02	70	223.27	40	200.99
nice500V	965604	287	234.89	246	201.98	196	160.98	117	137.45	101	210.99	80	180.49	96	261.45	70	234.99	39	199.49
path25V*	699707	0	1.01	0	0.09	0	0.01	0	0.51	0	0.05	0	0.01	0	0.91	0	0.50	0	2.98
path50V*	924908	3	3.32	3	2.13	0	1.03	1	3.66	1	3.07	0	0.52	0	3.66	0	3.07	0	2.98
path100V*	754753	3	2.78	0	1.87	1	0.76	1	3.39	0	0.94	0	0.38	0	3.39	0	0.94	0	3.76
path200V	836754	326	267.05	244	199.98	172	140.87	134	299.53	100	209.99	70	149.44	100	320.53	98	377.99	60	299.44
path500V	808376	342	280.45	293	240.42	244	199.72	140	290.23	120	246.21	100	210.86	107	360.23	91	356.00	80	310.86
nice25PV*	428662	2	2.02	0	1.01	0	0.01	1	2.01	0	0.51	0	0.01	0	3.01	0	2.21	0	2.89
nice50PV*	426269	5	6.34	6	4.99	3	3.98	2	5.17	2	4.50	1	2.99	0	5.17	0	4.50	0	3.99
nice100PV	455406	49	40.21	39	32.11	28	23.18	20	53.11	16	35.06	12	27.59	13	83.11	9	45.06	6	37.59
nice200PV	454889	349	286.07	245	200.98	159	130.42	143	269.04	100	202.49	65	160.21	102	367.02	99	387.49	67	360.21
nice500PV	457381	299	244.76	220	180.67	146	120.03	122	249.38	90	190.34	60	179.02	103	379.38	80	363.34	50	349.02
nice1000PV	459142	271	222.02	248	203.32	196	160.95	111	248.01	102	206.66	80	180.48	100	388.01	89	377.66	70	327.48
path25PV*	356088	0	2.02	0	1.01	0	0.01	0	2.01	0	0.51	0	0.01	0	4.98	0	2.90	0	4.76
path50PV*	487447	5	5.04	3	3.02	0	2.01	2	5.52	1	2.51	0	1.01	0	5.52	0	2.51	0	3.89
path100PV*	386033	15	12.65	11	9.42	5	6.21	6	26.33	5	11.71	2	7.11	2	14.33	1	14.71	0	4.78
path200PV	422705	232	190.56	135	110.43	97	79.22	95	198.28	55	113.22	40	92.61	71	229.28	54	180.22	27	149.61
path500PV	381860	343	280.76	123	101.08	109	88.98	140	299.38	51	104.54	44	89.49	99	340.38	50	174.54	32	156.49
path1000PV	425768	356	291.76	322	263.76	246	201.76	146	302.88	132	171.88	101	204.88	106	399.88	97	389.88	75	371.88

TABLE 8.7 – Impact de la variation de la période T et du nombre de processeurs utilisés η sur les performances de l'algorithme parallèle PCGBS.

	T=1 Sec			T=2 Sec			T=3 Sec		
	$\eta = 6$	$\eta = 10$	$\eta = 20$	$\eta = 6$	$\eta = 10$	$\eta = 20$	$\eta = 6$	$\eta = 10$	$\eta = 20$
Av_N_j	139	106	75	57	43	31	45	36	26
Av_cpu	114.56	87.06	61.58	116.81	91.94	67.79	172.76	140.16	118.69
$\text{nb}_{Best/Opt}$	44	44	44	44	44	44	38	38	38

TABLE 8.8 – Résumé des résultats de la variation de la période T et du nombre de processeurs utilisés η sur les performances de l'algorithme parallèle PCGBS.

#Inst.	Sol.SCGBS	SCGBS				PCGBS $\delta = 8$				
		$\delta = 2$	cpu	$\delta = 4$	cpu	Sol.	Gap	cpu /$\eta = 6$	cpu /$\eta = 10$	cpu /$\eta = 20$
UL1H	887393	887393	114.00	876682	–	**889401**	2008	91.81	76.15	60.71
UL2H	906712	906712	245.24	900923	–	**907972**	1260	170.17	140.07	112.06
UL3H	1021124	1021124	279.58	1009031	–	**1101201**	80077	201.06	149.77	119.95
WL1H	687786	683656	322.21	687786	–	**694172**	6386	220.97	189.16	139.23
WL2H	780018	780018	314.74	778028	–	**784189**	4171	241.55	199.04	153.60
WL3H	899890	899890	412.76	897373	–	**900449**	559	303.06	290.35	254.75
UL1V	891516	891516	100.01	888762	–	**900199**	8683	98.87	81.99	65.81
UL2V	908982	908982	115.54	906600	–	**910021**	1039	125.77	102.05	77.07
UL3V	1022483	1022483	210.09	1018888	–	**1027491**	5008	179.09	139.58	118.95
WL1V	697164	697164	301.12	694657	–	**699998**	2834	200.85	176.81	121.95
WL2V	783836	783836	385.90	782149	–	**789997**	6161	277.25	187.68	163.04
WL3V	890696	890696	352.24	881931	–	**892709**	2013	320.54	261.01	211.69
Av_Gap							**10017**			
Av_cpu			248.65		–			191.86	159.26	126.10

