N° d'ordre : 09/2004-E./MT

République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur
Et de la Recherche Scientifique

UNIVERSITE des SCIENCES et de la TECHNOLOGIE HOUARI BOUMEDIENNE

FACULTE DE MATHEMATIQUES

THESE PRESENTEE POUR L'OBTENTION DU DIPLOME DE DOCTORAT D'ETAT EN MATHEMATIQUES

Spécialité : Recherche Opérationnelle

Par : OUAFI RACHID

Sujet :

APPROCHE ALGORITHMIQUE POUR UNE CLASSE DE PROBLEMES DE DECOUPE

Soutenue le 13/12/2004, devant le jury composé de :

AIDER Meziane	**M.C**	**USTHB**	**Président**
KHELLADI Abdelkader	**Professeur**	**USTHB**	**Directeur de Thèse**
HIFI Mhand	**Professeur**	**U.Amiens**	**Co-Directeur de**
ROUCAIROL Catherine	**Professeur**	**U.Versailles**	**Examinatrice**
ABBAS Moncef	**Professeur**	**USTHB**	**Examinateur**
AIDENE Mhand	**Professeur**	**U. Tizi Ouzou**	**Examinateur**
BERRACHEDI Abdelhafid	**Professeur**	**USTHB**	**Examinateur**

Table des matières

Chapitre 3 Un nouvel algorithme pour la résolution du problème de découpe (non) pondéré à deux dimensions.

Chapitre 4 Approche séquentielle exacte pour la résolution du problème de découpe généralisée sans contraintes

Chapitre 5 Un algorithme optimal pour le problème d'assemblage

Chapitre 6 Problème de découpe de pièces trapézoïdales

INTRODUCTION

L'optimisation combinatoire est une branche particulièrement active des mathématiques appliquées. L'analyse de nombreux problèmes de la recherche opérationnelle en univers déterministe conduit assez souvent à des modèles d'optimisation combinatoire. C'est de la réunion d'une large panoplie de techniques adaptées à la nature combinatoire des problèmes, qu'est née l'optimisation combinatoire. Son usage s'est répandu rapidement pour la résolution de problèmes complexes posés en économie appliquée, avec un recours de plus en plus étendu aux moyens de calcul modernes. Avec l'avènement de l'outil informatique, des perspectives de recherche nouvelles s'offrent pour le traitement de problèmes de grandes dimensions, suscitant un intérêt croissant de la part des entreprises industrielles. Les domaines d'application se multiplient et les problèmes dont la résolution devient possible changent à la fois de nature et de taille, nécessitant l'élaboration de méthodes algorithmiques de plus en plus raffinées qui font appel à de nouvelles approches, ayant pour objectif le perfectionnement de techniques de calcul.

Le thème abordé dans nos travaux de recherche se situe au carrefour de la recherche opérationnelle et de l'informatique en réponse à des besoins dictés par des applications concrètes, en relation avec les systèmes de production propres à certaines industries. Sur ce vaste terrain d'application, notre aspiration s'est focalisée particulièrement sur les problèmes de découpe. Ces problèmes interviennent dans de nombreux processus de fabrication de diverses industries : textile, verre, bois, papier métal, etc… Au-delà de l'aspect réducteur de sa modélisation, le problème de découpe est un problème NP-complet [27] et possède de nombreuses variantes. Cette diversité est reliée au support ou matériel considéré, a la forme des objets (qu'on appellera pièces) à placer et aux contraintes imposées sur la façon de découper le(s) support(s). Suivant la contrainte de découpe imposée et appliquée au support, les chutes dans le support seront plus ou moins importantes.

Un plan de découpe entraîne un coût (perte), et un bon choix provoquant peu de perte de matière, peut se traduire par des gains substantiels. Ainsi la recherche de bonnes solutions au problème de découpe et à fortiori d'une solution optimale qui minimise les pertes, est d'un grand intérêt.

On dit que le problème de découpe est à deux dimensions, si la forme du (ou des) support(s) et de chaque objet est à deux dimensions (exemple : rectangle, cercle etc.).
Dans ce large registre abordé par de nombreux auteurs [8,9,10,11], notre contribution s'est focalisée sur la conception d'approches algorithmiques pour une classe de problèmes de découpe à deux dimensions dont les supports sont rectangulaires.

Le schéma général que nous avons utilisé dans l'élaboration des approches de résolution, consiste en une structure d'arborescence représentant l'espace de solutions du problème considéré. Le parcours de l'arborescence nécessite par conséquent, la spécification du sommet initial (racine), des sommets solutions (pendants), ainsi que les règles de mouvements (cheminements) permis, privilégiant certaines directions par rapport à d'autres dans la visite des sommets autorisés de l'arborescence. Pour le problème de découpe les règles qui définissent les cheminements permis tiennent compte, des ensembles de discretisation des points de découpe, des procédés de détection des formes dupliquées des pièces produites, ainsi que des dissections inutiles [7, 22].
Une recherche de solution ''efficace'' via l'exploration d'une arborescence, impose l'utilisation de méthodes d'énumération implicites afin de limiter au maximum l'explosion combinatoire de celle ci. Ces méthodes s'appuient sur des procédures à qui appartient la décision du choix des règles de mouvement, c 'est à dire à quel moment et quelle opération doit être appliquée sur les sommets disponibles de l'arborescence, des résultats de ces règles intelligentes découle irrémédiablement les performances d'optimalité des algorithmes développés.
C'est ce raisonnement qui a constitué l'idée principale sur laquelle nous avons conçu nos approches algorithmiques, avec pour objectif premier de réduire l'effort de recherche sans risque de perdre la solution optimale (pour les méthodes exactes), par l'introduction de techniques de la recherche opérationnelle tel que la programmation dynamique, auquel nous faisons appel dans les opérations d'évaluation des bornes (inférieures et supérieures) initiales et intermédiaires des sommets de l'arborescence, ainsi que des méthodes d'intelligence artificielle en vue de détecter et d'éliminer les découpes non nécessaires ou infructueuses.
Cette thèse comporte cinq chapitres. Dans le premier chapitre, nous rappelons les principales définitions et résultats qui constituent les notions de base concernant les approches de résolution des problèmes d'optimisation combinatoire.

Le deuxième chapitre est entièrement consacré à la description et à la classification des problèmes de découpe. Nous présentons les définitions des concepts de base et les propriétés sur lesquelles se basent les modélisations des problèmes de découpe. Comme nous l'avons souligné, les problèmes de découpe sont de nature diverse. Cette diversité est reliée au support considérée, au type d'objets (pièces) à découper (placer) et aux contraintes sur la façon de découper le(s) support(s). Pour ces raisons, il nous a semblé intéressent de donner une classification des problèmes de découpe afin de pouvoir les identifier plus facilement. Nous proposons une classification sous forme de représentation symbolique. Cette classification regroupe toutes les informations sur le problème considéré.

Dans le chapitre 3, nous présentons une nouvelle approche [41] pour résoudre le problème de découpe sans contraintes à deux dimensions, pour ses deux versions pondéré et non pondérée. L'approche proposée s'appuie sur la combinaison entre une recherche globale et l'utilisation de raffinements sur les bornes de l'arborescence crée. Nous introduisons à cet effet, une procédure de calcul des bornes inférieures basée sur des méthodes de programmation dynamique, ainsi qu'une borne supérieure obtenue par la résolution de problèmes de sac à dos unidimensionnels relaxés. L'algorithme s'applique au problème de découpe dans le cas où le profit sur les pièces est quelconque, les pièces sont pondérées par un poids qui n'est pas nécessairement égal à leur surface. Nous proposons ensuite, une heuristique dans le but de renforcer les performances de l'approche exacte notamment dans le traitement des instances de grande taille.

Dans le quatrième chapitre, nous proposons une méthode exacte et une heuristique [41] pour la résolution du problème de découpe sans contraintes généralisé. Le principe de la méthode exacte, repose sur l'utilisation de l'approche de Gilmore et Gomory [31] et de l'introduction d'une nouvelle stratégie qui permet de réduire l'espace de recherche. Au début, nous donnons la définition de support rectangulaire *non-dominant*, en utilisant cette définition, nous démontrons que la présence de tel support peut être omise sans affecter la solution optimale du problème.

Par la suite, nous démontrons que si toutes les longueurs des supports respectaient un ordre croissant, alors nécessairement toutes les hauteurs respecteraient l'ordre inverse. Finalement, en utilisant l'ordre sur les longueurs, nous démontrons la convergence de l'algorithme.

Le chapitre 5 porte principalement sur l'étude du problème d'assemblage. Ce problème fait partie de la famille des problèmes de découpe, Il consiste à déterminer le meilleur placement d'un nombre de pièces rectangulaires, dans une surface rectangulaire minimale, tel qu'à chaque pièce est associé un nombre qui désigne le nombre d'apparitions de la pièce dans le regroupement final. Nous proposons un algorithme constructif exact [42] qui repose sur une méthode d'énumération ascendante. La méthode consiste dans le développement d'une arborescence, où chaque sommet représente une solution réalisable du problème. L'évaluation d'un sommet est composée d'une évaluation sur la surface d'assemblage et d'une autre portant sur les utilités des pièces assemblées. Des raffinements sur le calcul des bornes et une stratégie de stérilisation des sommets sont utilisés en vue d'accélérer le parcours de l'arborescence de recherche. Les résultats numériques sur des expériences réalisées indiquent que l'algorithme exact est capable de résoudre des instances de petite et moyenne taille de manière efficace, en un temps d'exécution raisonnable. Les modifications [43] apportées à l'algorithme par l'heuristique de recherche allège considérablement l'arborescence de recherche. Les ajustements appropriés sur les paramètres de variation de l'heuristique, augmente de manière significative l'efficacité de l'algorithme pour des solutions de qualité comparées aux solutions optimales.

Le chapitre 6 est consacré à l'étude du problème de découpe de pièces trapézoïdales introduit initialement par A. Khelladi [45], c'est une variante du problème de découpe non orthogonale. Le problème consiste à déterminer le meilleur modèle de découpe du maximum de pièces de formes trapézoïdales sur un support rectangulaire fixé avec le minimum de chute. A signaler, pour ce problème ; il n'existe aucune approche de résolution. Nous proposons une approche de résolution basée sur une procédure constructive permettant d'obtenir le meilleur plan de découpe à partir de combinaison des bandes homogènes horizontales et verticales. Nous démontrons que notre approche admet un rapport d'approximation constant.

Finalement, nous présentons une thématique principale sur des recherches en cours et futures axées essentiellement, sur la conception de nouveaux algorithmes séquentiels et parallèles portant sur des améliorations des algorithmes existants ou ayant trait à d'autres problèmes non traités.

Chapitre 1

Problèmes d'optimisation combinatoire : Approches de résolution

1.1. Introduction

La classe des problèmes qui définit le contexte de l'optimisation combinatoire apparaît à première vue d'une déconcertante trivialité. Il s'agit de déterminer un meilleur (plus petit ou plus grand) élément d'un ensemble fini.

La plupart des modèles de recherche opérationnelle se ramènent à la recherche d'un optimum d'un ensemble fini de valeurs. Traditionnellement la combinatoire, qui s'était surtout développée en conjonction avec le calcul des probabilités, se fixait comme objectif essentiel de dénombrer des combinaisons d'un certain type d'objets. La théorie des graphes proposait un outil puissant d'investigation des structures combinatoires.

Le mot combinatoire évoque, outre le caractère fini d'un ensemble d'éléments ; de cardinalité assez élevée. A ce contexte s'ajoute le cas de problèmes de taille relativement modestes pouvant conduire à un nombre de possibilités de solutions astronomique. Une distinction doit être considérée entre, les problèmes d'optimisation combinatoire et les problèmes combinatoires, ces derniers pour lesquels la recherche d'une solution faisable étant l'objectif principal. Cette distinction peut apparaître comme étant de moindre importance du moment que, la recherche d'une solution faisable peut toujours être interprétée comme un objectif de «minimisation de l'infaisabilité ». Cependant, il ne faut pas perdre de vue que le rapport du degré de difficulté et de complexité des problèmes dépendent précisément de l'objectif à atteindre. Certains problèmes, comme le problème du voyageur de commerce ou le problème de sac à dos, sont considérés comme des problèmes d'optimisation combinatoire, alors que pour d'autres, comme les problèmes de l'emploi du temps, seule une solution faisable est requise.

Le contexte suivant est assez typique aux problèmes d'optimisation combinatoire : il s'agit de déterminer une forme de combinaison optimale entre des éléments distincts d'un ou de plusieurs ensembles.

1.2. Problèmes d'optimisation combinatoire

Les problèmes d'optimisation combinatoire peuvent être répertoriés comme suit :

- *Problèmes séquentiels :* il s'agit de déterminer un sous ensemble d'éléments selon un *ordre faisable ou optimal*, le problème le plus connu de cette classe est le problème du voyageur de commerce. Le problème de plus court chemin fait partie aussi de cette catégorie.
- *Problème de sélection :* Ils consistent à *choisir* un sous ensemble d'éléments satisfaisant à certains critères. Les problèmes de sac à dos (knapsack) et le problème de couverture d'ensembles sont les problèmes standards connus de ce groupe.
- *Problèmes d'affectation :* Il est demandé d'affecter un nombre d'éléments d'un sous ensemble à un autre. Les problèmes types de ce groupe sont les problèmes d'affectation linéaire et les problèmes d'affectation quadratique.

Nous retrouvons aussi un grand nombre de problèmes réels qui sont des combinaisons de problèmes de ces groupes, tel que le problème de tournée (combinaison séquence - affectation) ou le problème de l'emploi du temps (combinaison affectation – séquence - sélection).

Définition 1.2.1.

Un problème d'optimisation combinatoire est défini à partir d'un ensemble fini S et d'une application $f : S \to R$ il s'agit de déterminer

$$s^* \in S \text{ tel que : } f(s^*) = \min_{s \in S} f(s)$$

Le problème qui consiste à chercher un élément maximum au lieu du minimum est de même nature, nous avons : $\max_{s \in S} f(s) = -\min_{s \in S}(-f(s))$

En général, f prend ses valeurs dans un ensemble totalement ordonné. Le plus souvent dans l'ensembles des entiers naturels.

Définition 1.2.2

S'il existe un ensemble E (fini) et une application :

$$c : E \to R$$

Tels que :

$$S \subset P(E) \text{ et } f(s) = \sum_{e \in S} c(e)$$

Alors, le problème d'optimisation combinatoire associé à f est dit à *fonction objectif séparée.*

1.2.3. Exemples de quelques problèmes d'optimisation combinatoire

1. Problème du voyageur de commerce

Un voyageur de commerce, ayant *n* villes à visiter, souhaite établir une tournée qui lui permet de passer une fois et une seule dans chaque ville, pour finalement retourner à son point de départ, ceci en minimisant le chemin parcouru.

Soit $R=(X,U,d)$ un réseau, où (X,U) est un graphe orienté dont les sommets sont les villes, les arcs désignent les chemins reliant les villes et $d:U\to R$ une application qui à chaque arc associe son coût de parcours.

Soit $(C_{i,j})_{i=1,\cdots,n}^{j=1,\cdots,n}$ la matrice $n\times n$ définie par :

$$C_{i,j}=\begin{cases} d_{ij} & \text{si } (i,j)\in U \\ \infty & \text{sinon} \end{cases}$$

Le problème du voyageur de commerce consiste à trouver une permutation sans cycles *p* sur les *n* villes telle que :

$$\sum_{i=1}^{n} C_{i,p(i)} \text{ soit minimum.}$$

2. Problème de sac à dos

On dispose de *n* objets ayant chacun un poids p_j et une utilité de valeur c_j pour. Il est demandé de faire une sélection sur l'ensemble des objets, dont le poids total soit inférieur ou égal à un nombre donné *b* et dont la somme des utilités des objets sélectionnés soit maximum.

Le problème de sac à dos consiste à déterminer $J\subset\{1,2,\cdots,n\}$ tel que :

$$\sum_{j\in J} c_j \quad \text{soit maximum,}$$

$$\text{Sous la contrainte } \sum_{j\in J} p_j \le b \quad .$$

1.3. Programmation linéaire en nombres entiers

Définition 1.3.1

Soit *A* une matrice $m\times n$, *b* un *m*-vecteur colonne et *c* un *n*-vecteur ligne. Sans perte de généralités nous supposons que *A, b et c*, sont à composantes entières.

Le problème :

$$(P)\ \begin{cases} max\, z = c.x \\ Ax \le b \\ x_j \in N, \quad j = 1,\cdots,n \end{cases}$$

est appelé programme linéaire en nombres entiers.

Remarques

- Si $x_j \in \{0,1\}$, le problème *(P)* est appelé programme linéaire en 0-1 ou à variables bivalentes.
- Pour les programmes linéaires en nombres entiers, on suppose que :

$$D = \{x / Ax \le b, x \ge 0\}$$

est borné, ceci implique que :

$$S = \{x / Ax \le b, x_j \in N, \quad j = 1,\cdots,n\}$$

est fini, ainsi *(P)* est un problème d'optimisation combinatoire.

- Le programme linéaire

$$(P')\ \begin{cases} max\, z = c.x \\ Ax \le b \\ x \ge 0 \end{cases}$$

obtenu à partir de *(P)* en relâchant les contraintes d'intégrité sur les variables est dit *programme linéaire relaxé* de *(P)*. Toute solution réalisable de *(P)* est une solution réalisable de *(P')* et si, x^* est une solution optimale de *(P')*, alors x^* est une solution optimale de *(P)* si elle est entière, sinon elle est faisable pour *(P)*.

La programmation linéaire en nombres entiers représente un outil très général de formulation des problèmes d'optimisation combinatoire, ceci permet d'utiliser la solution du programme linéaire relaxé pour cerner la solution entière.

1.4. Complexité des algorithmes

C'est avec l'apparition de l'informatique en tant que science (1936) et les travaux de Alan Mathison Turing (1912-1954), que le concept de complexité a commencé à émerger scientifiquement. La complexité à la «turing» dite aussi, complexité algorithmique est donnée pour un programme, par le temps de calcul nécessaire à celui-ci pour résoudre un représentant de la classe de problème pour lequel il a été conçu, dans le pire des cas. Ce qui est important, c'est de savoir alors exprimer le temps de calcul d'un programme en fonction de la taille du problème à résoudre, lorsqu'on fait tendre cette taille vers l'infini. Ainsi, on a des algorithmes dont le temps de calcul, s'exprime en fonction polynômiale ou en logarithme, d'autres en fonction exponentielle.

La théorie de la complexité des algorithmes[48] donne un sens précis aux termes d'algorithme efficace et de problème ``difficile``. Elle a permis d'asseoir l'optimisation combinatoire sur une base théorique solide. Pour décrire l'apport de cette importante transition innovatrice sur le plan conceptuel, nous reprenons les propos de Sakarovitch [57] : « Avant que ce sens ne fût donné, la formulation des problèmes d'optimisation combinatoire en termes de programmation linéaire en nombres entiers, voilait ce qui pouvait apparaître comme une nudité conceptuelle indécente ».

Généralement, les algorithmes que l'on met en œuvre pour résoudre un problème d'optimisation combinatoire dépendent de la complexité de ce dernier. La théorie de la NP-complétude (Gary et Johnson [29]) fournit de précieux renseignements sur le genre de méthodes à adopter en fonction de la difficulté intrinsèque des problèmes.

A côté des problèmes polynomiaux, considérés comme *faciles* et pour lesquels il existe (par définition) des algorithmes polynomiaux (par rapport à la taille des instances à traiter) permettant de les résoudre exactement, on trouve des problèmes NP- complets qu'on ne sait pas résoudre polynômialement et qui, de ce fait, sont considérés comme *difficiles.*

Définition 1.4.1.

Un algorithme de résolution d'un problème donné *(P)* est une procédure, décomposable en opérations élémentaires, transformant une chaîne de caractères représentant les données de toute instance du problème *(P)*, en une chaîne de caractères représentant les résultats de *(P).*

Le nombre d'opérations élémentaires effectuées par un algorithme est une fonction de la taille des données à traiter. La complexité d'un algorithme est donc directement relié à cette fonction : plus elle croit rapidement avec le nombre de données à traiter ; plus la complexité de l'algorithme est grande. La complexité de l'algorithme est donc, par définition, le terme

prépondérant dans la fonction donnant le nombre d'opérations élémentaires effectuées par un algorithme en fonction des données à traiter.

Quelques exemples d'évaluation de la complexité

1. *Evaluation d'un polynôme* :

Soit

$$P(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_n x^n$$

L'évaluation directe de *P(x)* conduit à une complexité

$$C_{P(x)} = O(n+n(n+1)/2) = O(n^2)$$

Un algorithme plus performant qui utilise une factorisation du polynôme sous la forme :

$$P(x) = a_0 + x(a_1 + x(a_2 + x(\cdots + a_n)\cdots)$$

Cette factorisation évite tout calcul de puissance, la complexité qui en découle est de l'ordre de *O(n).*

2. *Calcul matriciel* :

Soient *A* et *B* deux matrices de $n \times n$ éléments. Les principales opérations sur ces matrices ont les complexités suivantes :

- Lecture : $O(n^2)$
- Calcul de la trace : $O(n)$
- Multiplication : $O(n^3)$
- Déterminant (par la méthode de Cramer) : $O(n \times n!)$

Si on suppose qu'on utilise un ordinateur qui effectue une opération élémentaire en 10^{-9} secondes. On obtient, alors les temps de calcul suivants pour plusieurs «petites» valeurs de *n.*

n	5	10	15	20	50
Temps de calcul	$6.\ 10^{-7}$ secondes	0.04 secondes	5.5 heures	1543 années	$4.8\ .\ 10^{46}$ millénaires

D'où la nécessité de faire un calcul de complexité, avant de mettre un algorithme en exécution (à moins de travailler exclusivement pour les générations futures).

L'idée de la caractérisation des algorithmes efficaces a été introduite pour la première fois par J. Edmonds[22].

Définition 1.4.2.

Un algorithme est dit polynômial si le nombre d'opérations élémentaires pour résoudre un exemple de taille *n* est borné par un polynôme de taille *n*.

1.4.1 Problème de la classe NP

La difficulté d'un problème peut être mesurée par le temps d'exécution d'un algorithme qui le résout, ou encore par la quantité de mémoire requise lors de la mise en œuvre de cet algorithme. La notion de machine de Turing permet de définir une telle mesure absolue de la difficulté d'un problème; elle est appelée la complexité (algorithmique) du problème. La notion de complexité permet de définir deux grandes classes de problèmes:

Définitions 1.4.3.

- Un problème est dit de *classe P* s'il peut être résolu par un algorithme polynômial.
- Un problème de reconnaissance est un problème dont les résultats sont les deux valeurs "vraie" ou "faux.". Considérons le problème d'optimisation combinatoire :

$$\text{Trouver } s^* \in S \text{ tel que : } \quad f(s^*) = \min_{s \in S} f(s) \qquad (1)$$

Etant donné un nombre *r*, on définit le problème de reconnaissance associé au problème (1) par :

$$\text{Existe-t-il } \tilde{s} \in S \text{ tel que } f(\tilde{s}) \leq r$$

- Les algorithmes non déterministes sont une construction abstraite, ils sont basés sur des instructions de choix non spécifiés à priori. Les algorithmes non déterministes ne peuvent être mis en œuvre que sur une machine de Turing non déterministe. Pour un problème de reconnaissance, un algorithme non déterministe effectue des choix successifs de telle sorte qu'on arrive à la réponse *vraie* en un minimum de temps. Pour l'exemple du tri de *n* nombres, un algorithme non déterministe effectue le tri en *O(n)* opérations alors que tout algorithme déterministe pour un tel problème est au moins en *O(nlg(n)).*

- Un problème appartient à la classe NP s'il peut être résolu en temps polynômial par un algorithme non déterministe polynômial .
- Un problème de la classe NP est dit NP-complet, si tout problème dans la classe NP peut se réduire polynômialement à ce problème.

Théorème 1.4.4. [48]

Si le problème de reconnaissance associé à un problème d'optimisation combinatoire donné est difficile, le problème d'optimisation combinatoire est lui-même difficile.

Conséquence:

La propriété de la "NP-complétude" est très forte, puisqu'elle signifie, que s'il existe un algorithme polynômial pour un problème NP-complet, alors il existe un algorithme polynômial pour tous les problèmes de la classe NP [48].

Plusieurs choix sont possibles pour traiter un problème NP- complet. Nous retiendrons les choix suivants (parmi ceux que recensent Barthélemy, Cohen et Lobstein dans [6]) :

- Appliquer des algorithmes exacts dont la complexité peut croître de manière exponentielle avec la taille des instances à traiter ;
- Concevoir des heuristiques, c'est à dire des algorithmes qui fournissent une solution approchée, souvent assez bonne mais pas nécessairement optimale, en un temps de calcul raisonnable ;
- Concevoir des algorithmes polynomiaux avec garantie de performance, appelés aussi algorithmes d'approximation ; ces méthodes ne conduisent pas nécessairement à une solution optimale (ou très proche de l'optimum), mais garantissent que l'erreur (absolue ou relative selon les cas) commise en adoptant, la solution approchée calculée au lieu de la solution optimale reste en deçà d'une certaine valeur ;
- Utiliser dans les approches précédentes des machines parallèles pour augmenter d'une part la taille des instances et d'autre part diminuer le temps de calcul.