TABLE 8.9 – Résultats de l'algorithme parallèle PCGBS et de l'algorithme séquentiel SCGBS. Le symbole "–" signifie que le temps de résolution de SGBS a dépassé 900 secondes.

De la table 8.9, on peut observer que :

1. SCGBS_4 (avec une largeur de faisceau $\delta = 4$) produit des solutions moyennes à cause de la limitation du temps de résolution (900 secondes).
2. SCGBS_2 résout toutes les instances dans un temps plus intéressant que SCGBS_4. Le temps de résolution moyen est égale à 248.65 secondes.
3. PCGBS résout plus efficacement que SCGBS_2 et SCGBS_4. Il améliore toutes les meilleures solutions de la littérature. Dans ce cas, la moyenne de la déviation est de 10017. Concernant le temps de résolution de PCGBS, nous avons analysé le temps nécessaire pour produire une solution, selon le nombre de processeurs utilisés par PGBS :
   (a) Le temps de résolution moyen de PGBS_8^6 est plus petit que 248.65 secondes (temps de résolution moyen de SCGBS_2) et des 900 secondes nécessaires à SCGBS_2. Par rapport aux temps obtenus par SGBS_2 ; PGBS_8^6 réduit le temps moyen de résolution de 35.9%. Le speedup $S(6)$ et l'efficacité $F(6)$ de PGBS_8^6 sont, respectivement, égale à 1.29 et 0.21.
   (b) PGBS_8^{10} réduit de 16.9% le temps de résolution de PGBS_8^6. Le speedup $S(10)$ et l'efficacité $F(10)$ sont égales, respectivement, à 1.56 et 0.15.
   (c) Le temps de résolution de PGBS_8^{20} est plus intéressant. Il réduit le temps de $65,76$ secondes (PGBS_8^6) à 126.10 secondes. On remarque que la diminution est plus importante par rapport à la meilleure version de l'algorithme séquentiel (Il là réduit de 51.71% en moyenne). Le speedup $S(20)$ et l'efficacité $F(20)$ de PGBS_8^{20} sont égales, respectivement à 1.97 et 0.09.

8.3 Étude comparative

L'objectif principal des tests présentés dans les sections 8.1.5 et 8.2.3 été de mesurer les performances par rapport aux méthodes séquentielles. En effet, l'algorithme $\text{PGBS}_{\delta=4}$ semble incapable de résoudre les instances traitées dans un temps raisonnable. La version parallèle de cet algorithme nous a permis d'atteindre des largeurs plus importantes ($\delta = 8$) et d'améliorer, ainsi les qualités des solutions produites. De la même manière, l'algorithme CGBS $_{delta=8}$ été incapable de résoudre ces instances (dans un temps inférieur à 900 secondes) et la version parallèle nous a permis d'atteindre une largeur égale à $\delta = 16$.

Nous présentons dans cette section une comparaison numérique des deux algorithmes. Nous avons traité les instances précédente avec les deux algorithmes dans les même conditions (Largeur du faisceau δ, nombre de processeur η, période de synchronisation T). L'objectif est de mesurer l'impact de la procédure de remplissage complémentaire (MFP) et le calcul de la borne supérieure sur les performances de PGBS.

Les table 8.10 et 8.11 présentent les résultats obtenus par PGBS et PCGBS dans le cas où la période de synchronisation est égale à 1 ($T = 1$). Pour cette même valeur, nous avons traité le premier groupe d'instances avec PGBS_8^{20}, PGBS_{16}^{20}, PCGBS_8^{20} et PCGBS_{16}^{20}. Nous avons ensuite lancé PGBS_4^{20}, PGBS_8^{20}, PCGBS_4^{20} et PCGBS_8^{20} sur le deuxième groupe d'instances. Pour chaque version, la table 8.10 rapporte, la valeur de la solution réalisable obtenue et le temps du traitement pour le premier groupe d'instances et la table 8.11 montre les mêmes résultats pour le deuxième groupe d'instances.

La table 8.10 qui présente les résultats du premier groupe d'instance monte que :

1. PGBS_8^{20} est la version la plus rapide avec un temps de résolution moyen de 1.25 secondes. Le nombre de solutions améliorées par cette version est de 27 sur les 44 instances traitées. PGBS_8^{20} génère une perte dans les qualités des solutions avec une déviation de $-1400,36$.
2. PGBS_{16}^{20} est la version la plus lente avec un temps moyen de $93,84$. Cette version arrive à améliorer toutes des instances traitées.
3. PCGBS_8^{20} est plus long que PGBS_8^{20} avec un temps de résolution moyen de 52.24 secondes. Le nombre de solutions réalisables améliorées est de 19 sur les 44 traitées et la perte moyenne dans la qualité de solution est de 1397 (très proche de la perte générée par PGBS_8^{20}).
4. PCGBS_8^{20} donne les même résultats que PGBS_{16}^{20} (en terme de qualité des solutions) en un temps moins important (61.58 secondes contre $93,84$ secondes).

A travers ces résultats, en remarque que le paramètre δ reste un facteur très important dans la résolution parallèle. L'augmentation de la largeur, augmente systématiquement le temps de calcul et montre la limite de la politique de diversification utilisées (synchronisation périodique).

Ces résultats montrent, néanmoins l'intérêt de l'utilisation d'une procédure de remplissage complémentaire pendant la résolution avec une largeur de faisceau importante (PCGBS_{16}^{20} a réduit de 34.37% de temps de résolution de PGBS_{16}^{20}), cependant la méthode de remplissage complémentaire alourdie la résolution des petites instances.

#Inst.	Best/Opt	$PGBS_8^{20}$			$PGBS_{16}^{20}$			$PCGBS_8^{20}$			$PCGBS_{16}^{20}$		
		Sol.	Gap.	cpu	Sol.	Gap.	cpu	Sol.	Gap.	cpu	Sol.	Gap.	cpu
nice25H*	860829	860829	○	○	860829	○	3.76	860829	○	1.01	860829	○	1.01
nice50H*	913738	913738	○	○	913738	○	8.89	913738	○	2.98	913738	○	3.21
nice100H	911203	911203	○	0.01	911203	○	13.98	911122	-81	6.98	911203	○	8.98
nice200H	940040	940040	○	0.01	940040	○	42.01	932686	-7354	20.01	940040	○	22.01
nice500H	953608	953598	-10	0.04	953608	○	176.98	952015	-1593	120.91	953608	○	141.98
path25H*	892765	892765	○	0.01	892765	○	3.98	892765	○	0.01	892765	○	0.01
path50H*	750215	750215	○	0.01	750215	○	3.98	750215	○	0.01	750215	○	0.01
path100H	888991	888991	○	0.01	888991	○	8.87	888578	-413	2.01	888991	○	2.01
path200H	890223	890223	○	0.21	890223	○	160.98	889174	-1049	80.98	890223	○	101.21
path500H	887667	887664	-3	2.01	887667	○	160.98	885112	-2555	89.05	887667	○	99.01
nice25PH*	441993	441993	○	○	441993	○	10.09	441993	○	0.11	441993	○	0.21
nice50PH*	453460	449292	-4168	○	453460	○	6.98	453460	○	2.01	453460	○	1.99
nice100PH	459781	456709	-3072	0.01	459781	○	190.98	459214	-567	108.94	459781	○	130.76
nice200PH	456888	456888	○	0.01	456888	○	101.87	455727	-1161	75.98	456888	○	89.87
nice500PH	459203	459001	-202	0.61	459203	○	180.98	455036	-4167	123.34	459203	○	141.76
nice1000PH	459098	459076	-22	0.31	459098	○	143.89	454089	-5009	99.06	459098	○	107.76
path25PH*	464721	464721	○	○	464721	○	3.87	464721	○	0.23	464721	○	0.07
path50PH*	391946	391626	-320	○	391946	○	2.98	391931	-15	1.05	391946	○	0.87
path100PH	457997	457997	○	○	457997	○	12.98	456300	-1697	4.98	457997	○	6.78
path200PH	459856	459856	○	0.12	459856	○	180.87	452402	-7454	111.98	459856	○	130.98
path500PH	440221	440167	-54	8.98	440221	○	230.87	434478	-5743	140.87	440221	○	169.05
path1000PH	441242	440001	-1241	17.98	441242	○	256.98	434966	-6276	109.81	441242	○	117.98
nice25V*	834157	834157	○	0.01	834157	○	4.44	834157	○	1.09	834157	○	0.98
nice50V*	868781	854425	-14356	0.23	868781	○	9.98	868781	○	2.99	868781	○	3.23
nice100V	911958	911958	○	0.89	911958	○	32.91	911958	○	12.98	911958	○	15.89
nice200V	936690	936690	○	0.97	936690	○	122.94	935290	-1400	80.06	936690	○	91.97
nice500V	965604	965604	○	1.98	965604	○	201.65	960767	-4837	143.98	965604	○	160.98
path25V*	699707	699707	○	○	699707	○	4.98	699707	○	1.05	699707	○	0.01
path50V*	924908	924908	○	0.03	924908	○	4.87	924908	○	1.43	924908	○	1.03
path100V*	754753	754753	○	0.76	754753	○	2.98	754678	-75	0.99	754753	○	0.76
path200V	836754	836754	○	0.87	836754	○	220.98	836022	-732	130.21	836754	○	140.87
path500V	808511	799903	-8608	0.72	808511	○	277.87	808376	-135	184.98	808511	○	199.72
nice25PV*	428662	428662	○	0.01	428662	○	4.87	428662	○	0.87	428662	○	0.01
nice50PV*	426269	425921	-348	0.01	426269	○	9.04	426269	○	2.07	426269	○	3.98
nice100PV	455406	455406	○	0.01	455406	○	29.98	451851	-3555	19.97	455406	○	23.18
nice200PV	454889	454889	○	0.42	454889	○	190.98	454754	-135	117.97	454889	○	130.42
nice500PV	457403	457381	-22	2.03	457403	○	210.98	457381	-22	11.96	457403	○	120.03
nice1000PV	459907	459142	-765	0.95	459907	○	230.87	459142	-765	142.97	459907	○	160.95
path25PV*	356088	356088	○	0.01	356088	○	4.89	356088	○	1.06	356088	○	0.01
path50PV*	487447	487447	○	0.01	487447	○	8.98	487447	○	1.99	487447	○	2.01
path100PV*	386033	386033	○	0.21	386033	○	14.98	386033	○	4.86	386033	○	6.21
path200PV	422706	422705	-1	0.22	422706	○	136.98	418024	-4682	70.95	422706	○	79.22
path500PV	381860	369381	-12479	0.64	381860	○	150.32	381860	○	80.87	381860	○	88.98
path1000PV	425768	409823	-15945	13.76	425768	○	344.87	425768	○	181.07	425768	○	201.76
$AV_{Gap.}$			-1400.36			○			-1397.09			○	
$AV_{cpu.}$				1.25			93.84			52.24			61.58