1.5. Schéma de résolution des problèmes d'optimisation combinatoire

Une approche de résolution pour un problème d'optimisation combinatoire en un temps raisonnable, écarte toute procédure basée sur la revue exhaustive de toutes les configurations possibles. En effet, la recherche de la solution, par énumération de l'ensemble des cas, même si elle est mathématiquement possible, implique un volume de calcul extraordinaire très coûteux. Il est impérativement nécessaire en optimisation combinatoire de distinguer les procédés de recherche de solutions selon leurs performances.

Comme les problèmes d'optimisation combinatoire ont un certain nombre de propriétés en commun il n'est pas surprenant qu'une approche générale de résolution soit suggérée, sous forme d'un guide ou de manuel pouvant s'adapter aux différentes structures des problèmes proposés. Elle consiste en les étapes suivantes :

- Position du problème.
- Conception d'un modèle de programmation en nombres entiers.
- Estimation du nombre de solutions.
- Recherche de méthodes d'énumération explicites adéquates.
- Application des méthodes arborescentes (énumération implicite).
- Développement et application de méthodes heuristiques.

1.5.1. Méthodes d'exploration arborescente

Une méthode d'exploration arborescente, construit itérativement une structure d'arbre qui consiste en des sommets reliés par des arcs orientés. Chaque sommet de l'arbre est identifié à une solution partielle du problème considéré. L'arborescence évolue au fur et à mesure que des sommets sont sélectionnés, des stratégies de branchement sont effectuées induisant de nouvelles directions, créant ainsi de nouveaux sommets. Une arborescence particulière est caractérisée par les spécificités concernant :

- Le choix des sous-ensembles à traiter.
- Les Procédés d'évaluation.
  a) Evaluation et mise à jour,
  b) Tests,
  c) Stérilisation.
- Les Stratégies de branchement.

Les stratégies de branchement et les procédés d'évaluation sont étroitement rattachés aux spécificités du problème considéré. Un sommet stérilisé est un sommet qui ne mène ni à une solution faisable, ni à la solution optimale. Les procédés d'évaluation permettent d'accélérer le processus de stérilisation des sommets, et de réduire ainsi leur énumération.

Le développement de l'algorithmique et de l'intelligence artificielle ont donné un très grand élan à l'essor des méthodes d'exploration arborescentes, cet apport a permis de marquer clairement, la différence fondamentale entre les méthodes d'exploration arborescentes et les procédures de séparation et d'évaluation progressive (SEP). En effet L'exploration des cheminements possibles par les méthodes arborescentes n'est pas guidée uniquement par la valeur de la fonction d'évaluation, comme c'est le cas pour les méthodes SEP. Plus précisément une méthode d'exploration arborescente est caractérisée par :

- Des états d'informations qui correspondent aux sommets de l'arborescence.
- Un jeu de règles spécifiant les moyens de succession de l'information d'un état donné à un autre.
- Une stratégie de contrôle global qui gère le déroulement de l'arborescence.

C'est la stratégie de contrôle qui détermine ; quelle règle doit être appliquée et à quel moment, ce qui dote ainsi, les arborescences de *systèmes adaptatifs.* Ce sont des mécanismes qui apprennent et sont doués d'une certaine intelligence.

Le fait de réserver le terme d'énumération implicite aux seules méthodes d'exploration arborescente, est dû exclusivement au codage spécial des sommets de l'arborescence, évitant ainsi, de garder en mémoire les informations cumulées sur l'ensemble des sommets de l'arborescence. La connaissance du sous ensemble en cours de traitement et d'un certain nombre d'informations sur les traitements des sommets antérieurs, résume complètement l'évolution de la procédure, à l'instant correspondant au sommet en cours de traitement.

1.5.2. Méthodes heuristiques

Il existe des problèmes que les méthodes d'énumération implicites ne sont pas capables de résoudre en un temps de calcul raisonnable. Ceci est dû en grande partie, soit à la structure générale soit à la spécificité du problème considéré. Le problème de l'affectation quadratique pour *n=10* au maximum et le problème de sac à dos pour n=50 et avec c_j / p_j très petit, en sont des exemples.

Dans ce cas, les méthodes dites heuristiques sont le seul moyen pour examiner ce type de problèmes. Les méthodes heuristiques ne sont pas astreintes à des critères mathématiques précis, tel que les critères d'optimalité. D'autre part, il n'existe pas de restrictions particulières quant aux structures des approches heuristiques. Les méthodes heuristiques sont considérées comme des procédures d'énumération implicite incomplète. Une heuristique abandonne la recherche sur un nombre de branches de l'arborescence sans être sûre qu'elles ne contiennent pas la solution optimale, mais conduit toujours à une solution réalisable (approchée), qui n'est pas nécessairement optimale.

Une heuristique est une procédure informelle, donnant toujours une solution réalisable, mais pas nécessairement optimale en un temps polynômial. Il n'y a aucune garantie pour qu'une heuristique adaptée à un problème NP-complet puisse être efficace pour un autre problème appartenant à la même classe.

Parmi les méthodes heuristiques, nous citerons les méthodes gloutonnes, et les algorithmes de recherches locales [13]. De manière générale, une ``bonne`` heuristique doit avoir les propriétés suivantes :

a. Fournir des solutions en un temps de calcul *raisonnable*

b. Les solutions fournies sont *proches* de la solution optimale, en *moyenne.*

c. La *probabilité* d'avoir des solutions individuelles assez éloignées de l'optimum est très *petite.*

Les conditions b) et c) ne sont pas identiques, puisque de bonnes solutions en moyenne n'excluent en aucun cas la possibilité d'obtenir de mauvaises solutions pour certaines exemples d'instances du problème considéré.

CHAPITRE 2

Notions générales sur les problèmes de découpe

2.1. Introduction

Le problème de découpe a été posé par Kantorovich [47] qui s'intéressait à une modélisation cohérente des problèmes rencontrés dans l'industrie. Il fut repris par Gilmore et Gomory [31] pour la résolution de quelques variantes du problème. Ces mêmes auteurs ([30]) l'ont ensuite généralisé aux problèmes de découpe à deux et à plusieurs dimensions.

Peu après, ce problème a été étudié par Codd [12] et par Garey et Graham [28] pour des applications en relation avec les systèmes multiprogrammés et les systèmes multiprocesseurs. Par la suite, ce problème a été utilisé par d'autres chercheurs ([36,38]) pour différentes variantes du problème de découpe (papier, bois, cuir, verre, etc.).

2.2. Famille de problèmes de découpe

Ces problèmes viennent d'un contexte pratique. Dans de nombreuses industries se pose en effet, à un moment ou un autre du stade de fabrication, le problème de découpe de pièces dans un support :

- Des pièces d'acier (verre ou papier) dans des plaques ;
- Des éléments de vêtement dans des lés de tissus ;
- Des éléments de chaussures dans des peaux ;
- Des panneaux dans des plaques de contre-plaqué ou d'aggloméré, etc.

Suivant la contrainte de découpe imposée et appliquée au support, les chutes dans le support seront plus ou moins importantes.

Un plan de découpe entraîne un coût (perte), et un bon choix provoquant peu de perte de matière, peut se traduire par des gains substantiels. Comme nous l'avons souligné dans l'introduction, les problèmes de découpe sont de nature diverse. Cette diversité est reliée au support ou matériel considéré, aux types d'objets (qu'on appellera pièces) à placer et aux contraintes imposées sur la façon de découper le(s) support(s).

Formellement, une instance du problème de découpe est définie par :

- Un (ou plusieurs) support (s) ;
- Un ensemble d'objets (appelés pièces) ;
- Un profit associé à chaque objet.
- Un procédé de découpe

2.2.1. Support de découpe

Concernant le support de découpe, nous retenons les spécificités suivantes.

a) *Forme géométrique* : Si la forme du support est rectangulaire. Il est appelé *rectangle initial*. Dans ce cas, ce support est distingué par ses dimensions *(L,H)* où *L* et *H* sont respectivement sa longueur et sa hauteur. On parle alors de *problèmes de découpe à deux dimensions*.

Dans de nombreux cas, on pourra "travailler" dans une seule dimension. La dimension hauteur (ou la longueur) est négligée si la hauteur (ou la longueur) est très grande par rapport aux dimensions des pièces à produire. Dans ce cas, le support sera représenté par une bande.

Un support de dimension *(L,H,E)* désigne un support de base rectangulaire de dimension *(L,H*) et d'épaisseur *E*. Le problème de découpe est alors à trois dimensions.

b) *Homogénéité et régularité* : Le support est homogène, si toute sa surface est la même et il est régulier, si le support ne présente aucun défaut. Il n'y a pas alors de restriction sur la position des pièces.

c) *Dissymétrie :* Quelque fois, sur des supports comme des tissus à motifs dissymétriques, les pièces à découper ont souvent une orientation imposée. Dans ce cas on parlera de pièce *fixée*, par opposition au cas où une *rotation* de 90^0 serait permise.

2.2.2. Pièces à découper

Les seules caractéristiques essentielles des pièces à découper, sont liées à sa *forme géométrique*, au respect des *contraintes liées au support* et à son *coût* de production.

Nous considérons que la forme d'une pièce *i* est rectangulaire (resp. Circulaire) de dimension (l_i, h_i) (resp. de rayon r_i). On dit que les pièces sont fixées si l'orientation est imposée, sinon les rotations sont permises et la pièce peut être pivoté de 90^0.

De plus , on dit que le problème de découpe est *non pondéré* si le coût de production de la pièce est exactement sa surface et *pondéré* sinon.

2.2.3. Types de découpe

Nous retenons les trois types de découpe suivants :

(a) *Découpe guillotine* (voir la Figure 2.1.(*a*)) :

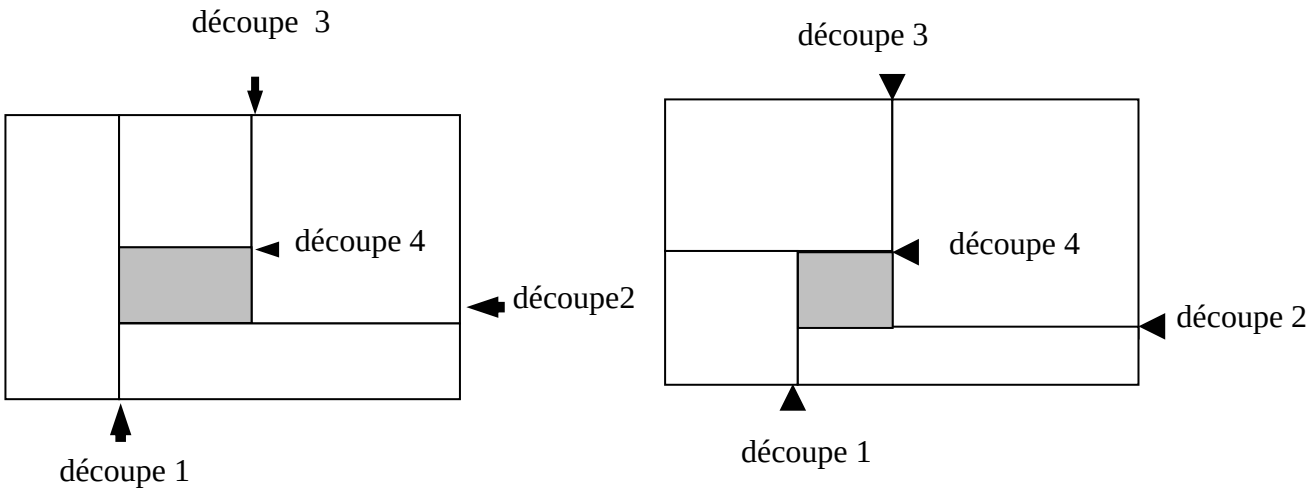
Sur une plaque rectangulaire, la découpe est effectuée d'une seule tranche en allant d'un côté de la plaque rectangulaire à son opposé, horizontalement ou verticalement.

(b) Découpe non- guillotine (voir la Figure 2.1.(*b*)) :

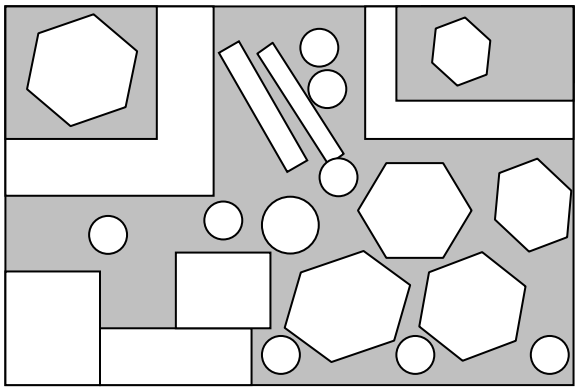
En générale, cette découpe engendre une solution meilleure qu'une solution produite par les découpes du type guillotine. En effet, elle consiste à utiliser le même procédé que dans la découpe guillotine, mais elle peut être effectuée tout en marquant des arrêts alternant coupe verticale et coupe horizontale.

(c) *Découpe non- orthogonale* (voir la Figure 2.1.(c)) :

Les pièces peuvent être pivotées et translatées (les rotations sur les pièces sont permises). Généralement, ce type de découpe est typique à la découpe au laser : un bras pivotant, qui se déplace dans tous les sens à une vitesse variable, effectuant les découpes. Quelques fois, on est confronté en plus au problème d'optimiser la durée du trajet à effectuer par la découpe.



(a) (b)



(c)

Figure 2.1 : *(a) la découpe guillotine.*
(b) la découpe non- guillotine.
(c) la découpe non- orthogonale.

Nombre d'auteurs se sont intéressés à l'étude du problème, en supposant que la contrainte imposée sur la découpe est du type ''guillotine'' (par exemple, Gilmore et Gomory [30], Herz [40], Hifi [44], Adamowicz et Albano[1], Dyson et Gregory [18], Christofides et Whitlock [11]).

En 1985, Beasley [8] s'est intéressé à l'étude du problème en considérant que la contrainte de découpe est du type ''non- guillotine''. Par la suite, d'autres chercheurs se sont intéressés à l'étude de ce problème (par exemple, Daniels et Ghandforoush [14], Hadjiconstantinou et Christofides [35], Arenales et Morabito [3]).

D'autres auteurs se sont aussi intéressés à l'étude du problème de découpe en imposant la découpe du type ''non- orthogonale'' . Parmi ceux qui apparaissent dans la littérature, Heassler [37] (par l'utilisation de la programmation non linéaire), Biro et Boros [9] (en se servant d'une modélisation sous forme d'un réseau de flot), Rinnoy Kan, De Wit et Wijmenga [56], ainsi que Dowsland et Dowsland [16].
Par ailleurs, Hahn [36] s'est plutôt intéressé à la résolution du problème de découpe en considérant que la contrainte de découpe est du type guillotine et le support n'étant pas régulier, c'est-à-dire qu'il existe des points défectueux (surface non utilisable) sur ce support.

2.3. Classification des problèmes de découpe

Vu la diversité des problèmes de découpe, il nous a semblé intéressant de donner une classification des problèmes afin de pouvoir les identifier plus facilement. Nous proposons une classification générale pour ces problèmes de découpe en nous appuyant sur la représentation symbolique suivante :

$\left|S_{F,ns}^{nh,nr}\right|P_{F,B,R}\left|C_{D,NC}\right|D\left|\right.$.

Cette classification regroupe toutes les informations essentielles sur le problème à traiter. Les informations sur le *support* considéré, sur les *pièces* à placer et sur la contrainte de *découpe* utilisée, y sont représentées.
Les informations traduites par un symbole sont résumées comme suit (dans l'ordre) :

$S_{F,ns}^{nh,nr}$:*S* spécifie le support considéré.

Si ce support est un rectangle (ou carré), alors nous affectons à l'indice *F* la lettre *R*. Dans le cas où le support n'est pas rectangulaire, il suffit de donner une lettre pour spécifier la forme (par exemple, *B* pour spécifier une bande).

ns est le nombre de supports considéré. Si le nombre de supports est 1, alors *ns* prendra la valeur 1.

Le symbole *nh* (resp. *nr*) est utilisé si le support est non homogène (resp. non régulier).

$P_{F,B,R}$ *: P* spécifie les pièces à placer.

L'indice *F* prend la lettre *R* si la pièce est rectangulaire, *Q* pour quelconque et *C* pour circulaire.

Le symbole *B* est utilisé pour indiquer qu'il y a une contrainte sur le nombre de pièces à placer dans le support considéré. *B* est remplacé par *b* si la contrainte existe . *R* permet d'indiquer si la rotation est possible. L'indice *R* est remplacé par *r* si la rotation est permise.

$C_{D,NC}$ *: C* spécifie les contraintes sur le type de découpe utilisée.

L'indice *D* pend la lettre *G* pour une découpe guillotine, *NG* pour une découpe *non-guillotine* et *NO* pour une découpe *non- orthogonale*.

Le symbole *NC* spécifie, s'il existe une contrainte sur le nombre de fois où le changement de découpe doit être effectué.

Par exemple, si on applique des découpes de type guillotine et que la machine est programmée pour effectuer un seul changement (pour des raisons de coût et de temps), on sera obligé de regrouper tout ce qui peut être découpé dans une première vague de découpe, puis faire le changement et refaire les découpes. Dans ce cas, on est limité à deux changements de découpe et donc le symbole *NC* prend la valeur 2.

Par défaut, le symbole *NC* est négligé lorsqu'on considère le cas général, c'est-à-dire le nombre de changement de la découpe n'est pas limité.

D : spécifie la dimension du problème.

S'il s'agit d'un problème en deux dimensions, *D* prend la valeur 2.

Dans le tableau 2.1, nous avons essayé de classer l'ensemble des problèmes répertoriés et bien connus dans la littérature.

Evidemment, nous nous sommes limités à quelques problèmes à deux dimensions à découpe guillotine ou non- orthogonale.

Problème	*Support*		*Pièce*			*Découpe*		
	Forme support	Nombre supports	Forme Pièce	rotation	Borne Pièce i	Contrainte Découpe	Nombre de chargements	Dim.
Pb.1	Rectangle	1	Rectangle	non	Non	Guillotine	∞	2
Pb.2	Rectangle	1	Rectangle	non	Non	Guillotine	K< ∞	2
Pb.3	Rectangle	1	Rectangle	non	Oui	Guillotine	∞	2
Pb.4	Rectangle	1	Rectangle	non	Oui	Guillotine	K< ∞	2
Pb.5	Rectangle	m ≥ 2	Rectangle	non	Non	Guillotine	∞	2
Pb.6	Rectangle	m ≥ 2	Rectangle	non	Non	Guillotine	K< ∞	2
Pb.7	Rectangle	m ≥ 2	Rectangle	non	Oui	Guillotine	∞	2
Pb.8	Rectangle	m ≥ 2	Rectangle	non	Oui	Guillotine	K< ∞	2
Pb.9	Rectangle	1	Rectangle	Oui	Non	Guillotine	∞	2
Pb.10	Rectangle	1	Rectangle	Oui	Non	Guillotine	K< ∞	2
Pb.11	Rectangle	1	Rectangle	Oui	Oui	Guillotine	∞	2
Pb.12	Rectangle	1	Rectangle	Oui	Oui	Guillotine	K< ∞	2
Pb.13	Rectangle	m ≥ 2	Rectangle	Oui	Non	Guillotine	∞	2
Pb.14	Rectangle	m ≥ 2	Rectangle	Oui	Non	Guillotine	K< ∞	2
Pb.15	Rectangle	m ≥ 2	Rectangle	Oui	Oui	Guillotine	∞	2
Pb.16	Rectangle	m ≥ 2	Rectangle	Oui	Oui	Guillotine	K< ∞	2
Pb.17	Rectangle	1	Circulaire		Non	Non- orthogonale		2
Pb.18	Rectangle	1	Circulaire		Oui	Non- orthogonale		2
Pb.19	Bande	1	Rectangle	Non	Oui	Guillotine	∞	2
Pb.20	Bande	1	Rectangle	Non	Oui	Guillotine	K< ∞	2
Pb.21	Bande	1	circulaire		Oui	Non- orthogonale		2
Pb.22	Bande	1	Rectangle	Oui	Oui	Guillotine	∞	2
Pb.23	Bande	1	Rectangle	Oui	Oui	Guillotine	K< ∞	2

Tableau 2.1 : Classification de quelques problèmes de découpe à deux dimensions.

Le nombre de problèmes peut être facilement doublé si nous considérons que la découpe imposée est du type *non- guillotine* ou *non- orthogonale*. Par ailleurs, pour les problèmes à trois dimensions, il suffit de reprendre quelques problèmes du tableau 2.1 et de remplacer la dimension 2 (dernière colonne) par 3.

Nous avons débuté le2.1 par le problème **Pb.1** qui définit le **problème standard** de la littérature, connu sous le nom du **problème de découpe à deux dimensions sans contraintes**. En utilisant la classification décrite ci-dessus (par la formule générale), nous obtenons sa représentation symbolique dans la première ligne du tableau 2.2, donnée par $/S_{R,1}/P_R/C_G/2/$, ce qui se lit bien comme un problème de découpe avec : $(S_{R,1})$ un support rectangulaire (homogène et régulier), (P_R) des pièces rectangulaires à placer (aucune borne sur les pièces n'étant imposée et les pièces sont fixées), (C_G) une découpe du type guillotine (le nombre de changements sur la découpe n'étant pas imposé), (/2/) en deux dimensions.

Problème	*Représentation symbolique*
Pb.1	$/S_{R,1}/P_R/C_G/2/$
Pb.2	$/S_{R,,1}/P_R/C_{G,K}/2/$
Pb.3	$/S_{R,1}/P_{R,b}/C_G/2/$
Pb.4	$/S_{R,1}/P_{R,b}/C_{G,K}/2/$
Pb.5	$/S_{R,m}/P_R/C_G/2/$
Pb.6	$/S_{R,m}/P_R/C_{G,K}/2/$
Pb.7	$/S_{R,m}/P_{R,b}/C_G/2/$
Pb.8	$/S_{R,m}/P_{R,b}/C_{G,K}/2/$
Pb.9	$/S_{R,1}/P_{R,r}/C_G/2/$
Pb.10	$/S_{R,1}/P_{R,r}/C_{G,K}/2/$
Pb.11	$/S_{R,1}/P_{R,b,r}/C_G/2/$
Pb.12	$/S_{R,1}/P_{R,b,r}/C_{G,K}/2/$
Pb.13	$/S_{R,m}/P_{R,r}/C_G/2/$
Pb.14	$/S_{R,m}/P_{R,r}/C_{G,K}/2/$
Pb.15	$/S_{R,m}/P_{R,b,r}/C_G/2/$
Pb.16	$/S_{R,m}/P_{R,b,r}/C_{G,K}/2/$
Pb.17	$/S_{R,1}/P_C/C_{NO}/2/$
Pb.18	$/S_{R,1}/P_{C,b}/C_{NO}/2/$
Pb.19	$/S_{B,1}/P_{R,b}/C_G/2/$
Pb.20	$/S_{B,1}/P_{R,b}/C_{G,K}/2/$
Pb.21	$/S_{B,1}/P_{C,b}/C_{NO}/2/$
Pb.22	$/S_{B,1}/P_{R,b,r}/C_G/2/$
Pb.23	$S_{B,1}/P_{R,b,r}/C_{G,K}/2/$

Tableau 2.2 : Représentation symbolique de quelques problèmes de découpe à deux dimensions.

- **Pb.5** $/S_{R,m}/P_R/C_G/2/:$ est le problème de découpe guillotine généralisé sans contraintes.
- **Pb.7** $/S_{R,m}/P_{R,b}/C_G/2/:$ est le problème de découpe guillotine généralisé avec contraintes sur les pièces.
- **Pb.8** $/S_{R,m}/P_{R,b}/C_{G,K}/2/:$ est le problème de découpe guillotine généralisé, avec contraintes sur les pièces et contrainte sur le niveau de changement de la découpe.

Par ailleurs, en plus de la représentation que nous avons donné, nous pouvons toujours rajouter des spécifications ou des contraintes lors de la définition du problème (par exemple, une pièce du type *i* ne doit pas être placée à côté d'une autre pièce du type *j*, avec $i \neq j$, etc.)

2.4. Classe des problèmes de découpe guillotine

Dans notre étude, nous nous sommes intéressés aux variantes du problème de découpe vérifiant les hypothèses suivantes :

- Le support à découper est de forme rectangulaire
- Les découpes sont du type guillotine.
- Les pièces à découper sont fixées, une pièce de longueur *l* et de hauteur *h* est différente de celle de longueur *h* et de hauteur *l*.
- Les dimensions du support et des pièces à découper sont tous des entiers

2.4.1. Présentation des problèmes de découpe guillotine

Formellement, une instance du problème de découpe guillotine à deux dimensions consiste à découper un support rectangulaire appelé *rectangle initial* de dimensions *(L, H)* en petites pièces rectangulaires de dimensions (l_i, h_i) et chacune de profit c_i pour $i=1,\ldots, n$. L'objectif étant de maximiser la somme des utilités des pièces produites sur le(s) support(s) à découper. De plus, on dit que le problème de découpe est *non pondéré*, si le profit de chaque pièce est égal à sa surface et *pondéré* sinon.