TABLE 8.10 – Comparaison entre PGBS et PCGBS - Premier groupe d'instances.

La table 8.11 montre les résultats obtenus pour le deuxième groupe d'instances que :

1. $PCGBS_4^{20}$ est la version la plus rapide avec un temps de résolution moyen de 121.21 secondes. Cette version n'améliore aucune solution et génère une perte dans les qualités des solutions avec une déviation de -10360.75.
2. $PGBS_8^{20}$ est la version la plus lente avec un temps moyen de 250.32 secondes. Cette version arrive à améliorer toutes des instances traitées.
3. $PCGBS_8^{20}$ réalise le meilleur compromis avec un temps de résolution moyen de 133.23 se-

#Inst.	Best/Opt	$PGBS_4^{20}$			$PGBS_8^{20}$			$PCGBS_4^{20}$			$PCGBS_8^{20}$		
		Sol.	Gap.	cpu	Sol.	Gap.	cpu	Sol.	Gap.	cpu	Sol.	Gap.	cpu
UL1H	889401	889386	-15	67.76	889401	○	110.32	887393	-2008	49.07	889401	○	60.71
UL2H	907972	907866	-106	129.76	907972	○	160.98	906712	-1260	110.98	907972	○	112.06
UL3H	1101201	1100992	-209	134.65	1101201	○	167.98	1021124	-80077	100.06	1101201	○	119.95
WL1H	694172	694099	-73	187.23	694172	○	201.87	683656	-10516	123.91	694172	○	139.23
WL2H	784189	784128	-61	198.6	784189	○	239.06	780018	-4171	133.87	784189	○	153.6
WL3H	900449	900367	-82	276.65	900449	○	388.93	899890	-559	234.96	900449	○	254.75
UL1V	900199	900099	-100	66.21	900199	○	144.98	891516	-8683	60.71	900199	○	65.81
UL2V	910021	910021	○	99.07	910021	○	190.08	908982	-1039	69.01	910021	○	77.07
UL3V	1027491	1027491	○	160.65	1027491	○	290.02	1022483	-5008	99.87	1027491	○	118.95
WL1V	699998	699994	-4	151.65	699998	○	299.54	697164	-2834	111.98	699998	○	121.95
WL2V	789997	789821	-176	182.04	789997	○	310.98	783836	-6161	160.96	789997	○	163.04
WL3V	892709	892650	-59	221.67	892709	○	499.06	890696	-2013	199.08	892709	○	211.69
$Av_G^{ap.}$			-73.75			○			-10360.75			○	
Av_c^{pu}				156.32			250.31			121.20			133.23

TABLE 8.11 – Comparaison entre PGBS et PCGBS - Premier groupe d'instances.

condes tous en améliorant toutes les solutions.

Ces résultats confirment l'utilité de la procédure de remplissage complémentaire pour la résolution de grandes instances. En effet, les grandes instances (ou largeur de faisceau important) gérèrent des chemins de recherche profonds, ce qui pénalise la qualité des bornes supérieures calculées pendant les développements. La procédure de remplissage permet d'approcher la solution réalisable finale et de rendre le filtrage plus efficace. Cependant, la procédure complémentaire, alourdie la recherche dans les arborescences moins profondes.

8.4 Conclusion

Dans ce chapitre, nous nous sommes intéressé à la résolution parallèle approchée du problème de découpe à deux dimensions. Les méthodes proposées s'appuient sur le paradigme maître/esclave et les spécificités de la recherche par faisceaux. L'objectif de ces deux algorithmes parallèles est d'améliorer les solutions produites pas la recherche par faisceaux en augmentant le nombre de nœuds considérés à chaque niveau de l'arbre de recherche. En effet, les méthodes séquentielles ont montré leurs limites en terme de temps de résolution et de consommation mémoire. La parallélisme a permis, non seulement de réduire les temps de résolution, mais aussi à d'explorer un plus grand espace de l'arbre de recherche et d'améliorer, ainsi les solutions réalisables finales.

Cinquième partie

CONCLUSION

Conclusion et perspectives

Dans cette thèse, nous avons étudié deux variantes du problème de découpe et de placement, le problème contraint et le problème double contraint à deux dimensions. La première partie concerne le problème de découpe contraint à deux dimensions, pour lequel, nous avons proposé plusieurs méthodes de résolution approchées. Nous nous sommes basés, dans la première méthode séquentielle, sur une recherche en faisceau et un nouveau mécanisme d'évaluation global pour approcher la solution optimale. Dans la deuxième méthode séquentielle, nous avons proposé un schéma de coopération entre la première méthode et une nouvelle méthode de remplissage complémentaire. Naturellement, la méthode de remplissage proposée peut être utilisée comme heuristique pour le problème considéré.

Afin de perfectionner les résultats et les performances des deux méthodes séquentielles et augmenter la largeur du faisceaux de recherche, nous avons choisi de paralléliser les deux méthodes précédentes. Ainsi, nous avons proposé deux méthodes de résolution approchée parallèles basées sur le paradigme maître/esclave. Ces deux méthodes parallèles ont permis d'améliorer sensiblement les qualités des solutions obtenues pour les instances traitées et de résoudre de plus grandes instances (non résolues par les méthodes séquentielles dans un temps raisonnable).

La deuxième partie concerne le problème de découpe double contraint à deux dimensions, pour lequel, nous avons proposé un algorithme exact. Cet algorithme est une procédure de séparation et évaluation basée sur une construction par pièce. Afin d'optimiser l'espace de recherche, nous avons introduit une nouvelle borne inférieure. La borne inférieure proposée, est une heuristique basée sur la résolution du problème de découpe à une seule contrainte et une méthode de restauration de la réalisabilité. Par ailleurs, nous avons proposé plusieurs bornes supérieures basées sur la résolution du problème de sac à dos unidimensionnel. Naturellement, l'algorithme exact proposé peut être facilement adapté à d'autres variantes du problème de découpe et de placement, notamment le problème contraint à deux dimensions traité dans la première partie.

Afin d'exploiter les résultats obtenus et expérimenter la recherche par faisceau sur d'autres variantes du problème de découpe et de placement, nous avons traité le problème de découpe à trois dimensions. Pour lequel, nous avons proposé une heuristique basée sur une décomposition et une simplification du problème initial en plusieurs sous problèmes de découpe à deux dimensions.

Parmi les perspectives à court terme, il nous semble intéressant d'étudier l'encadrement de variables pour le problème de découpe à deux dimensions et l'introduction du parallélisme dans la procédure de construction de l'algorithme exact.

Bibliographie

[1] A.A.Farley : The cutting stock problem in the industry. *European Journal of Operational Research*, Vol.44:pp.256 to 266, 1990.

[2] A.A.Farley : Selection of stockplate characteristics and cutting style for two dimensional cutting stock situation. *Operational Research*, Vol.44:pp.247 to 255, 1990.

[3] A.Lodi et M.Monaci : Integer linear programming models for 2−staged two−dimensional knapsack problems. *Mathematical Programming*, Vol.94:pp.257 to 278, 2003.

[4] A.RinnoyKan, J.DeWit et R.Wijmenga : Non−orthogonal two− dimensional cutting patterns. *Management Sci.*, Vol.33:pp.670 to 684, 1987.