La contrainte imposée sur la manière de découper le(s) support(s) disponible(s) étant bien évidemment du type guillotine. Une découpe qui produit deux rectangles est appelé *découpe guillotine*, précisément une *découpe guillotine* sur une plaque rectangulaire s'effectue d'une seule tranche en allant d'un côté de la plaque rectangulaire à son opposé horizontalement ou verticalement.

Un ensemble de pièces rectangulaires définit un modèle de découpe si ces pièces peuvent être obtenues par une séquence de découpes possibles sur le support disponible. Toutes les pièces produites qui différent des pièces à découper sont considérées comme des chutes.

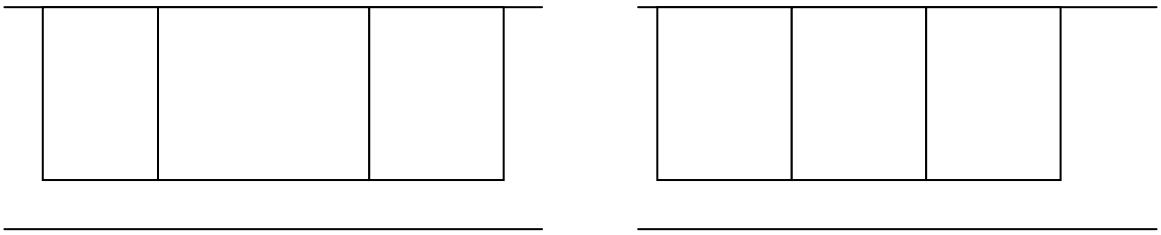
2.4.2. Modèles de bandes

Une possibilité conception de modèles de découpe passe par la génération de modèles de bande dont nous distinguons les catégories suivantes :

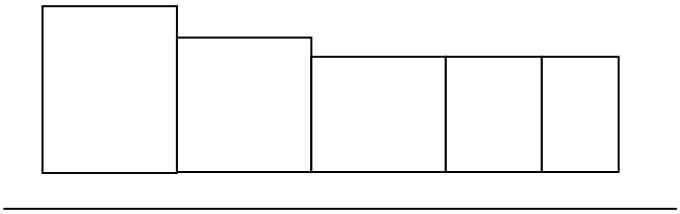
- *Une bande générale horizontale* (resp. *verticale*) est une bande composée d'au moins deux pièces de *hauteur (resp. longueur)* différentes.

1. *Une bande uniforme horizontale* (resp. *verticale*), *(voir figure 2.2(a)* est une bande où toutes les pièces participantes sont de même *hauteur (resp. longueur).*

2. *Une bande homogène horizontale (resp. verticale), (voir figure 2.2(b)* est une bande où il n'y a qu'un *seul type de pièces* participant.
3. *Une bande générale horizontale* (resp. *verticale*) *(voir figure 2.2 (c)* est une bande composée d'au moins deux pièces de *hauteur (resp. longueur)* différentes.
4. *Une bande optimale* de longueur α et de hauteur h (resp. de hauteur β et de longueur l) est une bande générale occupant la surface maximale de la bande (α,h) (resp.(l,β)).



(a) bande uniforme (b) bande homogène



(c) bande générale

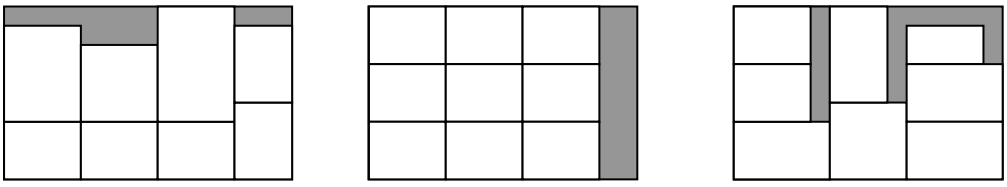
Figure 2.2. Modèles de bande

2.4.3. Modèles de découpe

Un modèle de découpe réalisable s'obtient donc naturellement, comme combinaison d'un certain nombre de bandes horizontales ou verticales. Nous retenons les différents types de modèles de découpes suivants :

1. Un modèle de découpe *uniforme (figure 2.3(a))* est un modèle de découpe réalisable constitué uniquement de bandes uniformes.
2. Un modèle de découpe *homogène (figure 2.3(b))* est une dissection uniforme caractérisée par la répétition d'un seul type de pièces.

3. Un modèle de découpe *général* (ou dissection générale) *(figure 2.3(c))* est un modèle constitué par la combinaison de bandes générales verticales et horizontales.



(a) (b) (c)

Figure 2.3. Modèles de découpe

2.4.4. Points de découpe

Les points de découpe désignent les positions de découpe permettant l'emplacement des pièces autorisées sur le (sous) rectangle à découper, ces points de découpe sont générés par un processus qui garantit un nombre fini de découpe. Le procédé utilisé a cet effet fut introduit par Herz [40], repris ensuite par un certain nombre d'auteurs dans la conception d'algorithmes récursif basés sur les possibilités de découpe sur le rectangle initial. Ce procédé a pour conséquence :

- *Sauts de points de découpe* : Les découpes verticales sont limitées par l'ensemble P qui forme les combinaisons linéaires entre les différentes longueurs de chaque type de pièces de **S**, de même que les découpes horizontales sont bornées par l'ensemble Q des combinaisons linéaires sur les différentes hauteurs des pièces de **S**, ces ensembles sont définis par :

$$P = \left\{ p \in N \,/\, p = \sum_{i=1}^{n} l_i x_i \le L, x_i \in N \right\}$$

$$Q = \left\{ q \in N \,/\, q = \sum_{i=1}^{n} h_i x_i \le H, x_i \in N \right\}$$

- *Effets de symétrie* : permet de limiter les points de découpes (verticales et horizontales) à la moitié de la longueur $\left(\left\lfloor \frac{\alpha}{2} \right\rfloor \text{tel que} \, \alpha \le L \right)$ et de la hauteur $\left(\left\lfloor \frac{\beta}{2} \right\rfloor \text{tel que} \, \beta \le H \right)$ du sous rectangle (α, β) en cours de dissection, éliminant ainsi les formes de pièces dupliquées.

Chapitre 3
Un nouvel algorithme pour la résolution du problème de découpe (non) pondéré à deux dimensions

3.1. Introduction

Le problème de découpe est un problème NP-complet avec de nombreuses applications industrielles et commerciales. Dans ce chapitre on s'intéresse au problème de découpe guillotine non contraint à deux dimensions qui est la forme standard du problème de découpe dans la littérature. Il s'agit d'optimiser l'utilisation d'une entité disponible en taille limitée en y plaçant des sous- entités prédéterminées. L'entité disponible est sous forme rectangulaire de dimensions fixées, qu'on appelle rectangle initial. Les sous- entités ont aussi des formes rectangulaires qu'on appelle pièces disponibles. L'objectif étant de découper le rectangle initial en pièces appartenant à l'ensemble des pièces disponibles ***S*** tout en minimisant la surface des pièces produites n'appartenant pas à cet ensemble, appelé chute (version non pondérée du problème) ou en maximisant le profit total sur l'ensembles des pièces à découper (version pondérée). Un choix du matériel de découpe est indispensable, dans notre étude nous supposons que les découpes sont du type guillotine, ce qui consiste à prendre la découpe d'un côté à son opposé parallèlement aux deux autres.

Ce chapitre est disposé de la manière suivante : la section 3.2 comporte une présentation détaillée des deux versions pondérée et non pondérée du problème de découpe guillotine sans contraintes à deux dimensions. La section 3.3 comporte une description de l'algorithme de Herz [40] pour la version non pondérée du problème. Dans la section 3.4, nous présentons une nouvelle approche de résolution du problème de découpe sans contraintes applicable aux deux versions pondérée et non pondérée, notre méthode s'appuie sur des techniques de programmation dynamique, et une recherche arborescente avec critère de sélection du meilleur d'abord. La section 3.5 est consacrée au développement d'une heuristique de recherche basée sur les stratégies du hill climbing [34], il s'agit de techniques dérivées des méthodes d'intelligence artificielle. Finalement dans la section 3.6 on présente une étude expérimentale qui justifie l'efficacité des approches proposées comparées aux meilleurs algorithmes rencontrés dans la littérature, en l'occurrence celui de Beasley[8] pour le cas pondéré, et celui de Morabito[53] et al pour le cas non pondéré.

3.2. Présentation du problème

Une instance du problème de découpe sans contraintes qu'on note par *TDGC* est définie par le triplet *(*R,***S**,*c)*. R =*(L,H)* est le rectangle initial , où *L* et *H* sont ses longueur et hauteur respectivement . Les sous–entités rectangulaires sont représentées par l'ensemble ***S*** *={p*$_1$*,...,p*$_n$*}* dans lequel chaque pièce p_i est de dimensions *(*l_i*,* h_i*) < (L,H).*

c= (c_1*,...,*c_n*)* est le vecteur poids (la pièce i a un poids *ci*).

Définition 3.2.1.

- La version non pondérée du problème sans contraintes est la version dans laquelle il n'existe pas de limitation sur le nombre d'apparitions de chaque type de sous entité sur l'entité disponible, le poids de chaque type de pièce est sa surface, et l'objectif étant de minimiser la chute sur l'entité en stock.
- La version pondérée du problème sans contraintes possède les mêmes caractéristiques que la version non pondérée, Cependant le vecteur des poids diffère. A chaque sous-entitée est affecté un poids indépendant de sa surface. L'objectif ici est de maximiser la somme des utilités produites par l'ensemble des sous-entitées.

Nous supposons au cours de notre étude des deux versions du problème de découpe sans contraintes que:

1.Toutes les découpes sont du type guillotine,

2. Les orientations des différentes pièces sont fixées, i.e., deux pièces de dimensions (*l*,*h*) et (*h*,*l*) sont différentes si $l \neq h$,

3. Toutes les dimensions sont des entiers positifs.

Définitions 3.2.2.

Soit *M* l'ensemble fini des vecteurs représentant les différents modèles de découpe réalisables pour le problème *TDGC*, on note un élément de *M* par $\xi = (\xi_1,...,\xi_n)$, où $\xi_i \in \angle$, désigne le nombre de répétition de la $i^{ème}$ pièce de ***S*** dans le modèle de découpe ξ .

(a) Version non pondérée: La fonction objectif est représentée par l'application *F* définie par :

$$F : M \longrightarrow N \qquad \text{telle que :}$$

$$F^* = F(\xi) = LH - \sum_{i=1}^{n} c_i \xi_i$$

où $c_i = l_i h_i$ désigne la surface de la pièce *i=1,...,n.*

Une solution optimale du problème TDGC dans ce cas, consiste à trouver le couple *(ξ*, F*)* tel que :

$$F(\xi^*) = \min_{\xi \in M} F(\xi)$$

L'utilité du modèle de découpe réalisable ξ* est égale à *F** de valeur minimum, *ξ** est le modèle de découpe optimal qui minimise la chute sur le rectangle initial.

(b) Version pondérée : La fonction objectif est représentée cette fois sous la forme suivante:

$$\mathrm{F} : M \longrightarrow N$$

telle que

$$F(\xi) = \sum_{i=1}^{n} c_i \xi_i$$

où $c_i \neq l_i h_i$ désigne le profit associé a la pièce *i=1,...,n.*.

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

$$F^* = F(\xi^*) = \max_{\xi \in M} F(\xi)$$

(c) Cas avec contraintes de niveau:

Une solution optimale dans ce cas est représentée par le couple *(ξ*,F*)* telle que $F^* = c^t \xi^*$ parmi tous les modèles de découpes possibles de l'ensemble M^s, où les éléments de M^s sont soumis à une contrainte qui limite le nombre total de découpes verticales et horizontales à un nombre maximum *s* fixé.

3.3. Algorithme de Herz

Parmi les algorithmes exacts rencontrés dans la littérature pour la résolution du problème de découpe à deux dimensions, celui de Herz [40] pour le cas non pondéré. Nous donnons une description détaillée de l'algorithme de Herz dans le but présenter les procédés de découpes sous forme d'arborescence.

L'algorithme de Herz s'applique uniquement pour la version non pondéré du problème *TDGC*. Le procédé utilisé par l'algorithme de Herz consiste à découper le rectangle initial *(L,H)* en effectuant des découpes du type guillotine afin de réaliser des pièces appartenant à l'ensemble ***S*** tout en minimisant la chute.

La propriété suivante est à la base de l'algorithme de Herz, elle consiste à essayer toutes les premières découpes et ensuite à retenir celle qui donne la valeur total maximale des pièces rentrantes, pour les deux sous-rectangles produits. Ceci est appliqué de manière récursive sur chaque sous-rectangle.

3.3.1. Propriété récursive (Herz)

Une solution optimale est soit constituée d'une pièce appartenant à ***S,*** soit sa première découpe produit deux sous-rectangles qui sont aussi découpés d'une manière optimale.

La procédure générale de l'algorithme avec la propriété récursive principale est présentée par une structure arborescente, imposant toutes les possibilités de découpes sur le rectangle initial, par la suite une méthode de limitation est imposée sur les découpes verticales et horizontales afin d'aboutir à une méthode de séparation et d'évaluation utilisant, la stratégie de développement en profondeur.

Les solutions optimales produites par l'algorithme de Herz ont une structure particulière, chaque pièce est placée toujours à gauche et en bas. Dans [40] l'auteur montre l'équivalence entre une solution optimale du problème et la solution optimale ayant une telle structure pour le même problème.

D'autre part, les effets de symétrie (2.3.1.) permettent de limiter les points de découpe à la moitié de la longueur et de la hauteur du rectangle initial ou d'un sous-rectangle en cours de dissection.

3.3.2. Bornes de l'algorithme :

Pour accélérer le processus de découpe, l'algorithme utilise certaines bornes faciles à calculer ; les plus importantes sont les suivantes :

a) Pour le rectangle initial ou un sous rectangle quelconque produit au cours du processus de découpe, on calcule la valeur de la meilleur découpe homogène, composée d'une seule pièce de l'ensemble ***S***.

b) Au cours du processus, si pour un (sous) rectangle une découpe est pleine, c'est à dire la chute est égale à zéro, alors on arrête la division de ce (sous) rectangle car la solution partielle est optimale.

c) Si pour un (sous) rectangle, la valeur d'une découpe est connue et qu'elle est supérieure à la somme de la solution optimale du sous-rectangle gauche et de la surface du sous-rectangle droit (on suppose que la découpe est verticale), alors il est inutile d'envisager la découpe du sous-rectangle droit. De même, lorsqu'on effectue une découpe horizontale on peut envisager d'ignorer la découpe du sous-rectangle du haut.

L'algorithme est présenté dans l'encadré 1. On note par $P_{\alpha\beta}$ l'ensemble des points représentants les combinaisons linéaires des longueurs des pièces rentrantes, dans le (sous) rectangle (α,β) ,limité à la moitié de $\alpha/2$. De manière similaire, on définit $Q_{\alpha\beta}$ l'ensemble des points limité à $\beta/2$.

Encadré 1

Entrées : *L,H* : les dimensions du rectangle initial.

n : le nombre de pièces à découper.

Sorties : La solution optimale notée par **OPT.**

1. Construire les ensembles *P* et *Q*.
2. **OPT** = *R(L,H,0)*
3. Fonction $R(\alpha, \beta, v_0)$: *entier*

si $v_0 \geq S_{\alpha\beta}$ *alors* sortir *avec* $R = 0$

sin *on*

poser $\alpha_0 = \sup\{x / x \leq \alpha, x \in P\}$ *et* $\beta_0 = \{y / y \leq \beta, y \in Q\}$

si la valeur de (α_0, β_0) *est connu alors* sortir *avec cette valeur*

sin *on*

calculer la meilleure découpe hom *ogène h*

si elle est pleine alors sortir avec $R = h$

sin *on poser* $V = h$

fsi;

fsi;

pour chaque $c \in P_{\alpha\beta}$

calculer $w = R(c, \beta, \max(V, v_0) - S_{(\alpha - c)\beta})$

et $V_1 = w + R(\alpha - c, \beta, \max(V, v_0) - w)$

si V_1 *est pleine alors sortir avec* $R = V_1$

sin *on* $V = \max(V,\ V_1)$

fsi;

répéter la boucle pour tout $d \in Q_{\alpha\beta}$

frépéter;

fsi;

sortir avec la solution optimale représenté par R.

La surface du (sous)rectangle (α, β) est notée par $S_{\alpha,\beta}$. On note par *V* la valeur de la meilleure solution en cours. La valeur v_0 est la différence entre la valeur *V* et la surface d'un sous-rectangle considéré comme une borne inférieure pour le deuxième sous-rectangle. Elle peut être aussi la différence entre *V* et la meilleure solution d'un des deux sous-rectangles qui détermine la valeur à atteindre par le deuxième sous-rectangle.

Si au cours du processus on atteint l'optimum d'un sous-rectangle (α,β) alors on sauvegarde la valeur de la solution optimale ainsi que les dimensions du sous-rectangle (α_0,β_0) tels que

$$\alpha_0 = sup\{x / x \le \alpha, x \in P\} \quad et \ \beta_0 = \{y / y \le \beta, y \in Q\}$$

puisque les deux sous-rectangles ont la même solution optimale.

3.4. Algorithme *BFS*

Dans la section 3.3. nous avons exposé l'algorithme de Herz basé sur une méthode d'évaluation et de séparation pour la résolution du problème de découpe non contraint à deux dimensions dans le cas non pondéré ; i.e. que le profit c_i de chaque type de pièce est exactement sa surface. Nous avons vu que les avantages de l'algorithme repose sur l'utilisation de quelques bornes facilement calculables, relatives à la surface des pièces de l'ensemble ***S***, ainsi qu'à la surface du rectangle initial.

Cependant, les bornes ne sont pas valides pour le cas non pondéré, i.e. lorsqu'on affecte un poids à chaque type de pièces indépendant de sa surface. C'est sur ce point qu'apparaissent les complications des méthodes existantes qui tentent de résoudre le problème même de manière approchée, comme l'approche de la programmation linéaire généralisée [30], ou les procédures d'ordonnancement sur les profits des pièces [28].

3.4.1. Principe de l'algorithme

Notre approche se base sur une procédure constructive permettant d'obtenir des modèles de découpe guillotine sur l'entité initial R au moyen de constructions verticales et horizontales que nous détaillons dans les définitions suivantes:

Définitions 3.4.1.

a) Une construction verticale de deux (sous-) rectangles de dimensions (α_1,β_1) et (α_2,β_2) est représentée par le nouveau (sous-) rectangle de dimensions $(max\{\alpha_1,\alpha_2\},\beta_1+\beta_2)$.

b) Une construction horizontale produit un nouveau (sous-) rectangle de dimensions $(\alpha_1+\alpha_2,max\{\beta_1,\beta_2\})$.

Dans les deux cas de figures, chaque (sous-) rectangle R résultant de construction verticale ou de construction horizontale vérifiant $(l_R,h_R) \leq (L,H)$ est considéré comme un *rectangle guillotine*.

Nous utilisons la propriété récursive caractérisant les types de découpes guillotine. Pour le problème de découpe à deux dimensions, c'est la solution optimale du rectangle initial R obtenue par la somme des solutions optimales des deux sous-rectangles produits par une des découpes guillotines. Cette observation est à la base de la plupart des arborescences de recherche pour la résolution des problèmes de découpe.

1. Considérons toutes les possibilités de découpe verticales et horizontales sur le rectangle initial. Pour chaque dissection, considérons les deux sous-rectangles résultants, et pour chacun d'eux toutes les possibilités de découpes et ainsi de suite.

2. Il n'est pas nécessaire d'effectuer des dissections sur le sous-rectangle constitué d'une seule pièce de l'ensemble $\boldsymbol{S}$.

Cette procédure nous conduit à atteindre la solution optimale de chaque sous-rectangle, et par conséquent celle du rectangle initial.
L'approche utilise le principe du type « branch and bound » qui consiste à partir d'un rectangle guillotine R (placé dans le rectangle initial R), on construit un autre rectangle guillotine de dimensions supérieures. Le rectangle obtenu est une combinaison entre R et un autre rectangle déjà construit, en utilisant les constructions horizontales et/ou verticales. A chaque rectangle guillotine R, on fait correspondre un *fonction intermédiaire g(R)* qui représente la somme des profits des pièces contenues dans R. On lui fait correspondre une autre *fonction complémentaire h(R)* qui représente la valeur optimale du modèle de découpe sur le reste de la surface du rectangle initial $surface(\mathsf{R}) \setminus surface(R)$, représentée par la région F sur la figure 3.1. On désigne par :

$$f(R) = g(R) + h(R)$$

la valeur *maximale* de **l'ensemble des modèles de découpe** contraints de contenir le rectangle guillotine R.

Généralement, il n'est pas évident de calculer la valeur de *h(R)* en un temps raisonnable. Pour cela, on se limite à l'estimation d'une borne supérieure

$$f'(R) = g(R) + h'(R)$$

sur le rectangle guillotine *R*.

Ce processus de construction est schématisé par une arborescence dans laquelle chaque nœud désigne un niveau de construction.

Les évaluations des bornes inférieures et supérieures sont réalisées en modélisant le problème sous forme de problèmes de sac à dos unidimensionnels résolus par la programmation dynamique.

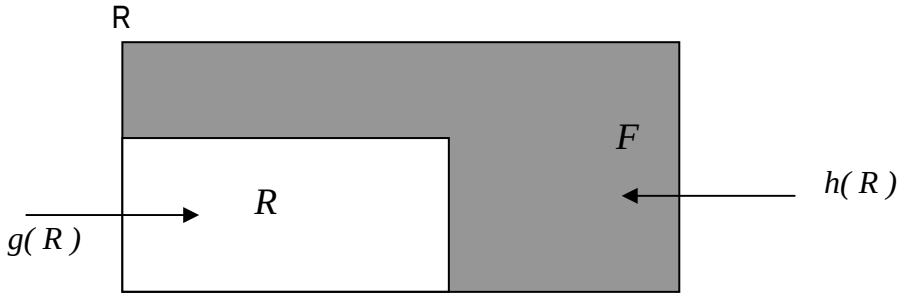


Figure 3.1: Paramètres d'évaluation d'un rectangle guillotine

De plus à chaque niveau de l'arborescence, nous exploitons les sous problèmes induits (par la programmation dynamique) pour calculer les bornes inférieures et supérieures des autres sommets intérieurs. Par ailleurs ces bornes sont aussi utilisées pour la stérilisation de certaines branches par :

- La récupération d'une solution à partir d'un niveau inférieur de l'arborescence.
- La réutilisation des résultats issus de la programmation dynamique des nœuds en cours d'exploration.

3.4.2. Critères d'optimalité

Nous développons dans ce qui suit les critères d'optimalité, concernant le calcul des bornes inférieures et supérieures qui apporte une remarquable accélération à l'algorithme.

L'évaluation des bornes, peut être obtenue par l'application d'une méthode rapide garantissant une solution partielle efficace. Nous exploitons l'idée des problèmes de sac à dos, afin de concevoir une procédure de calcul des bornes supérieure et inférieure initiales (pour le rectangle initial) et intermédiaires (pour les sommets intérieurs de l'arborescence),

représentant les sous-rectangles guillotines générés par construction horizontale et/ou verticale.

L'utilisation de la programmation dynamique sur le rectangle initial, conduit à la récupération de toutes les valeurs et les compositions des solutions sur certains sommets intérieurs de l'arborescence.

3.4.2.1. Bornes supérieures sur les sommets

Un modèle de découpe optimale pour la région *F* ne dépasse en aucun cas la surface de celle-ci, les résultats suivants donne une description du traitement de la région complémentaire *F* en tant que surface (barre), ainsi une estimation d'une borne supérieure pour *h(R)* peut être obtenue par la résolution d'un problème de sac à dos unidimensionnel relaxé.

Théorème 3.1. : Soit *R* un rectangle guillotine de dimensions (l_R, h_R), une borne supérieure pour *f(R)* est donnée par :

$$f'(R) = g(R) + h'(R)$$

où : $$h'(R) = S(F) \cdot max\left\{ c_j / l_j h_j , j \in F \right\}$$

Preuve:

Considérons *R* un rectangle guillotine de dimensions (l_R, h_R). Par définition, la fonction *complémentaire h(R)* représente la valeur optimale du modèle de découpe sur le reste de la surface du rectangle initial (non découpé) *S(F)= surface*(R) \ *surface*(*R*).

Etant donné, que tout modèle de découpe faisable sur le rectangle initial R ne doit guère dépasser la surface de R. Un modèle de découpe faisable pour la région *F* doit vérifier la contrainte exprimée sous la forme:

$$\sum_{j \in F} (l_j h_j) x_j \leq S(F)$$

Ce qui signifie que la somme des surfaces des pièces rentrante dans la région *F*, ne dépasse pas la surface de celle-ci. Par conséquent, une borne supérieure *h'(R)* pour la valeur de la solution optimale *h(R)* du modèle de découpe sur la région *F*, peut-être obtenue en considérant le problème de sac à dos relaxé de sa contrainte d'intégrité suivant:

$$K(F)\begin{cases} Max \sum_{j\in F} c_j x_j \\ s.c. \sum_{j\in F} (l_j h_j) x_j \le S(F) \\ 0 \le x_j \le nb_j(F_1 \cup F_2), j \in F \end{cases}$$

où: F_1 et F_2 sont les sous-rectangles représentés respectivement par $(L-l_R, H)$ et $(L, H-h_R)$. $nb_j(y)$ est le nombre maximum d'apparitions de la pièce j dans le sous-rectangle y.

Ce dernier a pour solution :

$$h'(R) = S(F) \cdot max\left\{ c_j / l_j h_j , j \in F \right\}$$

Le problème de sac à dos unidimensionnel $K(F)$, est une modélisation d'un problème de découpe à une dimension de la région F, représentée sous forme d'une bande de longueur $S(F)$ et de hauteur 1, chaque pièce rentrante dans F est représentée par une barre de longueur $l_j h_j$ et de hauteur 1.

Corollaire 3.1. :

Une borne supérieure initiale pour le rectangle R est donnée par :

$$B_{sup}(R) = L.H. \max\left\{ c_j / l_j h_j , j \in S \right\}$$

3.4.2.2. Bornes inférieures

La borne de départ de notre approche nécessite la résolution de quatre problèmes de sac à dos unidimensionnels, deux d'entre eux engendrent les différentes bandes optimales horizontales et verticales, deux autres sac à dos permettent de construire deux modèles de découpe réalisables, le premier est un modèle de découpe horizontal, obtenu comme combinaison des bandes optimales de différentes hauteurs, le deuxième est un modèle de découpe vertical, obtenu en combinant les bandes optimales verticales de différentes longueurs. La borne inférieure initiale, est choisie comme la valeur du meilleur modèle de découpe parmi ces modèles construits.

La modélisation des problèmes de sac à dos unidimensionnels repose sur les résultats suivants [25]

Lemme 3.1.:

Soient R = *(L,H)* le rectangle initial et $\boldsymbol{S} = \{(l_1, h_1), \cdots, (l_n, h_n)\}$ l'ensemble de pièces prises par ordre croissant selon les hauteurs. Toutes les bandes des différentes hauteurs et de longueurs α, avec $0 \le \alpha \le L$ sont obtenues en résolvant par la programmation dynamique un seul problème de sac à dos unidimensionnel défini part :

$$\mathrm{K} \begin{cases} Max \sum_{(l_i,h_i)\in S} c_i \xi_i \\ \sum_{(l_i,h_i)\in S} l_i \xi_i \le L \quad \xi_i \ entier. \end{cases}$$

Où ξ_i désigne le nombre d'apparitions de la longueur l_i ,et c_i dénote le profit de la pièce (l_i, h_i).

Preuve :

Soit *r* l'ensemble des différentes hauteurs des pièces de ***S***, avec

$$h_{k_1} \le h_{k_2} \le \cdots \le h_{k_n}$$

Pour chaque *i=1,...,r,* on définit les ensembles :

$$S_{k_i} = \{(l_j, h_j) \in S / \ h_j < h_{k_i}\}$$

une bande optimale de longueur α, avec $0 \le \alpha \le L$, et de hauteur h_{k_i}, est obtenue par la résolution du problème de sac à dos unidimensionnel :

$$\mathrm{K}_{\alpha}^{\mathrm{k_i}} \begin{cases} Max \sum_{(l_i,h_i)\in S_{k_i}} c_i \xi_i \\ s.c \sum_{(l_i,h_i)\in S_{k_i}} l_i \xi_i \le \alpha \quad \xi_i \ entier \end{cases}$$

où ξ_i est le nombre d'occurrence de l_i et c_i le profit de la pièce (l_i, h_i)

Si on résout le problème $K_{\alpha}^{k_i}$ en substituant la variable α par L, à l'optimum en utilisant des méthodes de la programmation dynamique [59], alors les solutions optimales de tous

les problèmes de sac à dos $K_{\alpha}^{k_i}$, avec $\alpha \leq L$ seront disponibles, en d'autres termes toutes les bandes optimales de longueur α et de hauteur h_{k_i}.

De la même façon, en supposant que les pièces sont rangées par ordre croissant sur les hauteurs, on peut obtenir les solutions de tous les problèmes $K_L^{k_i}$ pour *i=1,…,r en* résolvant seulement le problème $K_L^{k_r}$ qui est identique au problème K.

Exemple 3.1 :

Soit l'ensemble des pièces ***S*** ={(6,5),(4,5),(1,4)}, on peut constater que toutes les bandes présentées dans la figure 3.2. constituent des solutions faisables du problème de sac à dos suivant :

$$K_{11}^{5} \quad \begin{cases} max\, 30\xi_1 + 20\xi_2 + 4\xi_3 \\ s.c \quad 6\xi_1 + 4\xi_2 + \xi_3 \leq 11 \\ \xi_1, \xi_2, \xi_3 \in N \end{cases}$$

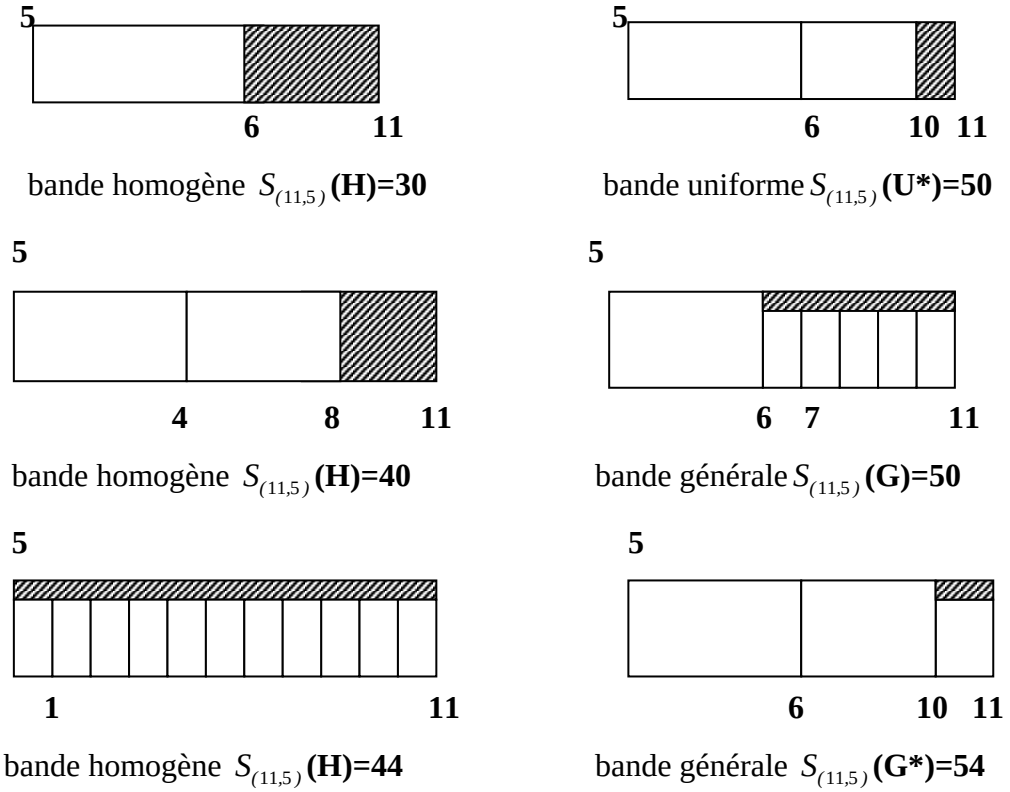


Figure 3.2. Représentation de quelques bandes horizontales de longueur 11 et de hauteur 5

La bande optimale de valeur $S_{(11,5)}$(G*)=54 est la solution optimale du problème K_{11}^5.A l'intérieur de cette même bande (conséquence du principe de la programmation dynamique), on retrouve la bande optimale de longueur 10 et de hauteur 5 (solution optimale du problème K_{10}^5) qui est une bande uniforme de valeur $S_{(10,5)}$(U*)=50, ainsi que la bande optimale de longueur 6, $S_{(6,5)}$(U*)=30 solution optimale de K_6^5.

Théorème 3.2.: Soient $E=\{(\alpha_1,\beta_1),\cdots,(\alpha_m,\beta_m)\}$ un ensemble fini de rectangles indépendants et $S=\{(l_1,h_1),\cdots,(l_n,h_n)\}$ un ensemble de pièces. Si le nombre d'apparitions de chaque pièce n'est pas borné supérieurement, alors une solution constituée par des bandes optimales pour le problème de découpe sur chaque rectangle, est obtenue en résolvant seulement $m+1$ problèmes de sac à dos unidimensionnels.

Preuve :

Soit *R= (L,H)* le rectangle fictif avec :

$$L=\max_{1\le j\le m}\{\alpha_j\} \quad \text{et} \quad H=\max_{1\le j\le m}\{\beta_j\}$$

On considère les r hauteurs différentes de S ordonnées selon un ordre croissant :

$h_{k_1}\le h_{k_2}\le\cdots\le h_{k_n}$ On note par $F_{k_i}(\alpha)$ le profit associé à chaque bande $S_\alpha^{k_i}$ avec:

$$F_{k_i}(\alpha)=\sum_{c_i\in S_\alpha^{k_i}} c_i$$

L'application du lemme 3.1. sur le rectangle fictif de longueur *(L,H)* fournit :

Toutes les bandes optimales de longueurs $\alpha_j\le L$ et de hauteurs $h_{k_i}\le\beta_j\le H$ pour *i=1,...s,* $s\le r$, avec leur valeur respective $F_{k_i}(\alpha_j)$.

Considérons un rectangle $(\alpha,\beta)\in E$. La résolution du problème $K_{\alpha\beta}$ décrit ci-dessus, produit un modèle de découpe général horizontal pour le rectangle (α,β).

$$K_{\alpha\beta}\begin{cases} max\sum_{i=1}^{s} F_{k_i}(\alpha)z_{k_i} \\ s.c \ \ \sum_{i=1}^{s} h_{k_i} z_{k_i} \le \beta, \ \ \alpha \le L, \beta \le H, s \le r \le n \\ \quad z_{k_i} \in N, pour \ \ i = 1,\cdots,s \end{cases}$$

Où s est le nombre de sous bandes optimales de hauteurs $h_{k_1} \le h_{k_2} \le \cdots \le h_{k_s} \le \beta$.

La résolution du problème $K_{\alpha\beta}$ revient à sélectionner les meilleures (sou-) bandes optimales qui forment la meilleure combinaison linéaire en hauteur.

Les solutions fournies par le problème $K_{\alpha\beta}$ sont des combinaisons linéaires de (sou-) bandes optimales d'utilités maximales.□

Proposition 3.1. :

Toutes les bandes optimales horizontales des différentes hauteurs et de longueur *L* sont générées par la résolution d'un seul problème de sac à dos.

Preuve :

Nous ordonnons les éléments de *S* par ordre croissant sur les hauteurs des pièces, tel que : $h_{k_1} \le h_{k_2} \le \cdots \le h_{kr}$ où r est le nombre des différentes hauteurs des pièces de l'ensemble S, et *(L,H)* désigne le rectangle initial.

En vertu du lemme 3.1., toutes les bandes optimales horizontales de différentes hauteurs $h_{k_i} \le H$, pour *i=1,...r* et de longueur L sont obtenues en résolvant par la programmation dynamique le seul problème de sac à dos suivant :

$$K_{LH}^{r}\begin{cases} \max F_r(L) = \max\sum_{j=1}^{n} c_j x_j \\ s.c \ \ \sum_{j=1}^{n} l_j x_j \le L \\ \quad x_j \in \mathrm{N} \end{cases}$$

Proposition 3.2.:

Toutes les bandes optimales verticales des différentes longueurs et de hauteur H sont générées par la résolution d'un seul problème de sac à dos.

Preuve :

Nous considérons maintenant tous les éléments de *S* ordonnés selon un ordre croissant sur les longueurs, tel que : $l_{k_1} \le l_{k_2} \le \cdots \le l_{k_{r'}}$ où *r'* est le nombre des différentes longueurs de l'ensemble des pièces ***S.***

En vertu du lemme 3.1, toutes les bandes optimales verticales des différentes longueurs $l_{k_i} \le L$; pour *i=1,...,r'* et de hauteur H sont générées par la résolution par la programmation dynamique du seul problème de sac à dos :

$$K_{HL}^{r'} \begin{cases} \max F_{r'}(H) = \max \sum_{j=1}^{n} c_j x_j \\ s.c \quad \sum_{j=1}^{n} h_j x_j \le H \\ \qquad x_j \in \mathrm{N} \end{cases}$$

Procédure de résolution des problèmes de sac à dos

Les problèmes de sac à dos unidimensionnels introduits dans l'algorithme sont plutôt de petites tailles, y compris pour des instances de grandes tailles, leur résolution par la programmation dynamique est particulièrement rapide, nous utilisons à cet effet l'équation suivante :

$$\begin{aligned} F_k(\alpha) = & \max_{x_k = 0,1,\cdots,\left\lfloor \frac{\alpha}{t_k} \right\rfloor} (c_k x_k + F_{k-1}(\alpha - x_k t_k) \\ & pour k = 1,\cdots,r \quad \text{et } \alpha = 0,\cdots,T \end{aligned} \qquad (1)$$

- Pour les bandes optimales horizontales, on pose $T = L$, $t_k = l_k$ et les différentes hauteurs des pièces de S sont ordonnées selon l'ordre : $h_{k_1} \le h_{k_2} \le \cdots \le h_{k_r}$
- Pour les bandes optimales verticales, on pose $T = H, t_k = h$ et les différentes pièces de l'ensemble S sont considérées selon l'ordre : $l_{k_1} \le l_{k_2} \le \cdots \le l_{k_{r'}}$.

Une version améliorée de l'équation (1) est donnée par l'expression de l'équation suivante :

$$F_k(\alpha) = \max_{x_k = 0,1,\cdots,\left\lfloor \alpha / t'_k \right\rfloor} \left(c_k x_k + \left\lfloor \frac{t'_k}{t'_{k-1}} \right\rfloor F_{k-1}(\alpha - x_k t'_k) \right) \quad (2)$$

$$pour \quad k = 1,\cdots,r \quad \text{et } \alpha = 0,\cdots,T$$

où: $F_0(\alpha) = 0,$ pour $\alpha = 1,\cdots,T, t'_0 = t'_1$.

Exemple 3.2. :

Considérons l'instance de petites taille suivante pour le problème TDGC suivante :

R *=(L,H) =(7,5),* ***S****={(3,1),(2,2)}.* Supposons qu'on s'intéresse uniquement aux bandes horizontales, l'utilisation de l'équation (1) permet de construire deux bandes représentées par la figure 3.3. (a) de valeurs $F_1(7) = 6$ et $F_2(7) = 12$ respectivement. Par l'utilisation de l'équation (2) on obtient là aussi deux bandes S'_1 et S'_2 décrites dans la figure 3.3. (b) de valeurs $F_1(7) = 6$ et $F_2(7) = 14$ respectivement

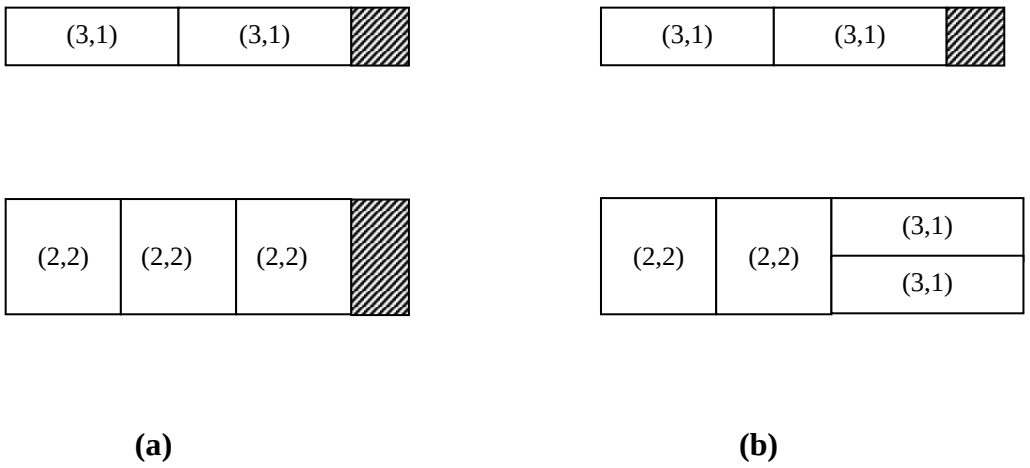


Figure 3 .3. Bandes optimales obtenues par les équations (1) et (2) sur un exemple.

Sélection des bandes :

La sélection des meilleures bandes conduit à la construction de modèles de découpe réalisables. Les deux résultats suivants nous montrent comment effectuer le choix du meilleur modèle de découpe horizontal (respectivement vertical), généré à partir de combinaisons linéaires des éléments de l'ensemble des bandes horizontales (respectivement verticales).

Proposition 3.3. :

Un modèle de découpe horizontal pour le rectangle initial R=*(L,H)* est obtenu par la résolution du problème de sac à dos suivant :

$$K_H^{hor} \begin{cases} B_{hor} = \max \sum_{i=1}^{r} F_i(L) y_i \\ s.c \quad \sum_{i=1}^{r} h_i y_i \le H \\ \quad y_i \in \mathrm{N} \end{cases}$$

où : r est la cardinalité de l'ensemble des bandes horizontales de longueur L, y_k est le nombre d'apparition de la $i^{ème}$ bande horizontale de hauteur h_{k_i} et de profit $F_i(L)$.

Preuve :

Un modèle de découpe horizontal réalisable pour le support initial R=*(L,H)*, est une combinaison de bandes horizontales de longueur L de différentes hauteurs.

Toutes les bandes optimales horizontales de longueur L et de différentes hauteurs $h_{k_i} \le H$ avec leur valeur respective $F_i(L)$, pour *i=1,...r* sont obtenues en résolvant par la programmation dynamique le seul problème de sac à dos K_{LH}^{r} (proposition 3.1.).

Par conséquent, une combinaison de ces bandes optimales horizontales disponibles, peut être formulée par le problème de sac à dos K_H^{hor} .

Proposition 3.4. :

Un plan de découpe vertical pour le rectangle initial R=*(L,H)* est obtenu par la résolution du problème de sac à dos suivant :

$$K_L^{ver} \begin{cases} B_{ver} = \max \sum_{i=1}^{r'} F_i(H) y_i \\ s.c \ \sum_{i=1}^{r'} l_i y_i \le L \\ y_i \in \mathrm{N} \end{cases}$$

où : *r'* est la cardinalité de l'ensemble des bandes verticales de hauteur *H*, y_i est le nombre d'apparition de la $i^{ème}$ bande verticale de longueur l_{k_i} et de profit $F_i(H)$.

Preuve :

Un modèle de découpe vertical réalisable pour le support initial R=*(L,H)*, est une combinaison, de bandes verticales de hauteur H de différentes longueurs.

Toutes les bandes optimales verticales de hauteur H et de différentes longueurs $l_{k_i} \le L$; avec leur valeur respective $F_i(H)$, pour *i=1,...,r'* sont obtenues en résolvant par la programmation dynamique le seul problème de sac à dos $K_{HL}^{r'}$ (proposition 3.2.).

Par conséquent, une combinaison de ces bandes optimales verticales disponibles, peut être formulée par le problème de sac à dos K_L^{ver}.

- **<u>Borne inférieure initiale :</u>**

Nous utilisons comme borne inférieure notée $B_{\inf}(\mathsf{R})$ sur la racine de l'arborescence correspondant au rectangle initial *R=(L,H)*, la meilleure valeur donnée par les deux modèles de découpe :

$$B_{\inf}(\mathsf{R}) = \max\{B_{hor}, B_{ver}\}$$

- **<u>Bornes inférieures intermédiaires</u>**

Les sommets intérieurs de l'arborescence de recherche sont associés aux rectangles guillotines (définition 3.4.1.). Chaque fois qu'un rectangle guillotine est construit, on lui associe deux solutions faisables obtenues par la procédure suivante :

Pour tout rectangle guillotine $R = (l_R, h_R)$ on peut lui associer deux solutions faisables possibles en subdivisant la région *F* en deux rectangles : un plus petit noté F_1 voir figure 3.4.a. (ou F_2 sur la figure 3.4.b) et un plus grand F_2 figure 3.4.a. (ou F_1 sur la figure 3.4.b). Pour ces deux rectangles, on effectue les opérations suivantes :

1. On détermine le modèle de découpe associé au plus grand rectangle, cette valeur est déjà disponible à partir des procédures de résolution utilisant la programmation dynamique des problèmes de sac à dos K_L^{hor}, K_L^{ver} . La valeur de la solution obtenue est notée L_1.
2. On complète la solution faisable avec la meilleure solution homogène sur le petit rectangle, on note la valeur de la solution par L_2

Finalement, la valeur du meilleur modèle de découpe faisable associé au sous-rectangle construit *R* est donnée par :

$$Lower(R)=g(R) + L_1 + L_2$$

(a) **(b)**

Figure 3.4. Représentation de deux modèles de découpe faisables (a) et (b) pour l'évaluation d'une borne inférieure sur le rectangle *R*.

Le résultat suivant, résume de manière explicite les procédures utilisées pour l'évaluation des bornes inférieures sur les sommets intérieurs. Nous soulignerons à cet effet, que les valeurs nécessaires pour l'évaluation des rectangles les plus grand (de la région complémentaire *F*) pour les deux solutions alternatives envisagées, sont déjà disponibles à partir de la procédure de résolution par la programmation dynamique des sac à dos unidimensionnels K_L^{hor} et K_H^{ver} utilisés pour l'évaluation de la borne inférieure initiale.

Théorème 3.3. :

Etant donné un rectangle guillotine *R* de dimension (l_R, h_R), F_1 et F_2 deux rectangles de dimensions $(L, H - h_R)$ et $(L - l_R, H)$, respectivement. Une borne inférieure pour *f(R)* est :

$$lower(R) = \begin{cases} g(R) + B_L^{hor}(H - h_R) + H^*(F_2) & si\ S(F_1) \geq S(F_2) \\ g(R) + B_H^{ver}(L - l_R) + H^*(F_1) & si\ S(F_1) < S(F_2) \end{cases}$$

où : $B_L^{hor}(H - h_R)$ est la valeur de la solution du sac à dos $K_L^{hor}(H - h_R)$, $B_H^{ver}(L - l_R)$ est la valeur de la solution du sac à dos $K_H^{ver}(L - l_R)$, $H^*(F_1)$ et $H^*(F_2)$ étant la valeur du meilleur modèle de découpe homogène pour F_1 respectivement F_2.

Preuve :

La procédure d'évaluation de la borne inférieure pour chaque sommet intérieur de l'arborescence, associé à un rectangle guillotine *R* de dimension (l_R, h_R), considère les deux rectangles F_1 et F_2 ,de dimensions $(L, H - h_R)$ et $(L - l_R, H)$. dans les deux cas de figures, en complétant la solution partielle *g(R)* par :

- Cas 1 :

$$B_L^{hor}(H - h_R) + H^*(F_2) \quad si\ S(F_1) \geq S(F_2)$$

Avec ; $L_1 = B_L^{hor}(H - h_R)$ la valeur de la solution du sac à dos $K_L^{hor}(H - h_R)$,disponible à partir de la solution du sac à dos K_L^{hor} (proposition 3.3.) et $L_2 = H^*(F_2)$ la valeur du meilleur modèle de découpe homogène pour F_2.

- Cas 2 :

$$B_H^{ver}(L - l_R) + H^*(F_1) \qquad si\ S(F_1) < S(F_2)$$

Avec ; $L_1 = B_H^{ver}(L - l_R)$ la valeur de la solution du sac à dos $K_H^{ver}(L - l_R)$, disponible à partir de la solution du sac à dos K_H^{ver} (proposition 3.4.) et $L_2 = H^*(F_1)$ la valeur du meilleur modèle de découpe homogène pour F_1.

Dans les deux cas, nous obtenons une borne inférieure pour le modèle de découpe optimale contenant le rectangle *R*, car un modèle de découpe générale est meilleur qu'un modèle de découpe homogène.

Dans l'encadré 2, nous donnons une description détaillée des étapes principales de l'algorithme exact qu'on note par: ***BFS (Best First Search).***

Encadré 2. Best First Search (BFS) pour le problème TDGC.

Input: une instance du problème TDGC.

Output: la valeur de la solution optimale notée $B_{inf}(R)$;

Initialisation:

{soit p_k une composante de la liste *Ouvert* qui contient les dimensions et les valeurs g, L_1, L_2, h' et f' de la k^{ème} pièce, pour $k=1,...,n$}

Soient *Ouvert* $\leftarrow$ $\{p_1,p_2,...p_n\}$; *Fermé* $\leftarrow$ ϕ; *fin = faux ;*

Calculer la borne inférieure initiale de R notée $B_{inf}(R)$;

Calculer la borne supérieure initiale de R notée $B_{sup}(R)$

Etape principale:

si ($B_{sup}(R) - B_{inf}(R) = 0$) alors sortir avec $B_{inf}(R)$ sinon

répéter

Choisir un rectangle $P \setminus écart = \max_{P \in Ouvert} \{f'(P) - B_{inf}(R)\}$;

Si ($écart \leq 0$) alors *fin: = vrai*

sinon

Transférer *P* de la liste *Ouvert* vers la liste *Fermé* ; construire le rectangle guillotine *G* tel que :

1. Le rectangle *G* est construit par le placement vertical ou horizontal de *P* avec des éléments de la liste *Fermé* dont les valeurs sont supérieures ou égales à $B_{inf}(R)$;
2. Les dimensions de chaque élément de *G* sont plus petites que celle de R;

Poser *Ouvert = Ouvert U G* avec les valeurs appropriées g, L_1, L_2, h' et f' (chaque élément de *G* vérifie $f'(P) > B_{inf}(R)$);

Effectuer la mise à jour de $B_{inf}(R)$.

jusqu'à (*fin*) or (*Ouvert* = ϕ);

Fin:

Sortir avec la solution optimale.