[5] B.Mans et C.Roucairol : Performances of parallel branch and bound algorithms with best first search. *Discrete Applied Mathematics*, Vol.66:pp.57 to 76, 1996.

[6] C.Carnieri, G.A.Mendoza et L.G.Gavinho : Solution procedures for cutting lumber into furniture parts. *European Journal of Operational Research*, Vol.73:pp.295 to 501, 1994.

[7] C.G.Codd : Multiprogram sheduling. *Comm.of the ACM*, Vol.3(6):pp.347 to 350, 1960.

[8] D.Fayard et V.Zissimopoulos : An approximation algorithm for solving unconstrained two−dimensional knapsack problems. *European Journal of Operational Research*, Vol.84(3): pp.618 to 632, 1995.

[9] E.Bischoff et M.Marriott : A comparative evaluation of heuristic for container loading. *European Journal of Operational Research*, Vol.44:pp.267 to 276, 1990.

[10] F.J.Vasko : A computational improvement to wang's two−dimensional cutting stock algorithm. *Computers and Industrial Engineering*, Vol.16:pp.109 to 115, 1989.

[11] G.Autie, J.M.Garcia, A.Ferreire, J.L.Roch, G.Villard, J.Roman, C.Roucairol et B.Virot : *Parallélisme et applications irrégulières*. Hermes, 1995.

[12] G.B.Danzing : Inductive proof of the simlex method. *IBM Journal*, p. pp.505 to 506, 1960.

[13] G.B.Danzing : *Linear programming and extensions*. Princeton university press, 1963.

[14] G.Belov et G.Scheithauer : A branch and cut and price algorithm for one dimensional stock cutting and two dimensional two stage cutting. *European Journal of Operational Research*, Vol.171:pp.85 to 106, 2006.

[15] G.Kuntz et J.P.Uhry : *Optimisation de Pertes de Matière en Industrie Textile*. Thèse de doctorat, Laboratoire IMAG, Université de Grenoble, 1982.

[16] H.Dickhoff et U.Finke : *Cutting and Packing Problems in Production and Distribution : Typology and Bibliography*. Springler−Verlag, Heildelberg, 1992.

[17] H.Dyckhoff : A typology of cutting and packing problems. *European Journal of Operational Research*, Vol.44:pp.145 to 159, 1990.

[18] H.Kellerer, U.Pferschyand et D.Pisinger : *Knapsack problems.* Springer, 2003.

[19] J.Blazewicz, M.Drozdowski, B.Soniewicki et R.Walkowiak : Two−dimensional cutting problem. *In International Inst.Appl.Sys.Analysis*, 1991.

[20] J.C.Herz : A recursive computing procedure for two−dimensional stock cutting. *IBM Jour.Res.Develop.*, Vol.16:pp.462 to 469, 1972.

[21] J.Daniels et P.Ghandforoush : An improved algorithm for the non−guillotine constrained cutting−stock problem. *Jour.Opnl.Res.Society*, Vol.41(2):pp.141 to 150, 1990.

[22] J.E.Beasley : An algorithm for the two−dimensional assortement problem. *European Journal of Operational Research*, Vol.19:pp.253 to 261, 1985.

[23] J.E.Beasley : Algorithms for unconstrained two−dimensional guillotine cutting. *Journal Operational Research and Society*, Vol.36:pp.297 to 306, 1985.

[24] J.E.Beasley : An exact two−dimentional non−guillotine cutting tree search procedure. *Operational Research*, Vol.33(1):pp.49 to 64, 1985.

[25] J.E.Beasley : An algorithm for set covering problems. *European Journal of Operational Research*, Vol.31:pp.85 to 93, 1987.

[26] K.V.Viswanathan et A.Bagchi : Best−first search methods for constrained two− dimensional cutting stock problems. *Operational Research*, Vol.41(4):pp.768 to 776, 1993.

[27] L.V.Kantorovich : Mathematical methods of organizing and planing production. *Management Sci.*, Vol.6:pp.363 to 422, 1960.

[28] M.Adamowicz et A.Albano : Two−stage solution of the cutting stock problem. *Information Procc, North−Holland*, Vol.71:pp.1086 to 1091, 1972.

[29] M.Adamowicz et A.Albano : A solution of the rectangular cutting stock problem. *IEEE Trans.Syst.Man and Sybern*, Vol.6:pp.302 to 310, 1976.

[30] M.Biro et E.Boros : Network flows and non−guillotine cutting patterns. *European Journal of Operational Research*, Vol.16:pp.215 to 221, 1984.

[31] M.Cosnard et D.Trystram : *Algorithmes et architectures paralleles.* InterEditions, 1993.

[32] M.Hifi : An improvement of viswanathan and bagchis exact algorithm for cutting stock problems. *Computers and Operations Research*, Vol.24:pp.727 to 736, 1997.