3.5. Développement d'une approche heuristique

Nous allons montrer maintenant, comment utiliser l'algorithme *(BFS)* décrit précédemment pour la résolution approchée des instances de très grandes taille du problème **TDGC** . Lorsque la cardinalité des ensembles D_L and D_H (désignant respectivement l'ensembles des points de découpes horizontales respectivement verticales) est assez grande, la procédure (modifiée [8]) récursive de Gilmore and Gomory [32] sur le plan numérique devient inapplicable, la complexité de cette procédure est de l'ordre $O(|D_L||D_H|\times|D_L|+|D_H|)$ et nécessite une capacité de stockage en espace mémoire équivalent à $O(|D_L||D_H|)$.

Nous présentons une approche hybride notée *(HBFS)* qui est une combinaison mixte, entre d'une part les techniques d'exploration arborescente profondeur/meilleur d'abord *(depth/ best-first search (DBFS))* et d'autre part les stratégies du Hill Climbing (HC stratégies). Les HC stratégies est une heuristique simple basée sur une recherche locale. Elle consiste à choisir sur la base de critères stratégiques un meilleur chemin local et abandonne les autres chemins définitivement (voir [55] pour plus de détails).

3.5.1. Stratégies de branchement

Nous allons décrire deux stratégies (**H.1**) et (**H.2**) qui représentent les stratégies HC . Par la suite, nous rajouterons une troisième stratégie (**H.3**) qui sera considérée comme la stratégie globale de contrôle des branchements, afin de renforcer les stratégies de HC.

(**H.1**) A chaque sommet de l'arborescence, une borne supérieure de la solution optimale pour la région *F* peut être déterminée, par la résolution du problème de sac à dos relaxé *K(F)*, nous éliminerons dans ce cas tous les rectangles guillotines construits $P \in G$ tel que

$$\frac{B_{inf}(\mathsf{R}) - g(P)}{h'(P)} \geq \sigma$$

où: $\sigma \leq 1$ est pourcentage prédéterminé. $B_{inf}(\mathsf{R})$ est la valeur de la meilleure solution faisable en cours, *g(R)* est la fonction d'utilité sur R et *h'(R)* est la borne supérieure sur la région F.

(**H.2**) Comme nous l'avons déjà mentionné, les modèles de découpe normalisés ont été utilisés afin de réduire le nombre de points de découpe, représentant les combinaisons linéaires des longueurs et hauteurs, des pièces à découper sur le rectangle initial. Dans [8,53], les auteurs ont utilisé des heuristiques basées sur la fonction de Christofides et Whitlock [11] ,pour générer le plus petit nombre possible de points de découpe. Notre approche utilise une version modifiée de cette même fonction, toujours pour le même objectif ; la construction des plus petits (les plus fructueux évidemment) ensembles de points de découpes horizontales et verticales. L'approche se résume comme suit:

Soit δ_i^l *(resp.*δ_i^h *)* pour *i= 1,...,n,* des entiers non négatifs, nous proposons une version modifiée de la fonction de Christofides et Whitlock pour limiter l'ensembles des points de découpe verticales, cette fonction est définie par l'expression suivante :

$$f_i(x) = \begin{cases} 0 & \text{si } i = 0 \text{ et } x = 0 \\ \infty & \text{si } i = 0 \text{ et } x \neq 0 \\ f_{i-1}(x) & \text{si } i \neq 0 \text{ et } ((x<l_i) \text{ ou } (x> \delta_i^l l_i)) \\ \min\{ f_{i-1}(x), \max\{h_i, \min_{1\le k\le \min\left\{\left\lfloor \frac{x}{l_i}\right\rfloor,\delta_i^l\right\}} f_{i-1}(x-kl_i)\}\} & \text{sinon} \end{cases}$$

$$\text{où : } 1 \le \delta_i^l \le \left\lfloor x/l_i \right\rfloor, \text{ pour } i=1,...,n.$$

Pour tout sous-rectangle (α,β), nous pouvons affirmer que α représente une combinaison d'un certain nombre de longueurs de pièces de $\boldsymbol{S}$ si $f_n(\alpha) \le \beta$. Pour un point de découpe vertical, (nous utilisons la même fonction pour la construction de l'ensemble des points de découpes horizontales en remplaçant : δ_i^l par δ_i^h ,l_i par $h_,$, h_i par l_i, x par y et L par H où :

$$1 \le \delta_i^h \le \left\lfloor y/h_i \right\rfloor \text{ pour } i=1,...,n.$$

A noter que, si $\delta_i^l = \left\lfloor L/l_i \right\rfloor$ (resp. $\delta_i^h = \left\lfloor H/h_i \right\rfloor$), la fonction récursive précédente est exactement celle de Chtistofides et Whitlock.

Dans l'heuristique *HBFS* , nous limitons de ce fait le nombre de sommets crées par l'arborescence de recherche aux seuls points caractérisés comme étant des points de découpes, identifiés par la fonction récursive. Nous éliminons (stérilisons) tout sommet crée, dont la longueur ou la hauteur n'est pas incluse dans D_L ou D_H respectivement. Par exemple, soient deux sommets correspondants à deux sous-rectangles existants (α_1,β_1) et (α_2,β_2), le sommet correspondant au sous-rectangle obtenu par construction horizontale (resp. verticale) est stérilisé si $\alpha_1+\alpha_2 \notin D_L$ (resp. si $\beta_1+\beta_2 \notin D_H$).

(**H.3**) Lors de l'utilisation des critères (**H.1**) et (**H.2**), nous remarquons que le fait de réduire au maximum le nombre de points de découpes risque de nous faire perdre la solution optimale du problème. Pour parer à cette éventualité, nous avons introduit une troisième stratégie sous forme de double recherche profondeur/meilleur d'abord *(DBFS)*, elle consiste à rechercher (**i**) le meilleur sommet actuel et aussi (**ii**) un autre chemin qui réalise la meilleur solution faisable pour le problème en cours.

Dans l'algorithme *BFS* , le choix du rectangle candidat *P* s'effectue sur la base du critère :

$$écart = \max_{P \in Ouvert} \{ f'(P) - B_{inf}(R) \}$$

Nous rajoutons un critère secondaire de choix du rectangle *P'* qui réalise :

$$écart' = \max_{P' \in Ouvert} \{ g(P') \}$$

Nous effectuons par conséquent, des constructions horizontales et verticales en combinant des éléments de la liste *Ouvert* non seulement avec *P* mais aussi combinés avec *P'*.

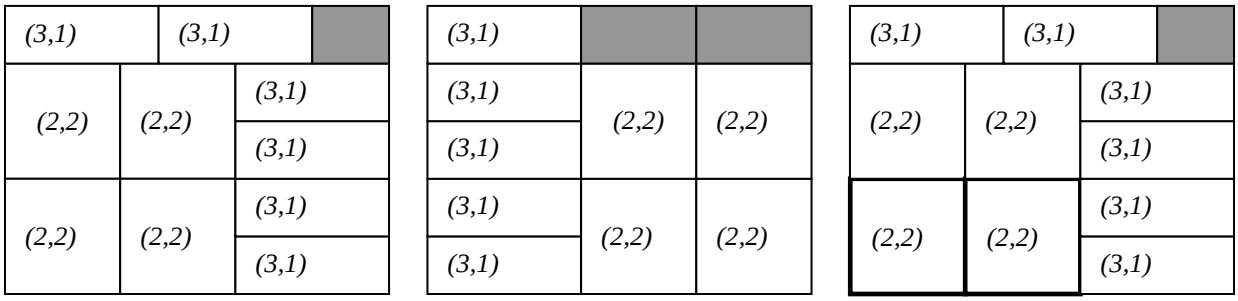
3.5.2. Déroulement de l'algorithme *HBFS*:

Considérons l'instance du problème de découpe sans contraintes, définie par l'exemple 3.2. Le modèle de découpe horizontal figure 3.5.a) et le modèle de découpe vertical figure 3.5.b) sont obtenues par l'étape initiale de l'approche *HBFS* . Le tableau suivant, résume les résultats de l'exécution de l'approche, on note par, $h(R_i, R_j)$ [respectivement $v(R_i, R_j)$] la construction horizontale (respectivement verticale) entre R_i et R_j, *(l , h)* les dimensions du rectangle guillotine construit. Le seuil de la stratégie **H.1** est fixé à σ =0.95. La borne inférieure initiale est égale à 34, elle représente la valeur de la solution optimale du problème. Pour les rectangles guillotines générés, la borne inférieure varie entre 29 et 34, ce qui représente un rapport d'approximation entre 0.853 et 1. La figure 3.5.c) représente le rectangle guillotine R_5 avec les solutions obtenues pour les deux régions F_1 et

F_2. Nous remarquons que pour le rectangle F_1, le modèle de découpe associé, représente une partie du modèle de découpe horizontale R_1 figure 3.5.a). Pour le rectangle F_2, la solution est donnée par le meilleur modèle de découpe homogène, composée de pièces p_1.

R	Construction	*(l,h)*	*g(R)*	*h'(R)*	L_1	L_2	*Lower(R)*	B_{inf} (R)	σ
R_1	p_1	*(3,1)*	3	32	28	3	34	34	0.969
R_2	p_2	*(2,2)*	*4*	*31*	*23*	*4*	*31*	*34*	*0.968*
R_3	$h(R_1, R_1)$	*(6,1)*	6	29	28	0	*34*	34	*0.965**
R_4	$v(, R_1\ R_1)$	*(3,2)*	*6*	29	*20*	*8*	*34*	34	0.965*
R_5	$h(R_2\ R_2,)$	*(4,2)*	*8*	*27*	*20*	*6*	*34*	34	0.963*
R_6	$v(R_2, R_2)$	*(2,4)*	*8*	*27*	*23*	*0*	*31*	34	0.963*
R_7	$h(R_1, R_2)$	*(5,2)*	*7*	*25*	*20*	*4*	*31*	34	1.080*
R_8	$v(R_1, R_2)$	*(3,3)*	*7*	*26*	*16*	*6*	*29*	34	1.038*

*=sommet stérilisé



(a) (b) (c)

(a) Modèle de découpe horizontal initial (solution optimale)
(b) Modèle de découpe vertical initial
(c) Modèle de découpe contenant $R_5 = h(R_2, R_2)$

3.6. Résultats numériques

Dans cette section nous présentons les performances de l'algorithme *BFS* (applicable aux versions pondérée et non pondérée du problème *TDGC*). Les tests empiriques concernant la performance de l'algorithme exact sont les mêmes effectués pour mesurer les performances de l'heuristique *HBFS* , comparée d'une part à l'heuristique de Morabito et al (*MMA*)[53] valable uniquement pour la version non pondéré, et d'autre part à celle de Beasley (*BE*) [8] pour le cas pondéré.

La comparaison entre les heuristiques *MAA, BE* et *HBFS* a été réalisée en tenant compte, des propres caractéristiques de performance pour chaque heuristique. La première heuristique (*MAA*) utilise deux paramètres α et β, ainsi qu'une approche qui génère les deux ensembles des points de découpe D_L et D_H (voir [53]). La seconde (*BE*) utilise uniquement une heuristique qui génère deux ensembles contenant un certain nombre de combinaisons linéaires de longueurs et de hauteurs de pièces (voir [8]). D'un autre côté notre algorithme utilise la version modifiée de la fonction de Christofides et Withlock pour générer les ensembles de points de découpes avec une valeur de σ fixé à 0.95. Pour l'ensembles des instances considérées:

- Le *MAA* est exécuté avec $\alpha = 0.99$, $\beta = 1$, $158 \leq |D_L| \leq 265$, $158 \leq |D_H| \leq 265$ et un pas descente : depth = 3.
- Le *BE* est exécuté avec $185 \leq |D_L| \leq 265$ et $158 \leq |D_H| \leq 265$.
- Le *HBFS* est exécuté avec $(1,1) \leq (\delta_i^t, \delta_i^w) \leq (2,2)$, pour $i = 1,...,n$ *i.e.*, $95 \leq |D_L| \leq 121$ et $92 \leq |D_H| \leq 139$.

Nous considérons quatre groupes d'instances générées aléatoirement. Les deux premiers groupes comportent 100 instances, avec L et H pris dans l'intervalle [250,500] et le nombre de pièces à découper choisi dans l'intervalle [20,50]. Les deux seconds groupes (groupe 3 et groupe 4) contiennent aussi 100 instances de grandes taille. Les valeurs de L et H sont rangés dans l'intervalle [500,1000], et le nombre de pièces à découper dans l'intervalle [50,150]. Les dimensions longueur respectivement hauteur des pièces rectangulaires à découper sont prises uniformément dans $]0,L[$ et $]0,H[$ respectivement.

Pour les instances groupe 1 et groupe 3, nous considérons le profit de chaque type de pièce i comme étant exactement sa surface, et pour les deux autres groupes, le profit affecté à chaque type de pièce est indépendant de sa surface $(c_i \neq l_i h_i)$ tel que $c_i = \lceil \gamma \pi_i \rceil$, où γ est un nombre uniformément distribué dans l'intervalle réel [0.4 , 2.5] et $\pi_i = l_i h_i$, est la surface de la pièce i, pour ; $i = 1,..., n$.

Groupes	1	3	Moyenne
Temps Moy (sec) *MAA*	2.41	7.29	4.85
Rap d'appr Moy (*MAA*)	0.885	0.897	0.891
(%) Solutions optimales	59	51	55
Temps Moy (sec) *HBFS*	2.07	5.35	3.71
Rap d'appr Moy (*HBFS*)	0.995	0.999	0.997
(%)Solutions optimales	85	91	88

Table 1. Performance de *HBFS* versus *MAA*.

Sur la table 1, nous pouvons constater la qualité des solutions partielles, aussi bien que le temps d'exécution de l'algorithme *HBFS*. Pour les instances traitées (groupe 1et groupe 3), des solutions d'excellente qualité sont obtenues avec un rapport d'approximation moyen ($\frac{A(I)}{Opt(I)}$) où : *A(I)* et *Opt(I)* représentent la solution partielle et la solution optimale respectivement de l'instance I) égale à 0.997 , en un temps moyen représentant 85.89% ($\frac{100 \times Temps(HBFS)}{Temps(MAA)}$) de celui requis par l'algorithme *MAA*.

De plus, 88% des solutions réalisées par *HBFS* sont des solutions optimales, contrairement à l'algorithme *MAA* qui réalise 55% des solutions optimales. Pour le groupe 1, les instances de moyenne taille, l'algorithme *MAA* produit 59% des solutions optimales en un temps moyen égale à 2.41 sec. D'autre part, pour le même groupe d'instance, l'algorithme *HBFS* produit 85% en un temps moyen égale à 2.07 sec. Dans le groupe 3, instances de grande taille, la qualité des solutions obtenues est meilleure : le rapport d'approximation moyen est égale à 0.999 (l'algorithme *MAA* réalise un rapport d'approximation moyen égale à 0.897), en un temps d'exécution moyen représentant

73.39% de celui de *MAA*. Le pourcentage de solutions optimales réalisées est en progression, il passe à 91%.

Groupes	2	4	Moyenne
Temps Moy (sec) *BE*	7.84	15.39	11.61
Rap d'appr Moy. (*BE*)	0.831	0.801	0.816
(%) Solutions optimales	37	38	37.5
Temps Moy (sec) *HBFS*	2.43	6.07	4.25
Rap d'appr Moy (*HBFS*)	0.989	0.995	0.992
(%)Solutions optimales	80	92	86

Table 2. performance de *HBFS* versus *BE* .

Sur la table 2, nous considérons maintenant le cas pondéré, en comparant *HBFS* à l'heuristique *BE*. Nous enregistrons les mêmes constatations observées sur la table 1. Précisément, 86% des solutions obtenus par l'algorithme *HBFS* sont optimales. Le gain moyen sur le temps de calcul pour l'ensemble des instances traitées, par rapport à celui de l'algorithme *BE*, est de 63.39%. A noter là aussi, pour les instances de grande taille les performances significatives enregistrées dans les qualités des solutions obtenues par l'algorithme *HBFS.*

Instance	(127,98)	(253,294)	(3000,3000)	(40,70)	(40,70)
Valeur de la solution optimale	12348	73176	8997780	3076	2240
Val. Sol. Par *MAA* || *BE*	12348	72172	8944026	3076	2240
Temps Moy (sec) de *MAA* || *BE*	0.22	0.33	32.57	0.51	0.63
. Val. Sol de *HBFS*	12348	73176	8997780	3076	2240
Temps Moy (sec) de *HBFS*	0.09	0.16	10.17	0.08	0.07

Table 3. Performance de *HBFS* versus *MAA* et *BE* .

Sur la tables 3, nous considérons quelques instances prises dans la littérature sur lesquelles, nous avons comparé l'algorithme *HBFS* aux algorithmes *MAA* et *BE* en terme de temps de calcul et de valeur de solution réalisée. La première instance est tirée de [40]. La seconde est prise de [53]. La troisième instance de [8]. Les deux dernières sont prises de [11]. Pour toutes ces instances l'algorithme *HBFS* produit la solution optimale en consacrant moins de temps que les deux autres algorithmes. Nous pouvons constater pour la troisième instance R *=(L,H)=(3000,3000),* avec *32 pièces à découper* que :

- la valeur de la meilleure solution publiée par Beasley est 8868950 en plus de *3 minutes* de temps de calcul.
- L'algorithme de Morabito et al donne une solution de valeur 8944026 en *33 sec*.
- Notre algorithme produit une solution optimale de valeur égale à 8997780 en *10 sec*.

3.7. Conclusion

Nous avons présenté une nouvelle approche, ainsi qu'une heuristique efficace pour la résolution du problème de découpe (non) pondéré non contraint à deux dimensions. Notre algorithme repose, sur l'utilisation des méthodes de programmation dynamique pour la résolution de problème de sac à dos unidimensionnels, dans l'élaboration des critères d'optimalité de l'arborescence de recherche. L'heuristique développée est basée sur l'utilisation de stratégies de branchement, combinant la recherche profondeur/meilleur d'abord *(BFS)* et les stratégies du Hill Climbing (HC). Un avantage important de l'algorithme réside dans son application aux cas pondéré et non pondéré des problèmes *TDGC*, ainsi qu'à la version avec contrainte de niveau. Les expériences numériques réalisées justifie l'efficacité des approches proposées, comparées aux meilleurs algorithmes rencontrés dans la littérature, en l'occurrence celui de Beasley[8] pour le cas pondéré, et celui de Morabito[53] et al pour le cas non pondéré.

Finalement, nous pensons que notre approche peut être généralisée pour la résolution du problème de découpe contraint à deux dimensions. Une implémentation parallèle de cet algorithme constitue un travail intéressent pour des recherches futures.

Chapitre 4

Approche séquentielle exacte pour la résolution du problème de découpe généralisé sans contraintes

4.1. Introduction

Le problème de découpe avec un seul support en stock, est en réalité un cas particulier du problème de découpe généralisé sans contraintes à deux dimensions *(GTDGC).*On suppose que l'on dispose de *n* pièces à découper et de *m* supports rectangulaires en stock. L'objectif est de découper l'ensemble des pièces sur l'ensemble des plaques de façon à maximiser la « rentabilité » de chaque support.

Dans ce chapitre, nous proposons un algorithme exact pour la résolution du problème *GTDGC.* Le principe de la méthode développée repose sur l'application de la programmation dynamique et l'introduction d'une nouvelle stratégie basée sur le principe de saut de points, qui permet de réduire l'espace de recherche. Dans la section 4.4 nous développons une méthode approchée pour le problème *GTDGC* basée sur le principe du plus grand rectangle fictif, au départ utilisons une réduction du problème initial en une série de problèmes de sac à dos unidimensionnels. Ensuite nous effectuons un tri par ordre croissant sur les longueurs des supports rectangulaires. Finalement nous appliquons une procédure basée sur la programmation dynamique pour générer un ensemble de bandes horizontales. Cet ensemble est généré progressivement selon l'ordre initialement appliqué sur les longueurs des supports. Par la suite, nous utilisons le même principe pour la génération progressive des bandes verticales. Finalement, la solution approchée de chaque support sera représenté par la combinaison de quelques bandes horizontales (ou verticales) disponibles.

Dans la section 4.4.1 nous démontrons que lorsque les sac à dos unidimensionnels utilisés sont résolus par l'utilisation d'une procédure gloutonne, l'algorithme garantissait un rapport d'approximation polynomial constant. Pour ce résultat, nous avons utilisé le résultat donné par Fayard et Zissimopoulos [23] qui considère le cas non pondéré du problème de découpe sans contraintes.

A la fin du chapitre (section 4.6.), nous présentons une étude expérimentale sur des groupes d'instances générées aléatoirement. Nous examinons les performances de l'algorithme exact et de l'heuristique sur ces instances, en les comparant au meilleur algorithme existant dû à Beasley[8].

4.2. Présentation du problème

Une instance du problème de découpe généralisé sans contraintes noté GTDGC est représenté par *m* supports rectangulaires R_j de dimensions (L_j,H_j), $j = 1,...,m$, un ensemble de *n* pièces rectangulaires $S=\{(l_1,h_1),...,(l_n,h_n)\}$ et un vecteur de profit $c = (c_1, \cdots, c_n)$ avec c_i le profit de la $i^{ème}$ pièce. L'objectif étant de découper l'ensemble des pièces sur les supports en stock de façon à maximiser la rentabilité de chaque support représentée par :

$$F_j(x_j) = \sum_{i=1}^{n} c_i x_{(j,i)}$$

où $x_{(j,i)} \in \angle$ désigne le nombre de pièces de type *i* produites sur le support *j*, pour $i = 1,...,n$ et $j = 1,...,m$.

Nous supposons que les pièces sont fixées, que la contrainte de découpe est du type guillotine, et le nombre de découpe guillotine est illimité (cas sans contraintes de niveaux).

4.3. Algorithme séquentiel exact pour le problème GTDGC :

Dans notre approche de résolution du problème *GTDGC*, il nous a semblé intéressent de concevoir un algorithme parallèle ou du moins un algorithme séquentiel efficace, capable de résoudre de manière exacte le problème considéré. Dans cette section nous développons une méthode exacte [41] basée d'une part sur les techniques de la programmation dynamique et d'autre part, sur l'introduction d'une nouvelle stratégie utilisant un principe de saut de points afin éliminer un certain nombre de configurations de découpe qui ne jouent pas un rôle important dans la détermination de la solution optimale, réduisant par l'occasion l'espace de recherche.

4.3.1 Principe de l'approche

Comme il n'existe pas d'approche spécifique à ce problème dans la littérature, nous avons tenté de proposer trois principes différents.

Le *premier principe* consiste à concevoir un algorithme parallèle en réduisant le problème de découpe sous forme d'une série de problèmes de découpe sans contraintes. Le principe de l'algorithme parallèle [10] peut être résumé par les étapes suivantes:

1. Pour chaque processeur *j*, pour *j = 1,...,m* lui affecter un support rectangulaire (L_j,H_j).
2. Résoudre chaque problème de découpe sans contraintes en utilisant une approche exacte *BFS* (chapitre 2).

Le *deuxième principe* consiste à construire un rectangle fictif $(L_{max},H_{max})=(\max_{1\le j\le m} L_j, \max_{1\le j\le m} H_j)$ et d'appliquer la programmation dynamique sur ce rectangle. Dans ce cas, toutes les solutions optimales des sous rectangles (supports initiaux) sont disponibles. Ce principe a été proposé par Gilmore et Gomory [30] sans aucun test de justification de l'approche.

le *troisième principe* revient à construire un meilleur algorithme exact séquentiel en :

1. Détectant quelques supports rectangulaires qui peuvent être traités simultanément.
2. Négligeant quelques régions de l'espace de recherche qui n'affectent en aucun cas la solution optimale du problème.

Nous nous sommes basés sur le troisième principe pour concevoir un algorithme séquentiel. Par exemple (voir figure 4.1.), si on considère deux rectangles en stock de dimensions respectivement (L_1,H_1) et (L_2,H_2) alors on procède de la manière suivante :

1. Les deux supports sont ordonnés selon un ordre croissant des longueurs (figure 4.1 a), pour simplifier, nous supposons que $L_1 \le L_2$.
2. La résolution, appliquée sur les différentes régions est effectuée comme suit:

a) Nous appliquons une procédure de programmation dynamique sur le rectangle (L_1,H_2), en ne considérant que l'ensemble des points :

$$\{ 1 \le x \le L_1 \wedge 1 \le y \le H_2 \}.$$

b) Nous appliquons la même procédure sur le sous rectangle $(L_2\text{-}L_1, H_2)$, c'est à dire l'ensemble des points $\{x> L_1 \wedge y \le H_2\}$.

c) Nous supprimons le rectangle (L_2,H_2) qui est résolu à l'optimum et nous appliquons de nouveau la même procédure sur le rectangle $(L_1, H_1\text{-} H_2)$ (c'est à dire l'ensemble des points $\{ x \le L_1 \wedge y > H_2 \}$).