[33] M.Hifi : *Rapport d'habilitation à diriger les recherches.* Thèse de doctorat, Université de Paris 1 Panthéon−Sorbonne, France, 1998.

[34] M.Hifi : Exact algorithms for large−scale unconstrained two and three staged cutting problem. *Computational Optimization and Applications*, Vol.18:pp.63 to 88, 2001.

[35] M.Hifi : Approximate algorithms for the container loading problem. *International Transactions in Operational Research*, Vol.9:pp.747 to 774, 2002.

[36] M.Hifi : Dynamic programming and hill−climbing techniques for constrained two−dimensional cutting stock problems. *Journal of combinatorial optimization*, Vol.8(1): pp.65 to 84, 2004.

[37] M.Hifi et C.Roucairol : Approximate and exact algorithms for constrained (un)weighted two−dimensional two−staged cutting stock problems. *Journal of Combinatorial Optimization*, Vol.5:pp.465 to 494, 2001.

[38] M.Hifi et R.MHallah : An exact algorithm for constrained two−dimensional two−staged cutting problems. *Operations Research*, Vol.53:pp.140 to 150, 2005.

[39] M.Hifi et R.MHallah : Strip generation algorithms for two−staged two−dimensional cutting stock problems. *European Journal of Operational Research*, Vol.172(2):pp.515 to 527, 2006.

[40] M.Hifi, R.MHallah et T.Saadi : Un algorithme par génération de couches pour le problème de découpe à deux niveaux. *In 7ème congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision −ROADEF−, Lille, France*, 2006.

[41] M.Hifi, R.MHallah et T.Saadi : Approximate and exact algorithms for the double−constrained two−dimensional guillotine cutting stock problem. *Computational Optimization and Applications*, Vol.ISSN 0926.6003 (Print) 1573 to 2894 (Online), 2007.

[42] M.Hifi, R.MHallah et T.Saadi : Algorithms for the constrained two−staged two−dimensional cutting problem. *Informs Journal On Computing*, Vol.20:pp.212 to 221, 2008.

[43] M.Hifi et T.Saadi : Using strip generation procedures for solving constrained two−staged cutting problems. *In The Fifth ALIO/EURO conference on combinatorial optimization, ENST, Paris, France*, 2005.

[44] M.Hifi et T.Saadi : A cooperative algorithm for constrained two−staged 2d cutting problems. *IEEE Service Systems and Service Management international Conference, Troyes, France*, Vol.2:pp.928 to 933, 2006.

[45] M.Hifi et T.Saadi : Using bounded knapsack problems for two−staged two−dimensional cutting stock problems. *In International Conference on Metaheuristics and Nature Inspired computing, META, Tunis, Tunisie*, 2006.

[46] M.Hifi et T.Saadi : Un algorithme coopératif pour les problèmes de découpe à deux dimensions à deux niveaux. *In Conférence scientifique conjointe en Recherche Opérationnelle et Aide à la Décision, FRANCORO V/ROADEF, Grenoble, France*, 2007.

[47] M.Hifi et T.Saadi : A cooperative algorithm for constrained two−staged two−dimensional cutting problems. *International Journal of Operational Research*, à paraitre, 2008.

[48] M.Hifi et T.Saadi : A parallel algorithm for constrained two−staged 2d cutting problems. *In International workshop on Operation Research, Madrid, Espagne*, 2008.

[49] M.Hifi et T.Saadi : A parallel algorithm for constrainted two−staged two−dimensional cutting problems. *Informs Journal On Computing*, Soumis, 2008.

[50] M.Hifi et T.Saadi : A parallel cooperative algorithm for constrainted two−staged two−dimensional cutting problems. *Operations Research*, Soumis, 2008.

[51] M.Hifi et T.Saadi : Un problème de placement en trois dimensions. *In 9ème congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision, Clermont−Ferrand*, 2008.

[52] M.Hifi et V.Zissimopoulos : An improved version for christofides and whitlock's algorithm for solving a two−dimensional cutting stock problem. *The Journal of the Operational Research Society*, Vol.48(3):pp.324 to 331, 1997.

[53] M.J.Flynn : Some computer organisation and their effectiveness. *IEEE Transaction on computer*, Vol.21:pp.948 to 960, 1979.

[54] M.J.Quinn et N.Deo : An upper bound for the speedup of parallel branch and bound algorithms. *In The 3rd Conf.on Found.of Software Technology and Theorical Computer Science*, 1985.

[55] M.R.Garey et R.L.Graham : Bounds on multiprocessing scheduling with resource constraints. *Siam, Jour.Comput.*, Vol.4(2):pp.187 to 200, 1975.