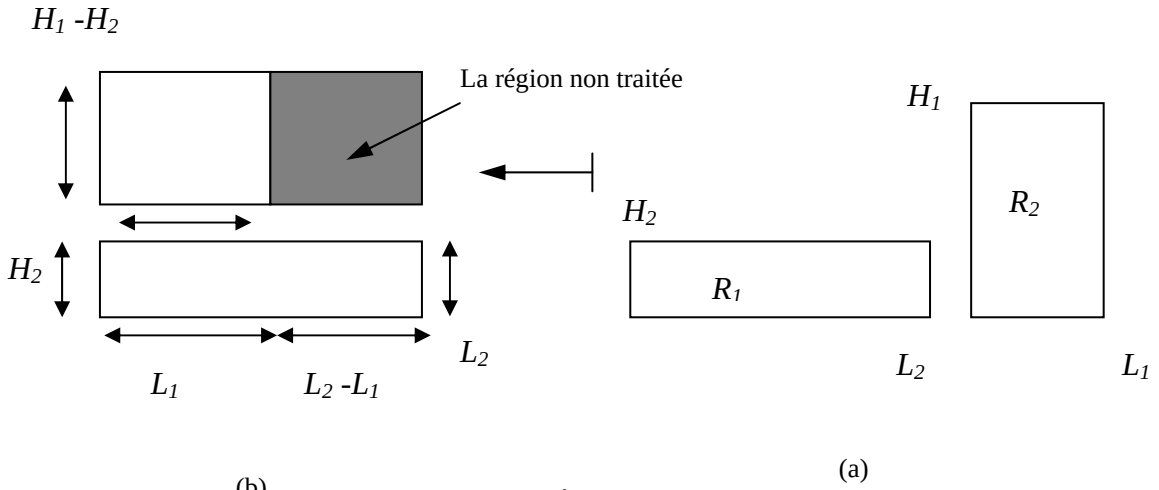


Figure 4.1

(a) Deux rectangles distincts en stock,

(b) Le résultat du traitement séquentiel des rectangles en stock .

Nous notons que cette procédure permet de construire toutes les solutions optimales des sous rectangles de dimensions inférieures. $x \leq L_1 \wedge y \leq H_2$ .Par ailleurs, l'approche ne traite en aucun cas la région $x > L_1 \wedge y > H_2$ (partie hachurée sur la figure 4.1.b) contrairement à l'approche qui s'appuie sur le deuxième principe.,

Encadré 1. Algorithme exact pour le problème GTDGC (EAG)

Entrée : Une instance du problème *GTDGC.*
Sortie : V_j^*, $1 \le j \le m$. valeur de la solution optimale du problème *GTDGC.*

Initialisation

Réordonner les longueurs des supports en stock tel que :$L_1 \le L_2 \le ... \le L_m$.
Poser $L_{max} = \max_{1 \le j \le m} L_j$ et $H_{max} = \max_{1 \le j \le m} H_j$;
Construire P et Q les deux ensembles de discrétisation des points ;
$\forall x \in P$ et $\forall y \in Q$, poser $F^*(x,y) = max\{ \{0,c_i\} \mid l_i \le x \wedge h_i \le y, \forall (l_i,h_i) \in \mathbf{S} \}$
poser $x_2 := min \{ x \mid x \in P \}$, $y_2 := min \{ y \mid y \in Q\}$, $k := m$ et $k' := 1$;
$\forall k : 1 \le k \le m$, poser $P_k := max \{ x \mid x \in P \wedge x \le L_k \}$ et $Q_k := max \{ y \mid y \in Q \wedge y \le H_k \}$;

Etapes principales

étape 1 :
Poser $x_1 := min \{ x \mid x \in P \}$;
répéter
si $F^*(x_1,y_1) + F^*(x_2,y_2) > F^*(x_1 + x_2, y_2)$ alors $F^*(x_1 + x_2, y_2) := F^*(x_1, y_2) + F^*(x_2, y_2)$;
poser $x_1 := min \{ x \in P \mid x > x_1 \}$;
jusqu'à $(x_1 > x_2)$ *ou* $(x_1 + x_2 > P_k)$;

étape 2 :
Poser $y_1 := min \{ y \mid y \in Q \}$;
si $x_2 > P_{k'}$ alors poser $k' := k' + 1$;
répéter
si $F^*(x_2,y_1) + F^*(x_2,y_2) > F^*(x_2, y_1+y_2)$ alors $F^*(x_2,y_1+ y_2) := F^*(x_2, y_1) + F^*(x_2, y_2)$;
poser $y_1 := min \{ y \in P \mid y > y_1 \}$;
jusqu'à $(y_1 > y_2)$ *ou* $(y_1 + y_2 > P_{k'})$;

étape 3 :
si $x_2 < P_k$ alors poser $x_2 := min \{ x \in P \mid x > x_2 \}$ et aller à l'étape 1;
si $y_2 < Q_k$ alors poser $y_2 := min \{ y \in Q \mid y > y_2 \}$,
$x_2 := min \{ x \mid x \in P \}$, $k' := 1$ et aller à l'étape 1;
poser $k := k-1$; si $k = 0$ alors aller à l'étape 4;
poser $x_2 := min \{ x \mid x \in P \}$, $y_2 := min \{ y \in Q \mid y > y_2 \}$, $k' := 1$ et aller à l'étape1 ;

étape 4 :
sortir avec les valeurs de la solution optimale $V_j^* := F^*(P_j, Q_j)$, pour $j = 1,...m$;
End .

P et Q représentent respectivement les ensembles des points de découpe verticale et horizontale (voir 2.4.)

4.3.2.Convergence de l'algorithme EAG

L'algorithme présenté dans l'encadré 1 donne une description détaillée des différentes étapes de l'approche séquentielle exacte utilisant le troisième principe. L'algorithme repose sur un certain nombre de résultats théorique que nous développerons plus loin.

Au préalable, nous donnons la définition d'un *support rectangulaire non-dominant.* En utilisant cette définition nous démontrons que la présence d'un tel support peut être omise sans affecter la solution optimale du problème. Par la suite nous montrons que si toutes les longueurs des supports respectent un ordre croissant, alors nécessairement toutes les hauteurs respectent l'ordre inverse. Finalement, en utilisant l'ordre sur les longueurs, nous démontrons la convergence de l'algorithme.

Définition. 4.3.1 Un support rectangulaire non dominant est un support rectangulaire dont les dimensions $(L_k,H_k)_{1\leq k\leq m}$ vérifient l'inégalité suivante:

$$\exists k' \neq k \wedge 1 \leq k' \leq m \setminus L_k \leq L_{k'} \wedge H_k \leq H_{k'}$$

Lemme 4.3.1 Soit k un indice d'un support rectangulaire. S'il n'existe pas de supports rectangulaires non dominants dans la liste des supports en stock, alors

$$\forall k \setminus 1 \leq k \leq m\text{-}1 : L_k \leq L_{k+1} \wedge H_k \geq H_{k+1}$$

Preuve. Supposons que l'ordre suivant sur les longueurs est vérifié $L_1 \leq L_2 \leq ... \leq L_{m'}$, où initialement $m' = m$. on procède de la manière suivante afin d'éliminer les supports rectangulaires non dominants. En clair, pour $k = 1$, on a $\forall k' \leq m' \wedge k' \neq k : L_{k=1} \leq L_{k'}$. ainsi on distingue deux cas:

(i) $H_k \leq H_{k'}$ *ou (ii)* $H_k > H_{k'}$.

Cas *(i)* : on supprime seulement le support rectangulaire non dominant (L_k,H_k), puisque $(L_k,H_k) \leq (L_{k'},H_{k'})$, et on garde toujours l'ordre $L_2 \leq L_3 \leq ... \leq L_{k'} \leq ... \leq L_m$ .il suffira à ce moment de réorganiser les indices comme suit:

$$(L_{m'-1},H_{m'-1}) \longleftarrow (L_{m'},H_{m'}),...,(L_{t-1},H_{t-1}) \longleftarrow (L_t,H_t), \text{ pour } t \geq 2.$$

On note par m' le nombre de supports rectangulaires restants, donc on pourra répéter le processus sur le nouveau support rectangulaire $(L_{k=1},H_{k=1})$ comparé aux autres.

Pour le cas (ii) : On passe à l'indice suivant de *k'*, et on répète le processus jusqu'à la dernière valeur *m'* qui désigne le nombre de supports rectangulaires restants.

En clair, cette procédure itérative permet de réorganiser l'ensemble des supports rectangulaires en stock, tout en éliminant tous les supports non dominants. Le nouvel ensemble des supports en stock sera constitué uniquement, des supports vérifiant un ordre croissant sur les longueurs $L_1 \leq L_2 \leq ... \leq L_{m'}$, où $m' \leq m$ et un ordre décroissant sur les hauteurs $H_1 \geq H_2 \geq ... \geq H_{m'}$

Théorème. 4.3.1. L'algorithme *EAG* détermine les valeurs de la solution optimale pour tous les supports rectangulaires en stock (L_j,H_j) pour $j = 1,...,m$.

Preuve. Supposons qu'il n'existe pas de supports rectangulaires non dominants (si un tel support existait alors il suffit juste d'appliquer au préalable le lemme précèdent). Comme nous l'avons constaté, l'utilisation du lemme permet d'obtenir l'ordre suivant:

$$L_1 \leq L_2 \leq ... \leq L_m \wedge H_1 \geq H_2 \geq ... \geq H_m .$$

Maintenant, il suffit de montrer que l'algorithme *EAG* résous le problème de découpe sans contraintes *TDGC* en utilisant toutes les pièces rectangulaires sur le rectangle fictif en stock (L_m,H_1), sans considération de la région à ne pas traiter. Nous montrons une telle affirmation pour *(i)* deux rectangles en stock et finalement *(ii)* une généralisation pour $m \geq 2$.

(i) Pour $m = 2$, le problème est représenté par la configuration $L_1 \leq L_2 \wedge H_1 \geq H_2$ l'étape 1 de l'algorithme *EAG* traite simultanément les sous rectangle (L_1,H_2) (qui est une partie commune des deux rectangles en stock) sans négliger l'ensemble des points sur le sous rectangle:

$(L_1,H_2\text{-}H_1) = (x \leq L_1, y > H_2)$, i.e. $(y_2 = 1..., H_1, y_1 = 1,...,H_1 \wedge y_1 < y_2)$.

Dans l'étape 2 de l'algorithme EAG, à chaque fois que x_1 ou x_2 est plus grand que L_1, les deux variables y_1 ou y_2 prennent des valeurs égales à $1,...,H_2$. De plus, dans cette procédure l'ensemble des points de la région à ne pas traiter sont identifiés est représentée par :

$$\{ x > L_1 \wedge y > H_2 \} .$$

En utilisant le même raisonnement pour $m = 3$, on peut déduire que la première région à négliger est $\{ x > L_1 \wedge y > H_2 \}$ et la seconde est $\{ x > L_2 \wedge y > H_3 \}$ (voir figure 4.2.)

(ii) Pour le cas général *($m \geq 2$)*, le raisonnement précèdent se résume comme suit :

- la résolution de (L_m,H_m) indique l'ensemble des points des régions inutilisables

$$\{ x > L_{m-1} \wedge y > H_m \}, \{ x > L_{m-2} \wedge y > H_{m-1} \},....\{ x > L_1 \wedge y > H_2 \};$$

- pour (L_{m-1},H_{m-1}) on obtient l'ensemble des points non considérés

$$\{ x > L_{m-2} \wedge y > H_{m-1} \}, \{ x > L_{m-3} \wedge y > H_{m-2} \}, ..., \{ x > L_1 \wedge y > H_2 \};$$

- • •
- pour (L_2,H_2) on néglige les points de la région $\{ x > L_1 \wedge y > H_2 \};$
- finalement pour (L_1,H_1) l'ensemble des points inutilisés est l'ensemble vide (ϕ).

De cette manière, l'introduction des méthodes de la programmation dynamique au sein de l'algorithme EAG pour l'ensemble des supports rectangulaires en stock, fournit les valeurs de la solution optimale, tout en utilisant le principe des sauts de points sur les points des régions inutilisables.

Remarque 4.3.1. L'algorithme décrit dans l'encadré 1 considère qu'il n'existe pas de supports rectangulaires non dominants. Si de tels supports rectangulaires existaient, il suffirait d'appliquer au préalable la procédure du lemme 4.3.1.pour les éliminer.

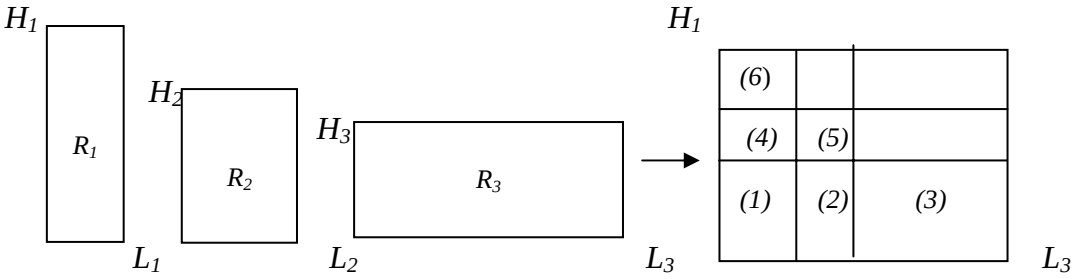


Figure 4.2.

(a) Trois rectangles distincts en stock,

(b) Traitement des parties communes avec le principe de saut de points.

Les parties non numérotées représentent l'ensemble des points non traités par l'algorithme *EAG*.

4.4. Développement d'une heuristique de recherche

L'approche de Gilmore and Gomory [30] pour le problème *GTDGC* utilise un programme linéaire pour la génération des bandes horizontales et verticales, son implémentation pratique est irréalisable, notamment pour des instances de grandes tailles. Par l'introduction à cet effet des problèmes de sac à dos unidimensionnels, nous avons développé une heuristique notée *HSG* de recherche [41] qui s'applique comme suit :

Bloc 1. Procédure de génération des bandes horizontales :

Résoudre le problème de sac à dos unidimensionnels avec le second membre:

$L_{max} = \max_{1 \le j \le m} L_j$, où L_j est la longueur du rectangle en stock *j,* Ainsi les meilleures bandes horizontales pour chaque support rectangulaire en stock sont disponibles.

Bloc 2. Procédure de génération des bandes horizontales :

Résoudre le problème de sac à dos unidimensionnels avec le second membre :

$H_{max} = \max_{1 \le j \le m} H_j$ où H_j est la hauteur de chaque rectangle en stock, les meilleures bandes verticales pour chaque rectangle en stock sont ainsi disponibles.

Bloc3. Pour chaque rectangle en stock (L_j,H_j) pour $j=1,...,m$, on combine les bandes horizontales disponibles de dimensions inférieures ou égales à (L_j,H_j) afin d'obtenir un modèle de découpe horizontal. D'un autre côté, la combinaison des différentes bandes verticales de dimensions inférieures ou égales à (L_j,H_j) fournit un modèle de découpe verticale. Finalement on choisit pour chaque support en stock (L_j,H_j) le meilleur modèle de découpe entre l'horizontale et le verticale.

Dans l'encadré 2 nous donnons une description détaillée des étapes principales de l'heuristique. A noter que notre approche est étroitement proche de celle d' Andonov et al [2] et de Chen et al [10], du point de vue implémentation parallèle.

Nous proposons une implémentation parallèle pour les procédures de résolution des deux blocs (**Bloc 1** et **Bloc 2**) et une approche séquentielle ou parallèle pour la procédure décrite dans le **Bloc3.**

Encadré 2. Heuristique de recherche pour le problème *GTDGC (HSG)*

Entrée : une instance du problème *GTDGC.*

Sortie : une solution sous optimale de valeur V_j pour *j=1,…,m*

Bloc 1.

réordonner les supports selon un ordre croissant $L_1 \leq L_2 \leq \ldots \leq L_m$;
réordonner les pièces de ***S*** *tel* que $h_1 \geq h_2 \geq \ldots \geq h_n$;
poser ***S'*** **=*S***, *k = 1* et $nb\_r_k^h = 0$;{ nombre de bandes horizontales}
répéter
choisir le sous ensemble $\Sigma \subseteq S'$ tel que les éléments de Σ rentrent dans le rectangle en stock (L_k,H_k);
si *k = 1* alors générer les r_k^h bandes horizontales sur le support (L_k,H_k) en utilisant la programmation dynamique (composée des pièces de **Σ);**
sinon
si $H_k > H_{k-1}$ alors générer les r_k^h bandes horizontales sur le support (L_k,H_k) en utilisant la programmation dynamique (composée des pièces de **Σ)** et en considérant le $(nb\_r_k^h)^{\text{ème}}$ bande;
fsi
fsi
poser *S'* := *S'* \ **Σ** , *k* := *k+1* et $nb\_r_k^h := nb\_r_k^h + r_k^h$;
jusqu'à *(k>m)* où ($S' = \phi$)

Bloc 2.

réordonner les supports selon un ordre décroissant tel que $H_1 \geq H_2 \geq \ldots \geq H_m$;
réordonner les pièces de ***S*** tel que $l_1 \geq l_2 \geq \ldots \geq l_n$;
poser ***S'*** **= *S*,** *k=1* et $nb\_r_k^v = 0$; { nombre de bandes verticales}
Répéter
choisir le sous ensemble $\Sigma \subseteq S'$ tel que les éléments de Σ rentrent dans le rectangle en stock (L_k,H_k);
si *k = 1* alors générer les r_k^v bandes verticales sur le support (L_k,H_k) en utilisant la programmation dynamique (composée des pièces de **Σ);**
Sinon
si $L_k > L_{k-1}$ alors générer les r_k^v bandes verticales sur le support (L_k,H_k) en utilisant la programmation dynamique (composée des pièces de **Σ)** en considérant le $(nb\_r_k^v)^{\text{ème}}$ bande;
fsi
fsi
Poser *S'* := *S'* \ **Σ** , k := *k+1* et $nb - r_k^v := nb - r_k^v + r_k^v$;
Jusqu'à *(k>m) ou* ($S' = \phi$);

Bloc 3.

Résoudre les *m* problèmes de sac à dos unidimensionnels, soit V_j^{hor} les valeurs des solutions obtenues représentant les modèles de découpe horizontaux de chaque support pour *j=1,...m.*
Résoudre les *m* problèmes de sac à dos unidimensionnels, soit V_j^{vert} les valeurs des solutions obtenues représentant les modèles de découpe horizontaux de chaque support pour *j=1,...m.;*
pour *j :=1* ,..., *m* faire $V_j := \max\{V_j^{hor}, V_j^{vert}\}$;

Fin

Le théorème suivant stipule que l'algorithme *HSG* pour le problème (non) pondéré *GTDGC* admet un rapport d'approximation constant, dans sa version polynomiale ou non-polynomiale. L'algorithme polynomial est basé sur une heuristique gloutonne [57] pour résoudre les problèmes de knapsack unidimensionnels. Pour la version non-polynomiale, on utilise une approche exacte pour la résolution des problèmes de knapsack unidimensionnels [58,59] en un temps pseudo-polynomial.

Théorème 4.4.1. Le problème (non) pondéré *GTDGC* admet un rapport d'approximation vérifiant:

(i) $\frac{A(I)}{Opt(I)} \geq \frac{1}{3}$ pour le cas non pondéré ;

(ii) $\frac{A(I)}{Opt(I)} \geq \frac{1}{4}$ pour le cas pondéré ;

où: *A(I)* (resp. *Opt(I))* est la valeur de la solution partielle (resp. optimale) de l'instance *I* du problème *GTDGC.*

Preuve. Il est facile de constater que pour la résolution du problème *GTDGC*, il suffit de résoudre individuellement le problème *TDGC* correspondant à chaque support rectangulaire en stock (L_k,H_k) pour $k=1,...,m$.

De plus, l'approche heuristique *HSG* réalise le meilleur modèle de découpe homogène , donc vérifie l'inégalité suivante :

$$A(I_k) \geq \left\lfloor \frac{L_k}{l_t} \right\rfloor \left\lfloor \frac{H_k}{h_t} \right\rfloor c_t$$

où : $I = (I_1,...,I_k,...,I_m)$ et $1 \leq t \leq n$.

Nous pouvons distinguer à cet effet deux cas: (i) le cas pondéré et (ii)le cas non pondéré.

Pour les deux cas, on pose :

$$\delta = \left\lfloor \frac{L_k}{l_t} \right\rfloor \text{ et } \quad \delta' = \left\lfloor \frac{H_k}{h_t} \right\rfloor \quad .$$

(i) Chaque poids est représenté par $c_i = l_i h_i$, pour $i=1,...n$, donc la valeur de la solution optimale vérifie :

$$Opt\ (I_k) \leq L_k H_k.$$

Ce qui nous permet d'avoir :

$$\frac{Opt(I_k)}{\mathrm{A}(I_k)} \le \frac{L_k H_k}{\delta\delta' l_t h_t}.$$

D'autre part, on a

$$\left\lfloor \frac{L_k}{l_t} \right\rfloor \left\lfloor \frac{H_k}{h_t} \right\rfloor \le \frac{L_k}{l_t}\frac{H_k}{h_t} \le \left(\left\lfloor \frac{L_k}{l_t} \right\rfloor + 1\right)\left(\left\lfloor \frac{H_k}{h_t} \right\rfloor + 1\right)$$

$$\Leftrightarrow \delta\delta' \le (\delta+1)(\delta'+1) \Rightarrow \frac{Opt(I_k)}{\mathrm{A}(I_k)} \le \frac{(\delta+1)(\delta'+1)}{\delta\delta'}$$

par conséquent ;

$$\frac{Opt(I_k)}{A(I_k)} \le 1 + \frac{1}{\delta} + \frac{1}{\delta'} + \frac{1}{\delta\delta'}.$$

Considérons maintenant,

$$t = Arg \max_{1\le i\le m}\left\{ t = i \left| \left\lfloor \frac{L_k}{l_i} \right\rfloor \left\lfloor \frac{W_k}{w_i} \right\rfloor c_i \right.\right\},$$

Donc pour, $\delta = \delta' = 1$, la solution fournie par l'algorithme *HSG* est la valeur de la solution optimale, celle réalisée par une seule pièce, car δ et δ' désigne respectivement la plus petite longueur et hauteur de l'ensemble ***S***. Par contre, pour $\delta \ge 1$ et $\delta' \ge 2$ (ou $\delta \ge 2$ et $\delta' \ge 1$) on obtient :

$$\frac{Opt(I_k)}{A(I_k)} \le \frac{6}{2},$$

finalement ;

$$\frac{Opt(I_k)}{A(I_k)} \ge \frac{1}{3} \quad \text{pour } k = 1,\ldots,m.$$

Ce qui nous permet d'avoir pour l'ensembles des supports en stock

$$\sum_{k=1}^{m} A(I_k) \ge \frac{1}{3}\sum_{k=1}^{m} Opt(I_k).$$

par conséquent ;

$$\frac{A(I)}{Opt(I)} \ge \frac{1}{3}$$

puisque :

$$\sum_{k=1}^{m} A(I_k) = A(I) \text{ et } \sum_{k=1}^{m} Opt(I_k) = Opt(I).$$

(ii) Pour le cas non pondéré, on effectue les mêmes calculs, seulement la borne supérieure est donnée par

$$Opt(I_k) \le \frac{L_k H_k}{l_t h_t} c_t$$

et

$$A(I_k) \ge \left\lfloor \frac{L_k}{l_t} \right\rfloor \left\lfloor \frac{W_k}{w_t} \right\rfloor c_t$$

avec :

$$t = Arg \max_{1 \le i \le m} \left\{ t = i \left| \frac{L_k}{l_i} \frac{H_k}{h_i} c_i \right. \right\} .$$

dans ce cas, on ne peut pas affirmer que la solution est optimale si $\delta = \delta' = 1$. Par conséquent, le rapport d'approximation pour la version pondéré du problème *GTDGC* est au moins égal à $\frac{1}{4}$.

Ce résultat peut être considéré comme une généralisation du résultat donné par Fayard et Zissimopoulos[63] qui considère seulement la version non pondérée du *TDGC* (le nombre de rectangle en stock est réduit à un et le poids considéré est la surface des pièces à découper)

4.5. Expérience numérique

Nous avons testé numériquement le comportement des approches développées *EAG* et *HSG* sur des instances générées aléatoirement. Nous avons également testé la performance de l'algorithme exact en le comparant à la version standard de l'algorithme exact de Beasley (*BEA*)[8], l'algorithme qui utilise le second principe (voir section 4.3.1). Tous les algorithmes ont été programmés en langage C, et exécuté sur environnement Unix (station *SUN*).