[56] N.Christofides : Optimal cutting of two−dimensional rectangular plates. *In CAD74*, 1974.

[57] N.Christofides et C.Whitlock : An algorithm for two−dimensional cutting problems. *Operational Research*, Vol.25:pp.31 to 44, 1977.

[58] N.Christofides et E.Hadjiconstantinou : An exact algorithm for orthogonal 2−d cutting problems using guillotine cuts. *European Journal of Operational Research*, Vol.83:pp.21 to 38, 1995.

[59] P.DeCani : A note on the two−dimensional rectangular cutting−stock problem. *European Journal of Operational Research*, Vol.29:pp.703 to 706, 1978.

[60] P.E.Sweeney et R.W.Haessler : One−dimensional cutting stock decisions for rolls with multiple quality grades. *European Journal of Operational Research*, Vol.44:pp.224 to 231, 1990.

[61] P.Gilmore et R.Gomory : A linear programming approach to the cutting stock problem. *Operational Research*, Vol.9(6):pp.849 to 859, 1961.

[62] P.Gilmore et R.Gomory : Multistage cutting problems of two and more dimensions. *Operational Research*, Vol.13:pp.94 to 119, 1965.

[63] P.Gilmore et R.Gomory : The theory and computation of knapsack functions. *Operational Research*, Vol.14:pp.1045 to 1074, 1966.

[64] P.S.Ow et T.E.Morton : Filtered beam search in scheduling. *Intern.J.Production Res.*, Vol.26:pp.297 to 307, 1988.

[65] P.Sweeney et E.Paternoster : problems : a categorized application−oriented research. *Journal of the Operational Research Society*, Vol.43:pp.691 to 706, 1992.

[66] P.Y.Wang : Two algorithms for constrained two−dimensional cutting stock problems. *Operational Research*, Vol.31(3):pp.573 to 586, 1983.

[67] R.Alvarez−Valdes, R.Martì, A.Parajón et J.M.Tamarit : Grasp and path relinking for the two−dimensional two−staged cutting stock problem. *INFORMS Journal on Computing*, Vol.19(2):pp.1 to 12, 2007.

[68] R.Corrêra et A.Ferreira : *A distributed implementation of asynchronous parallel branch and bound.In : Parallel Algorithms for Irregular Problems : State of the Art.* Kluwer Academic publishers, 1995.

[69] R.Ducan : A survey of parallel computer architectures. *IEEE Computer*, Vol.23(2):pp.5 to 16, 1990.

[70] R.G.Dyson et A.S.Gregory : The cutting stock problem in the flat glass industry. *Opnl.Res.Quart.*, Vol.25:pp.41 to 53, 1974.

[71] R.Morabito et M.Arenales : An and/or graph approach to the container loading problem. *International Transactions in Operational Research*, Vol.1:pp.59 to 73, 1994.

[72] R.Morabito et V.Garcia : The cutting stock problem in hardboard industry : a case study. *Computers and Operations Research*, Vol.25:pp.469 to 485, 1998.

[73] R.W.Haessler : Heuristic programming solution to a nonlinear cutting stock problem. *Mgmt.Sci.*, Vol.17:pp.793 to 802, 1971.

[74] R.W.Haessler : Controlling cutting pattern changes in one dimensional trim problems. *Operational Research*, Vol.23:pp.483 to 493, 1975.

[75] R.W.Haessler et P.E.Sweeney : Cutting stock problems and solution procedures. *European Journal of Operational Research*, Vol.54:pp.141 to 150, 1991.

[76] S.G.Hahn : On the optimal cutting of defective sheets. *Operational Research*, Vol.16:pp.1100 to 1114, 1968.

[77] S.P.Fekete et J.Schepers : A general framework for bounds for higher−dimensional orthogonal packing problems. *Mathematical Methods of Operations Research*, Vol.60:pp.311 to 329, 2004.

[78] T.H.Lai et A.Sprague : Performance in parllel branch and bound algorithms. *IEEE Trans.Comput.*, Vol.34:pp.962 to 964, 1985.

[79] T.H.Lai et S.Sahni : Anomalies in paralllel branch and bound algorithms. *Communication ACM*, Vol.27:pp.594 to 602, 1984.

[80] V.D.Cung, M.Hifi et B. Cun : Constrained two dimensional cutting stock : A best first branch and bound exact algorithm. *Technical Report 97/020,Laboratoire PRiSM − CNRS URA 1525. Université de Versailles, Saint Quentin en Yvelines. 78035 Versailles Cedex, FRANCE*, 1997.

[81] V.Zissimopoulos : Heuristic methods for solving (un)constrained two−dimensional cutting stock problems. *Math.Opns.Res.*, Vol.49:pp.345 to 357, 1985.

Liste des tableaux

Table des figures