On a considéré deux groupes d'instances générées aléatoirement. Le premier groupe (*Gr 1*) est composé de 100 instances, de dimensions *L* et *H* compris entre 80 et 250 et le nombre de supports rectangulaires en stock $m \in$ [7,15]. Le nombre de pièces à découper est pris dans l'intervalle [20,25]. Le second groupe (*Gr 2*) est composé aussi de 100 instances, avec *L* et *H* compris dans [250,500] et $m \in$ [30,50], le nombre de pièces à découper est

compris entre 20 et 100. Pour les deux groupes d'instances les dimensions l_i et h_i de chaque pièce sont prises respectivement dans les intervalles

$$\left]0, \max_{1\le j\le m} L_j\right[\text{ et } \left]0, \max_{1\le j\le m} H_j\right[$$

Pour la version pondérée du problème *GTGDC*, nous appliquons la même procédure de génération des poids des pièces, que celle utilisée pour la version pondérée du problème *TDGC* (voir section 2.5.).

Groupes	*Gr1*	*Gr2*	*Moyenne*
Temps Moy. d'exec (sec) *BEA*	17.04	93.48	55.26
Temps Moy. d'exec (sec) *EAG*	11.75	57.37	34.56
%.gain de temps (*EAG*) versus *BEA*	31.04	38.63	37.46

Table 1. Performance de l'algorithme *EAG* versus la version standard utilisant le second principe.

Sur la table 1, on observe que l'algorithme *EAG* est nettement plus performant que l'algorithme *BEA*. Le pourcentage de gain moyen en temps d'exécution sur l'ensembles des instances traitées est de l'ordre de 37.46%. Un point important est à souligner, nous constatons que le gain moyen en temps d'exécution augmente avec la taille des instances. Ceci peut s'expliquer par le fait que l'algorithme *EAG* utilise le principe de saut de points, donc néglige certains points de l'espace de recherche (appelé région non traitée), contrairement à la version standard *BEA*, basée sur le second principe qui parcours tous les points.

La table 2 montre les performances de l'approche *HSG* comparée à l'algorithme exact *EAG*. Nous avons examiné aussi bien les qualités des valeurs des solutions que le temps de calcul requis. Pour l'ensemble des instances traitées, des solutions d'excellentes qualités ont été obtenues en un temps moyen d'exécution représentant 2.65% du temps moyen requis par l'algorithme exact. De plus, le rapport d'approximation moyen considéré pour toutes les instances est égale à 0.995, ce qui fait des solutions faisables de l'approche *HSG*, très proche des solutions optimales.

Nous constatons dans le groupe 2, que pour un rapport d'approximation moyen égale à 0.994 le temps moyen d'exécution requis représente 6.47% de celui de l'algorithme exact *EAG*.

Pour les instances du groupe 2, cette efficacité constatée est encore plus importante : des solutions satisfaisantes un rapport d'approximation moyen égale à 0.996, nécessitent un temps d'exécution de l'ordre de 1.99% du temps réalisé par l'algorithme *EAG*. Nous pensons qu'une combinaison de ces conditions de limitation sur la qualité des solutions avec une recherche arborescente basée sur la stratégie du meilleur d'abord ou profondeur d'abord, pourrait constituer une puissante approche de résolution pour le problème *GTDGC* .

Groupes	*Gr1*	*Gr2*	Moyenne
Temps Moy.d'exec (sec) *EAG*	11.75	57.37	34.56
Temps Moy.d'exec (sec) *HSG*	0.76	1.34	1.05
(%) time HSG versus *EAG*	6.47	1.99	2.65
Rapp d'approx Moy	0.994	0.996	0.995

Table 2. Performance de l'algorithme *HSG* versus *EAG.*

4.6. Conclusion

Nous avons proposé une méthode exacte et une heuristique [41] pour la résolution du problème de découpe sans contraintes généralisé. Le principe de l'approche exacte repose sur l'utilisation de la procédure de Gilmore et Gomory [31] et de l'introduction d'une nouvelle stratégie qui permet de réduire l'espace de recherche considéré. Au début, nous donnons la définition de supports rectangulaires *non dominants*, par la suite nous démontrons que la présence de tels supports peut être omise sans affecter la solution optimale. L'algorithme à été comparé à une adaptation de l'approche de Beasley[8] sur un ensemble d'instances générées aléatoirement. Sur l'ensemble des instances, notre algorithme était largement supérieur.

La méthode heuristique destinée particulièrement pour le traitement d'instances de grande taille est basée sur le principe du rectangle fictif. Nous utilisons, une réduction du problème initial sous forme de problèmes de sac à dos unidimensionnels. La solution approchée de chaque support est représentée par la combinaison de quelques bandes horizontales ou verticales disponibles, de dimensions inférieures ou égales à celle du support considéré. Finalement, nous avons démontré que l'utilisation d'une procédure gloutonne (dans la résolution des problèmes de sac à dos pour la génération des différentes bandes) garantissait à la méthode heuristique un rapport d'approximation polynomial constant.

Il existe un nombre de problèmes ouvert pour lesquels il serait intéressent de développer des approches séquentielles ou parallèles, en s'appuyant notamment sur des techniques de l'intelligence artificielle, nous citerons à cet effet le problème de découpe non guillotine généralisé avec contraintes de niveau.

Chapitre 5

Un algorithme optimal pour le problème d'assemblage

5.1 Introduction

Dans ce chapitre, nous étudions principalement le problème d'assemblage rectangulaire, qui consiste à regrouper un certain nombre de petites pièces rectangulaires dans un rectangle final de surface minimale. Ce problème est une généralisation du problème d'assemblage sur bande connu sous le nom de bin packing [26] et possède de nombreuses applications intéressantes dans les processus de production de diverses industries, tels que : les placements de circuits intégrés en électronique, la conception des ouvrages en parpaing ou en briques, et autres dérivés dans l'industrie du ciment.

Des contraintes sont imposées sur la manière de regrouper ces pièces, en effet le regroupement peut-être effectué en utilisant l'une des découpes présentées (chapitre 2), étant donné que le dual du problème d'assemblage rectangulaire est le problème de découpe guillotine à deux dimensions. Ceci nous à amener à utiliser une stratégie d'assemblage basée sur des constructions orthogonales dérivées des modèles de découpes guillotines. Dans la section 5.3, nous décrivons les différentes étapes d'un algorithme exact pour le problème d'assemblage. L'algorithme repose sur une méthode arborescente ascendante, une borne supérieure initiale est calculée à l'aide d'une heuristique gloutonne, une stratégie de branchement est définie de manière à s'orienter vers un choix du nœud de surface d'utilité maximum. Par la suite nous introduisons certains raffinements sur l'algorithme tendant à améliorer sa performance. Ces modifications s'apparentent à une stratégie de coupe branche permettant de stériliser certains nœuds de l'arborescence et à une approche heuristique pour l'évaluation d'une solution locale. Finalement, nous évaluons les performances des approches proposées sur un nombre d'instances.

5.2. Présentation du problème

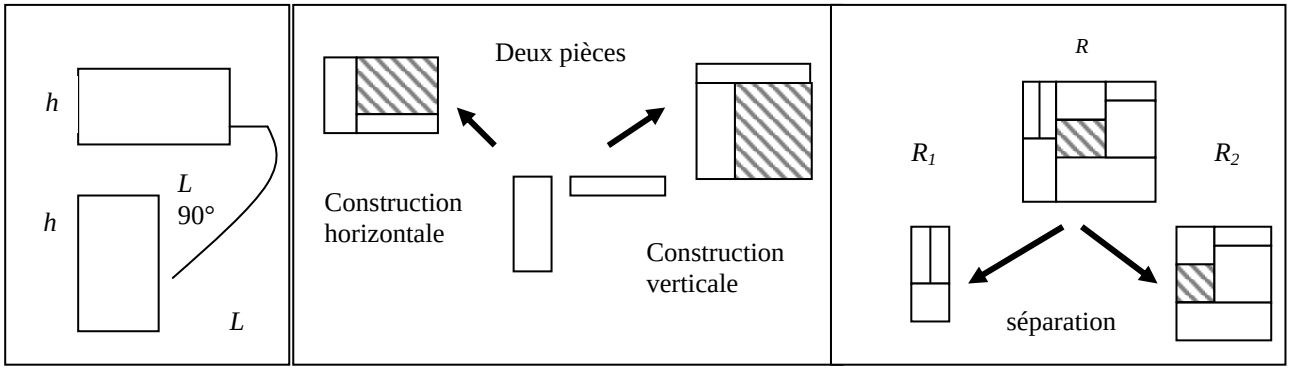
Une instance du problème d'assemblage rectangulaire (RP) est représentée par (**S**, b), ou **S** = $\{(l_1,h_1),(l_2,h_2),\cdots,(l_n,h_n)\}$ est un ensemble de pièces rectangulaire de longueur l_i et de hauteur h_i , pour *i=1,...n.*

De plus à chaque type de pièces est associé une contrainte de demande b_i, pour $i=1,\cdots,n$, i.e. la pièce *i* doit figurer exactement b_i fois dans l'assemblage final. Une solution optimale est le rectangle d'assemblage de surface minimale, regroupant l'ensemble des pièces de ***S*** avec leurs duplications.

Une stratégie développée (chapitre 3) pour la résolution du problème de découpe (non) pondéré sans contraintes produit des modèles de découpes réalisables, en combinant les pièces à découper avec leurs duplications sous forme de constructions horizontales et verticales. Nous utilisons ces formes de construction orthogonales pour la génération des assemblages rectangulaires, nous adoptons à cet effet, les définitions suivantes.

Définition 5.2.1.

Soient *p* et p' deux pièces **S** de dimensions respectivement (l,h) et (l',h'). Un assemblage rectangulaire obtenu par construction horizontale (resp. verticale) des pièces *p* et p' est l'assemblage de dimensions $(l+l',max\{h,h'\})$, (resp. $(max\{l,l'\},h+h')$) .(Figure 5.1.b). Les assemblages que nous utiliserons tout au long de ce chapitre désignent des assemblages rectangulaires.



(a) (b) (c)

Figure 5.1. **(a) Rotation sur chaque assemblage (pièce)**
(b) Construction horizontale et verticale
(c) Construction orthogonale

De plus le nombre d'apparitions de chaque type de pièces dans chacune des constructions est $\leq b_i$, pour $i = 1, \cdots, n$

Définition 5.2.2. un assemblage rectangulaire *R* est dit terminal, s'il contient toutes les pièces avec leurs duplications. On désigne par t-assemblage un assemblage terminal.

Remarque 5.2.1. Chaque assemblage peut-être orienté d'une rotation de 90°
(Figure 5.1.a). Dans ce cas, la définition 5.2.1 peut-être élargie aux assemblages non fixés, ainsi chaque construction verticale ou horizontale entre deux assemblages produit huit assemblages, quatre parmi eux sont distincts et les quatre autres sont identiques aux premiers (orientés de 90°).

De ce fait, la complexité de tout processus de conception d'assemblage augmente considérablement lorsque les rotations des pièces sont permises. Nous utilisons la propriété suivante afin de détecter et d'éliminer les formes répétées sur les assemblages.

Propriété 5.2.1.

Considérons R_1 et R_2 deux assemblages, on dit que R_2 représente une copie conforme de R_1, si sa structure est identique à celle de R_1 (dans le sens ou il contient les mêmes pièces), ou lorsque la structure de R_2 est plus petite que celle de R_1 et que R_2 ne peut pas produire une meilleure solution que R_1. Dans notre étude, nous considérons que chaque pièce et chaque assemblage (définition 5.2.1.) est fixé et les rotations ne sont pas permises.

5.3. Algorithme de résolution

L'algorithme repose sur une technique de séparation et d'évaluation. A cet effet il est indispensable de calculer une borne supérieure afin de pouvoir limiter les branchements sur les sommets de l'arborescence de recherche.
Nous proposons une procédure rapide pour l'évaluation d'un assemblage de départ, qui constitue une solution faisable initiale pour l'algorithme exact.

5.3.1. Evaluation d'une borne supérieure initiale

Une procédure intuitive qui permet de calculer une borne supérieure initiale pour l'approche exacte commence par construire deux t-assemblages. Le premier est un t-assemblage horizontal (figure 5.2.a) qui réalise la valeur :

$$E_h = \left(\sum_{i=1}^{n} b_i l_i \right) \left(\max_{1 \le i \le n} \{h_i\} \right) \qquad (1)$$

Le second est un t-assemblage vertical (Figure 2.b) défini par :

$$E_v = \left(\sum_{i=1}^{n} b_i h_i \right) \left(\max_{1 \le i \le n} \{l_i\} \right) \qquad (2)$$

où b_i, l_i *et* h_i désigne respectivement la contrainte sur la demande, la longueur et la hauteur de $i^{éme}$ pièce, pour $i=1,...n$.

Une évaluation initiale consiste à choisir l'assemblage parmi les deux précédents qui réalise :

$$E_{sup} = min\ (E_h, E_v) \qquad (3)$$

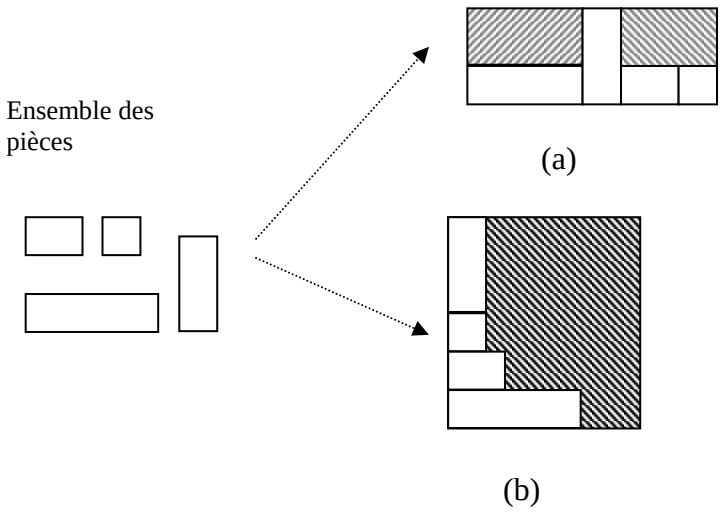


Figure 5.2. Représentation d'un : (a) t-assemblage horizontal
(b) t-assemblage vertical

L'évaluation de l'assemblage calculé précédemment sera considérée comme la première borne supérieure du problème. L'heuristique gloutonne que nous proposons à pour but d'améliorer cette borne. On considère à cet effet une liste ξ contenant l'ensemble de toutes les pièces à regrouper avec leurs duplications, de cette liste on construit dans une première étape l'assemblage R_{init} constitué uniquement des pièces de hauteur maximale. Dans l'étape principale de l'heuristique, on porte des raffinements sur cet assemblage, en effectuant des constructions horizontales successives de la manière suivante : On choisit la pièce de longueur maximale, on la place à droite du rectangle initial R_{init}, on effectue par la suite un remplissage avec les pièces restantes d'une manière gloutonne sur le sous rectangle restant $S_{rest} = (l_{rest}\ ,\ h_{rest}\ )$(voir figure 5.3.). Soit E_{init} l'évaluation de la surface d'assemblage de R_{init} résultant, la procédure s'arrête si $E_{init} \geq E_{sup}$, dans ce cas E_{sup} est la meilleure borne supérieure de départ par construction horizontale. Sinon cette étape est répétée jusqu'à $\xi = \varnothing$.

Algorithme 1 Raffinement de la borne supérieure

- Input : un ensemble de pièces rectangulaires (l_i,h_i), avec leur contraintes b_i, avec $1 \leq i \leq n$.
- Output : une solution sous-optimale notée E_{sup} .

Etape initiale :

1/ Calculer E_{sup} par l'expression donnée en (3) .

2/ Soit ξ l'ensemble de toutes les pièces avec leurs duplications.

choisir la pièce R tel que :

$$h_R = \max_{R' \in \xi} \{h_{R'}\}$$

3/ Etablir la liste : $(R_{Init}, l_{rest}, h_{rest})$

avec $R_{Init} = (b_R l_R, h_R)$, $l_{rest} = 0$ et $h_{rest} = 0$

Etape principale :

- Répéter
- Choisir la pièce $R' \in \xi$ de plus grande longueur

$$l_{R'} = \max_{R \in \xi} \{l_R\}$$

- Effectuer une construction* horizontale de R' avec R_{Init} .
- Poser $\xi = \xi / \{R'\}$
- Effectuer un remplissage du rectangle** $S_{rest} = (l_{rest}, h_{rest})$ avec les pièces de l'ensemble en cours ξ, en utilisant <u>une procédure gloutonne</u>. Soit E_{Init} la valeur de la surface de R_{Init}
- jusqu'à : $(\xi = \phi)$ ou $(E_{sup} \leq E_{Init})$

- ***Etape terminale :***

  si $E_{Init} < E_{sup}$ alors poser $E_{sup} = E_{Init}$

---

* Chaque construction horizontale s'effectue selon le schéma qui consiste à placer la pièce R' à droite de R_{Init} .

** Dans ce cas, la nouvelle liste $Init = (l_{Init} + l_{R'}, h_{Init})$, $l_{rest} = l_{R'}$ et $h_{rest} = h_{Init} - h_{R'}$

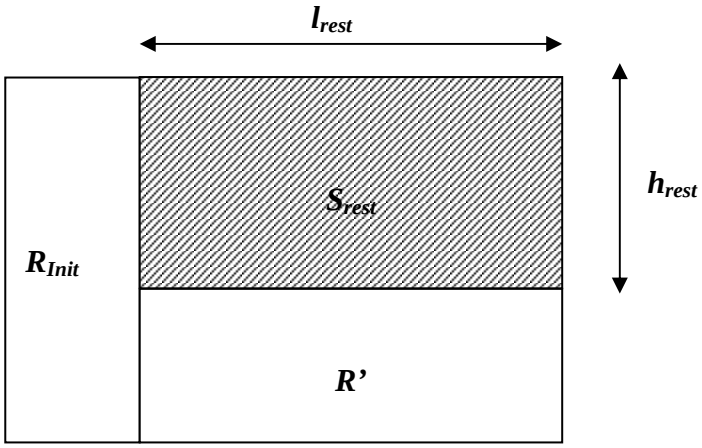


Figure 5.3.

Construction d'un assemblage partiel avec le sous rectangle restant S_{rest}

5.3.2. Solution locale sur un sommet de l'arborescence

Etant donné un assemblage partiel *R* en cours, la construction du meilleur t-assemblage passe par la recherche d'une borne supérieure des t-assemblages contenant *R*. Soit *g(R)* *la fonction d'utilité* sur *R* définie par la somme des surfaces des pièces composant *R.*

- *C(R)* : *la chute* sur le rectangle *R* ; c'est la différence entre la surface totale du rectangle *R* et la somme des surfaces de pièces qui contribuent à sa construction *(g(R)*).
- *h(R)* : la plus petite surface (supplémentaire pour constituer un t-assemblage faisable) qui couvre le reste des contraintes de demande, sans considérer l'ensemble des pièces composant *R.*

Ainsi ;

$$f(R) = G(R) + h(R)$$

où :

$$G(R) = g(R) + C(R)$$

représente la valeur minimale de l'assemblage faisable contenant *R* .

Une estimation (valeur approchée) de *h(R)*, notée *h'(R*) peut-être obtenue sans grands calculs comme suit :

Soit x_i le nombre d'occurrences de la $i^{éme}$ pièce dans *R*. la valeur de *h(R)* peut-être calculé par :

$$h'(R) = \sum_{i=1}^{n} c_i (b_i - x_i) \qquad (4)$$

où : $c_i = l_i h_i$ est la surface de la $i^{éme}$ pièce .

Ainsi ;

$$f'(R) = G(R) + h'(R) .$$

L'algorithme 2 décrit les différentes étapes de la méthodes de résolution. L'algorithme utilise deux listes ξ_1 et ξ_2. ξ_1 contient initialement les pièces (assemblages) R_i de ***S,*** $i=1,...,n$, de dimensions (l_i,h_i), la valeur de la fonction d'utilité $g(R_i) = l_i h_i$, les estimations $h'(R_i)$, *la borne inférieure* $f'(R_i)$ et un vecteur d de dimension *n* (avec $d_k \leq b_k$, $k=1,...,n$, est le nombre d'occurrences de la $k^{éme}$ pièces dans R_i). la liste ξ_2 est initialement vide.

A chaque itération l'assemblage *R* vérifiant $f'(R) = \min_{R_i \in \xi_1} f(R_i)$ est transféré de ξ_1 vers ξ_2 .

On introduit une liste intermédiaire ξ_3 constituée des assemblages construits à partir des différentes combinaisons des éléments de ξ_2 (*R* inclus) avec l'élément *R* en utilisant les constructions horizontales et verticales. La liste ξ_3 contient seulement les éléments *R'* tel que :

$$d_k \leq b_k, \; k=1,...n, \text{ et } f'(R') < Opt.$$

où *Opt* est la valeur de la meilleure solution en cours. L'algorithme s'arrête lorsque la valeur *f'(R)* de l'élément sélectionné $R \in \xi_1$ est supérieure ou égale à *Opt* ou bien lorsque ξ_1 est vide.

Algorithme d'assemblage (IA*RP)

ξ_1 : ensemble de sous problèmes.

ξ_2 : liste des meilleurs sous problèmes en mémoire.

R, R' : des assemblages.

f'(R) : une borne inférieure (surface) du sous problème contenant *R* .

Opt : valeur de la meilleure solution en cours.

Input : une instance du problème d'assemblage rectangulaire (RP).

Output : une solution optimale *Opt.*

Initialisation :

$\xi_1 = \{R_1,...,R_n\}$; $\xi_2 = \phi$.

- Soit E_{sup} une borne supérieure obtenue par l'heuristique gloutonne (Algorithme 1) .
- Poser $Opt = E_{sup}$.

Etape principale :

Répéter :

- Choisir un assemblage *R* tel que :

$$f'_{\min}(R') = \min_{R \in \xi_1} f'(R)$$

Si $Opt - f'_{\min}(R') \leq 0$ alors terminer

Sinon

Faire

Transférer l' assemblage R' de ξ_1 vers ξ_2 et construire tous les éléments de ξ_3 tel que :

- ξ_3 est composé des constructions orthogonales entre les éléments de ξ_2 et R'.
- Chaque élément de ξ_3 vérifie les contraintes b_i $(1 \leq i \leq n)$ et tel que : $f' < Opt$.

- S'il existe un assemblage terminal $P \in \xi_3 / f'(P) < Opt$, alors :

poser $Opt = f'(P)$

Poser $\xi_1 = \xi_1 \cup \xi_3$

Mettre à jour

$\xi_1 = \xi_1 / \{$ assemblage avec $f' \geq Opt\}$

Si $\xi_1 = \phi$; terminer

Sortir avec la valeur de la solution optimale *Opt.*

Nous allons maintenant énoncer un certain nombre de résultats qui justifient le choix des structures des assemblages utilisés par notre algorithme, de mêmes ceux concernant sa finitude et sa convergence.

Théorème 5.3.1

Soient *R* et *R'* deux assemblages initiaux, une construction horizontale (resp. verticale) représente l'assemblage de surface minimale parmi toutes les constructions horizontales (resp. verticales) sur les deux assemblages *R* et *R'* .

Preuve : la propriété est démontrée ici dans le cas d'une construction horizontale, dans le cas de construction verticale la preuve est établit de manière similaire. Nous supposons que les dimensions des assemblages initiaux sont données respectivement par (l,h) et (l',h').Il est clair que, l'évaluation de l'assemblage induit par une construction horizontale est donnée par

$$E = (l+l')(\max\{h,h'\}).$$

Une autre construction horizontale peut-être effectuée par le placement de R à droite de R', tout en déplaçant de t unités vers le haut en direction de la hauteur de R (voir figure 5.4.3.a). Dans ce cas, nous obtenons les situations suivantes :

$$h+t \le h', \quad \text{i.e.} \quad E \le E' = (l+l')h'$$

et dans le cas contraire (figure 5.4.3.b), on a :

$$h+t > h', \quad \text{i.e.} \quad E \le E' = (l+l')(h+t)$$

De manière similaire si le placement de *R* est effectué vers le bas de t' (figure 3.c) alors nous obtenons le même résultat, donc E est bien l'évaluation de l'assemblage de surface minimale.

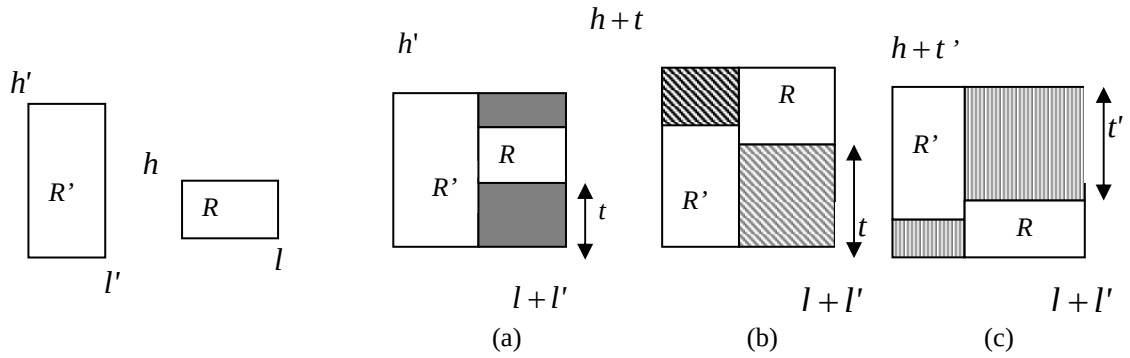


Figure 5.4. Assemblage horizontal entre avec différentes translations possibles.

Le théorème 5.3.1. montre que les assemblages obtenus par construction horizontale (respectivement verticale) conformément aux notions introduites dans la définition 5.2.1., sont les meilleurs configurations d'assemblages parmi toutes les constructions horizontales (respectivement verticales) possibles ayant la même structure (composé de pièces identiques).

Par conséquent, que l'utilisation des constructions horizontales et verticales avec des décalages ne sert qu'a l'introduction des duplications d'assemblages de même surface dans la liste ξ_1. Il s'ensuit immédiatement les conséquences contenues dans le résultat suivant

Corollaire 5.3.1.

Soient *R* et *R'* deux assemblages initiaux. Soit R_1 (resp. R_2) un assemblage engendré par construction horizontale (resp. verticale) sur les deux assemblages initiaux, la chute sur l'assemblage R_1 (resp. sur R_2) est supérieure ou égale à la somme des chutes sur les assemblages initiaux. Ainsi la fonction *C(R)* est croissante par construction horizontale (resp. verticale) .

5.3.3. Convergence de l'algorithme

Lemme 5.3.1.

Chaque assemblage sélectionné de la liste ξ_1, ne pourra plus la réintégrer.

Preuve :

Par récurrence, considérons un assemblage $R \in \xi_1$, et $s = 1$, donc R est composé d'une seule pièce. Tout élément de la liste ξ_3 obtenu à partir de constructions horizontales ou verticales réalisent $s \geq 2$ (chaque assemblage de ξ_3 contient moins deux pièces), donc tout élément transféré de ξ_3 vers ξ_1 est composé d'au moins deux pièces. Ainsi, tout élément $R \in \xi_1$ sélectionné avec $s=1$ ne pourra plus réintégrer la liste ξ_1.

Supposons maintenant, que $R \in \xi_1$ possède $s=m+1$ pièces tel que : $1 < m+1 \leq \sum_{i=1}^{n} b_i$, ainsi à une certaine étape de l'algorithme, R a été réalisé en assemblant deux éléments différents $R_1 \in \xi_1$ avec $R_2 \in \xi_2$, R_1 (qui est un élément sélectionné) ayant servi comme pivot, et ayant un nombre de pièces $\leq m$, donc n'appartiendra plus à la liste ξ_1, donc tout assemblage sélectionné de la liste ξ_1 ne pourra plus la réintégrer.

Théorème 5.3.2

L'algorithme IA*RP converge en un nombre fini d'itérations, lorsque $\xi_1 = \phi$ ou $Opt - f'_{\min} \leq 0$; avec Opt la valeur (évaluation) de la solution optimale du problème d'assemblage considéré.

Preuve :

A chaque itération (étape principale) de l'algorithme, un assemblage R est retiré de la liste ξ_1 et d'autres composés de plus d'éléments que R entre dans ξ_1. D'après le lemme précédent, étant donné qu'un élément sortant de la liste ξ_1 ne peut plus la réintégrer, ne sont admis que les nouveaux éléments (assemblages) composés de plus de pièces, le fait qu'il y ait un nouvel assemblage avec un grand nombre de pièces implique qu'un assemblage déjà sélectionné, ayant servi à sa construction est de facto exclu de ξ_1. Evidemment tel que c'est précisé dans l'algorithme, au bout de quelques itérations, seulement les *t-assemblages* R qui apportent une amélioration à la fonction objective en cours, vérifiant $f'(R)<Opt$. restent

admis dans la liste ξ_1, comme leur nombre est en cesse décroissant, donc fini, l'algorithme s'arrête en un nombre fini d'étapes.

D'autre part, il est clair que :

si $Opt - f'_{\min} \leq 0$ est vérifié alors *Opt* est la valeur de la solution optimale, car $f'_{min} = \min_{R \in \xi_1} f'(R)$ désigne la valeur de la plus petite fonction objectif dans ξ_1, et qu'une meilleure valeur pour *Opt* ne peut pas être trouvée. D'autre part, si ξ_1 est réduit à l'ensemble vide, alors *Opt* est la valeur de la solution optimale, étant donnée que toutes les combinaisons ont été passées en revue .

5.3.4. Stratégies de branchements

Nous avons mentionné précédemment qu'à afin de rendre l'algorithme plus efficace, nous avons été amenés à développer au sein de l'algorithme des stratégies de branchement appropriées afin de stopper le développement de l'arborescence. Nous présentons ici un résumé des stratégies utilisées.

1. **Heuristique gloutonne.** L'heuristique gloutonne appliquée tout au début de l'algorithme IA*RP pour l'évaluation de la borne initiale est considérée comme la première stratégie de branchement dans la procédure de recherche arborescente. Cette borne consiste à rejeter les assemblages qui ont une évaluation supérieure à la meilleure évaluation en cours (d'un assemblage terminal) ; ce qui permet d'écarter quelques assemblages (non terminaux) des différentes constructions activées sur les éléments des deux ensembles ξ_1 et ξ_2.

2. **Meilleur d'abord.** Afin que l'algorithme évolue d'une manière plus efficace, nous avons adopté une deuxième stratégie qui consiste à tenir compte du nœud sur lequel le développement de l'arborescence est effectué. En effet, cette stratégie consiste à choisir à chaque itération de l'étape principale de l'algorithme, l'assemblage *R** qui réalise $f'(R^*) = \min_{R \in \varepsilon_1} \{f'(R)\}$; ce qui est équivalent au choix $g(R^*) = \max_{R \in \xi_1} \{g(R)\}$, et cela permet d'atteindre plus rapidement une nouvelle borne supérieure *h'(R*)* (meilleure que la précédente) et donc un rejet prématuré de certains assemblages, étant donné que l'évaluation sur le meilleur assemblage en cours diminue.

3. **Coupe branche.** Le résultat suivant décrit la troisième stratégie permettant de stériliser certains sommets de l'arborescence crée. C'est cette stratégie (associée aux deux autres stratégies) que nous incorporons afin d'améliorer les performances de l'algorithme.

Théorème 5.3.3.

Soit R_1 le meilleur assemblage terminal en cours, et soit R_2 un autre assemblage (non nécessairement terminal).

Si *(a)* $G(R_2) \geq G(R_1)$, ou *(b)* $G(R_2) < G(R_1) \wedge C(R_2) \geq C(R_1)$

Alors tout assemblage engendré par des constructions horizontales et/ou verticales sur l'assemblage R_2 conduit vers une solution moins bonne que celle fournie par R_1

Preuve : Le cas *(a)* est trivial. En effet, si $G(R_2) \geq G(R_1)$ alors soit R_2 possède toutes les pièces de **S** avec toutes les duplications, et dans ce cas $G(R_1)$ $(= f'(R_1))$ est la meilleure solution en cours, soit R_2 ne contient pas toutes les pièces et dans ce cas par définition, une construction horizontale (ou verticale) effectué sur l'assemblage R_2 augmente l'évaluation de l'assemblage.

Pour le cas *(b)*, on ne s'intéresse qu'aux assemblages terminaux dans lesquelles toutes les pièces avec leur duplications sont représentées. Soit R_t un t-assemblage construit à partir de R_2. En vertu de la définition 5.2.1. et du corollaire 5.3.1, on a :

$$C(R_t) \geq C(R_2) \geq C(R_1), \text{ et } h'(R_t) = h'(R_1) = 0$$

Par conséquent ;

$$f'(R_t) = g(R_t) + C(R_t) + h'(R_t) \geq g(R_1) + C(R_1) + 0 = f'(R_1)$$

Ainsi, tout assemblage construit à partir R_2 n'engendre pas de solutions meilleures que celle de l'assemblage terminal R_1 .

Dans les deux cas *(a)* et *(b)* il est inutile d'examiner les assemblages construit à partir de l'assemblage d'évaluation $f'(R_2)$.

Dans l'algorithme IA*RP, tout sommet de l'arborescence en cours pour lequel le théorème précédent est vérifié est stérilisé.

5.3.5. Déroulement de l'algorithme

- Phase de construction

On commence par sélectionner un assemblage (ou une pièce) de ξ_1 et de la transférer vers la deuxième liste ξ_2. Par la suite, on introduit une liste auxiliaire notée ξ_3 dans laquelle on effectue toutes les constructions horizontales et verticales possibles, tout en ne gardant que celles qui respectent les contraintes sur les occurrences des éléments de ***S***. De plus on impose au rectangle retenu d'avoir une surface d'assemblage inférieure strictement à la meilleure évaluation de l'assemblage en cours notée *Opt* . A chaque construction d'un rectangle. on effectue l'évaluation des fonctions *f'(R)* et *g(R)* et on récupère la composition de *R* .Une fois la liste ξ_3 construite, on effectue le transfert des éléments de cette liste vers la liste ξ_1 tout en la détruisant. Notons qu'au moment du transfert, si la valeur de la borne supérieure d'évaluation a été améliorée alors qu'il existe un assemblage de surface supérieure ou égale dans ξ_1, celui ci est automatiquement rejeté de la liste ξ_1.

- Phase de reconnaissance

Lors de la récupération de la solution optimale du problème d'assemblage, on est évidemment intéressé par la composition de celle-ci. Afin de retracer sa structure finale représentée par le vecteur $d = (d_1, d_2, \cdots, d_n)$, qui sert aussi à reconnaître le parcours de n'importe quel sommet de l'arborescence, notamment pour les besoins des retours en arrière, pour les opérations de réévaluations de la procédure de recherche arborescente. Comme nous l'avons mentionné, les meilleurs sous problèmes sont stockés dans la liste ξ_2, ainsi la structure de la solution finale peut être récupérée de la liste ξ_2 qui comporte aussi tous les prédécesseurs *«parents »* de cette solution. En fin de compte, en introduisant certaines modifications sur chaque élément de la listes ξ_1 et ξ_2 par l'affectation d'attributs en relation avec la position, la construction verticale (respectivement horizontale), des *parents* des nouvelles constructions etc. La solution finale peut être facilement retrouvée par une simple procédure récursive basée essentiellement sur les positions de chaque assemblage interne (sommet intermédiaire) du t-assemblage (sommet final).

5.4. Adaptation de L'algorithme IA*RP au problème d'assemblage sur bande

Nous considérons maintenant le problème d'assemblage sur bande noté *SP* problème, qui consiste à déterminer la meilleure répartition de l'ensembles de toutes les pièces à placer avec leurs duplications dans la plus petite sous bande de chute minimale.

Nous allons montrer comment résoudre de manière exacte le *SP* problème, par une adaptation [43] de l'algorithme IA*RP.

Pour ce problème, nous considérons les mêmes hypothèses énoncées pour le *RP* problème. les définitions concernant les constructions orthogonales pour le *SP* problème sont dérivées directement de ceux énoncées précédemment (définitions 5.2.1. et 5.2.2.).

5.4.1. Borne supérieure pour le SP problème

Nous allons décrire la procédure qui permet d'obtenir une assez bonne borne supérieure initiale pour le *SP* problème. C'est une heuristique de recherche basée sur la résolution d'une suite de problèmes de sac à dos unidimensionnels. Le principe de l'heuristique se résume comme suit :

1) Poser $r=1$ (indice d'itération), ***S'*** ={toutes les pièces avec leurs duplications } et $V_{sol}=0$ (valeur de la solution initiale).

2) Choisir un sous rectangle de dimension $R=(L,H)$, où L, représente la plus longue pièce de l'ensemble ***S',*** i.e. $L = \max_{p_i \in S'} \{l_i\}$

3) Phase de construction : effectuer un remplissage du sous rectangle en utilisant les problèmes de sac à dos unidimensionnels bornés, on note par V_r^h la valeur de la solution obtenue pour le modèle de construction horizontale.

4) Phases d'évaluation et de complétion : Remplir la partie non utilisée du sous rectangle en respectant les contraintes sur les demandes.

Poser : $V_r^h = V_r^h + V'^h_r$ où : $V'^h_r \geq 0$ est la valeur supplémentaire sur la construction horizontale.

5) Mettre à jour les valeurs sur les b_i correspondant à ***S'***

Poser *r=r+1*, et. $V_{sol} = V_{sol} + V_{r-1}^h$

Si ***S'*** = $\varnothing$. Terminer, sinon répéter les étapes 2 à 6.

Il est clair que la l'assemblage construit constitue une solution faisable, puisque les problèmes de sac à dos unidimensionnels bornés vérifient les contraintes sur les bornes de chaque pièce. Nous allons décrire une itération *(r)* de l'heuristique, particulièrement les étapes 3 et 4 présentées au-dessus.

Etape 3 La modélisation du problème de sac à dos unidimensionnel borné pour le sous rectangle *R=(L,H)*, est comme suit :

$$\begin{cases} V_r^h = Max \sum_{p_i \subset (L,H)} c_i z_i \\ \text{s.c} : \sum_{p_i \subset (L,H)} h_i z_i \le H \\ z_i \le b_i, \; p_i \subseteq (L,H) \end{cases}$$

La valeur de la solution optimale de $f_q(H) = V_r^h$ est obtenue au bout de *q* opérations en générant successivement les $f_1, f_2, \cdots, f_q$ en utilisant l'équation récursive de Christofides Withlock (voir chapitre 3). Initialement, $f_0(x) = 0$, pour *x = 0, ⋯,H,* **S'** est l'ensemble des pièces de dimensions plus petite ou égale que *(L,H).*
Nous pouvons à ce moment, résoudre approximativement le problème de découpe à deux dimensions sur le sous rectangle *(L,H)* en utilisant *(*)* et tout en considérant toutes les pièces de ***S'*** rangées selon un ordre décroissant sur les longueurs, tel que $l_1 \le l_2 \le \cdots \le l_q$.

<u>*Etape 4*</u> Une approche d'identification du vecteur solution du problème de sac à dos, i.e. le vecteur $z^* \in N$ tel que $\sum_{i=1}^{m} c_i z_i^* = f_q(H)$, est basée sur la procédure de Hu [45]. Au cours de l'étape 3 un pointeur $u_k(x)$ est associé à chaque valeur $f_k(x)$, de telle façon que $u_k(x)$

représente l'indice du dernier type d'objet utilisé dans $f_k(x)$. En d 'autres termes, $u_k(x) = r$ signifie que $z_r \geq 1$, donc le $r^{ème}$ objet est utilisé dans $f_k(x)$ et $z_l = 0$ pour tout $l > r$. La valeur $u_k(x)$ mémorise l'historique du premier programme dynamique.

Les conditions sur les bornes pour $u_k(x)$ sont

$$\forall x : 0 \leq x \leq H : u_k(x) = \begin{cases} k & \text{si } f_k(x - l_k) > f_{k-1}(x) \\ u_{k-1}(x) & \text{sinon} \end{cases}$$

Nous allons décrire maintenant, les étapes principales de l'algorithme noté IA*SP. Etant donné le (sous) assemblage *R*, la valeur de la fonction économique est donnée par : $f(R) = G(r) + h(R)$, où $G(R) = g(R) + C(R)$ (son évaluation est la même que dans le RP problème), et $h(R)$ est définie comme suit:

On Suppose que ***S'***={des pièces avec leur duplication}, $\bar{S}$ ={des pièces qui contribue à la construction de *R*}et $\delta = \sum_{p_i \in S' \setminus \bar{S}} c_i$. Dans ce cas la valeur de $h(R)$ est donné par :

$$h(R) = max\{\delta, l_R \times (H - h_R)\}$$

où (l_R, h_R) sont les dimensions de *R* et *H* est la hauteur de la bande initiale . Signalons que les résultats énoncés pour le *RP* problème s'appliquent aussi pour le *SP* problème.

Input: Une instance du problème *SP*
Out put: Une valeur de la solution optimale Opt **.**

Initialisation :
Soient $\xi_1 = \{R_1, \cdots, R_n\}; \xi_2 = \varnothing$;
V_{sol} la valeur de borne supérieure obtenue par
la résolution des problèmes de sac à dos bornés.
poser $Opt = V_{sol}$;
Etape principale :
Répéter
choisir un (sous) assemblage $R \in \xi_1$
de valeur minimale{notée $f_{\min}$}
- transfer R de ξ_1 à ξ_2 et construire tous les éléments de ξ_3
tel que
- ξ_3 contient les éléments obtenues par
construction orthogonales R avec tous les élément de ξ_2;
- La largeur de tous (sous) assemblage ne dépasse pas H.
si $\exists$ un t - asseblage $R' \in \xi_3 \setminus f(R') < Opt$, alors poser $Opt = f(R')$;
poser $\xi_1 = \xi_1 \setminus \{$les assemblagesde valeur $f > Opt\}$;
jusqu'à $\xi_1 = \varnothing$;
Sortir avec la valeur de la solution optimale Opt.

5.5. Développement d'une approche heuristique

Lors de l'application de l'algorithme *IA*RP*, les problèmes d'espaces mémoire peuvent surgir pour certains instances de problèmes, notamment ceux de grandes tailles. Les deux principales listes $\xi_1$et ξ_2 contenant respectivement les meilleurs sous problèmes et le reste des sous problèmes parcourus ont tendance à devenir assez large, nécessitant en conséquence un assez grand espace mémoire. Pour cette raison, nous avons prévu une extension à notre algorithme, sous forme d'une approche heuristique notée HIA* afin de limiter le nombre de constructions. L'heuristique utilise des procédures optimisées de recherche locale qui consiste à choisir le meilleur chemin possible et d'abandonner les autres chemins alternatifs définitivement, au risque bien évidemment de perdre la solution optimale. Nous utilisons à cet effet, une généralisation des stratégies du hill climbing résumés dans les points suivants.

Soit R_{min} le meilleur t- assemblage en cours de valeur $f'(R_{min})$ et avec une chute $C(R_{min})$. On note par R_a l'élément en construction en cours (pas nécessairement terminal) se trouvant dans la liste prévisionnel ξ_3 . Sur chaque élément R_a le pourcentage de la surface inutilisée peut être calculé par l'expression

$$C(R_a) \Big/ f'(R_a) \times 100$$

Ceci nous conduit à introduire les deux stratégies suivantes :

- La première stratégie fixe un seuil sur le pourcentage de surface inutilisée. Ainsi l'élément en cours R_a est rejeté si l'inégalité suivante est vérifiée :

$$\frac{C(R_a)}{f'(R_a)} \geq \frac{C(R_{min})}{f'(R_{min})} \quad (5)$$

où $C(R_a) \neq 0$. L'équation (5) est équivalente à

$$C(R_a)(g(R_{min}) + h'(R_{min})) + C(R_a)C(R_{min}) \geq C(R_{min})(g(R_a) + h'(R_a)) + C(R_a)C(R_{min})$$

qui est aussi équivalente à :

$$C(R_a) \geq C(R_{min})$$

où $g(R_a) + h'(R_a) = g(R_{min}) + h'(R_{min})$ (conformément à la définition de $h'(.)$, équation 3). Toutefois, si $h'(.)$ est calculée selon une autre estimation alors

$$g(R_a) + h'(R_a) \neq g(R_{min}) + h'(R_{min})$$

En d'autres termes, si l'inégalité (5) n'est pas vérifiée alors dans ce cas R_a est accepté s'il réalise l'inégalité suivante

$$\frac{f'(R_a)}{f'(R_{\min})} < \beta_1 \qquad (6)$$

où $\beta_1 < 1$ est un pourcentage fixé à priori.

En vertu du théorème 5.3.2., nous avons montré que si

$$f'(R_a) = G(R_a) + h'(R_a) \leq f'(R_{min})$$

Alors dans ce cas l'algorithme s'arrête et la solution optimale est obtenue. Rappelons que lorsque $C(R_a) = 0$, cette situation se produit pour les deux configurations suivantes :

(a) R_a est composé d'une seul pièce, ou (b) R_a représente un modèle plein. Ces deux cas de figures étant prévus par l'algorithme. Par conséquent, si $C(R_a) > 0$, et si nous voulons décroître la valeur de $h'(R_a)$, nous introduisons un paramètre β_2 , et l'évaluation de l'élément en cours R_a est calculé par :

$$f'(R_a) = G(R_a) + \beta_2 h'(R_a) \qquad \text{où :} \quad \beta_2 \geq 1.$$

- Une manière de limiter le nombre de constructions sans risque de perdre la solution optimale, consiste à restreindre les emplacements des constructions horizontales (resp. verticales) au seul axe désignant l'ensemble des combinaisons linéaires sur les longueurs (resp. les hauteurs) des pièces.

Considérons un rectangle fictif de dimensions (L,H), et soit $R = (l_R, h_R)$ un assemblage obtenu par construction horizontale ou verticale. Dans ce cas la longueur l_R est assimilée au point de l'axe horizontal $\vec{L}$ de même que h_R est un point de l'axe vertical $\vec{H}$. L'ensemble des points sur l'axe horizontal représentant l'emplacement des constructions horizontales sont limitées sur l'ensemble P qui forme les combinaisons linéaires entre les longueurs de chaque type de pièces et les constructions verticales sont bornées par l'ensemble des combinaisons linéaires sur les hauteurs des pièces noté Q . Ces ensembles sont représentés comme suit

$$P = \left\{ x / x = \sum_{i=1}^{n} l_i z_i \leq L, z_i \leq b_i, z_i \in N, h_i \leq H \right\}$$

$$Q = \left\{ y \,/\, y = \sum_{i=1}^{n} h_i z_i \le H, z_i \le b_i, z_i \in N, l_i \le L \right\}$$

Ces points rappelons le caractérisent les points de découpes guillotine pour la résolution du problème de découpe sans contrainte non pondéré (voir chapitre 3). Nous utilisons une variante de la fonction de Christofides et Whitlock [11] pour la génération de ces points qui seront considérés comme points de supports pour des constructions horizontales ou verticales. La fonction récursive utilisée à cet effet est définie comme suit :

$$f_i(x) = \begin{cases} 0 & \text{si } i = 0 \text{ et } x = 0 \\ \infty & \text{si } i = 0 \text{ et } x \neq 0 \\ f_{i-1}(x) & \text{si } i \neq 0 \text{ et } ((x < l_i) \text{ ou } (x > \delta_i^l l_i)) \\ \min\left\{ f_{i-1}(x), \max h_i, \min\limits_{1 \le k \le \min\left\{ \left\lfloor x / l_i \right\rfloor, \delta_i^l \right\}} f_{i-1}(x - k l_i) \right\} & \text{sinon} \end{cases}$$

où $1 \le \delta_i^l \le \min\left\{ \left\lfloor L / l_i \right\rfloor, b_i \right\}$, pour $i = 1, \cdots, n$, et avec la valeur de $f_n(x)$ nous pouvons confirmer si le point x représente une combinaison linéaire ou non des différentes longueurs des pièces à assembler.

Ainsi, étant donné un assemblage $R = (x, y)$, il peut être considéré comme support de construction horizontale (nous utilisons la même forme de fonction pour les limitations de constructions verticales) si et seulement si :

$$x \in P \Leftrightarrow f_n(x) \le y$$

L'approche heuristique *HIA** limite le nombre de constructions crées en éliminant toute nouvelle construction engendrant une longueur (resp hauteur) n'appartenant pas à P (resp. à Q). Pour deux composants d'une construction $R_1 = (x_1, y_1) \in \xi_1$ et $R_2 = (x_2, y_2) \in \xi_2$, le nouvel élément résultant obtenu par construction horizontale (resp. verticale) est écarté si $x_1 + x_2 \notin P$ (resp. $y_1 + y_2 \notin Q$)

Résume de l'approche heuristique

L'heuristique s'applique aussi bien pour le RP problème que pour le *SP* problème dans sa version décrite par les étapes suivantes :

1. Générer la solution initiale du *RP* problème (ou du *SP* problème) en utilisant l'heuristique gloutonne (ou l'heuristique basée sur les problèmes de sac à dos unidimensionnels).

2. Générer les deux ensembles P et Q à l'aide de la version modifiée de la fonction de Christofides.

3. Fixer les seuils de variation β_1 et β_2.

4. Appliquer les étapes principales des algorithmes exactes *IA*RP* pour le problème *RP* (ou le *IA*SP* pour le *SP* problème), en incorporant les équations (5) et (6), et les ensembles P et Q .

5. Sortir avec la valeur de solution optimale et sa structure.

5.6. Résultats numériques

Nous présentons l'étude expérimentale effectuée afin d'analyser les performances des algorithmes proposés. Cette étude a pour but de mettre en évidence l'évolution de la vitesse de convergence de l'algorithme avant et après l'introduction des différents critères de stérilisation sur l'arborescence crée.

Dans une première étape, on teste l'algorithme exact sur un nombre d'instances générées aléatoirement sans l'utilisation de la stratégie de coupe branche (noté version1), afin d'analyser particulièrement l'effet des bornes supérieures (développées par l'heuristique gloutonne) dans la méthode de séparation et d'évaluation. Dans la deuxième étape on introduit cette stratégie dans l'algorithme (noté version 2) et nous la comparons à la première version. Nous évaluons par la suite les performances de l'heuristique comparée à l'algorithme exacte.

5.6.1 Performances de l'algorithme exact

Nous considérons deux groupes d'instances générées aléatoirement. Le premier groupe (*Gr1*) comporte 100 instances de petite taille, le nombre de chaque types de pièce à regrouper est donné dans l'intervalle [5,10], les dimensions des pièces sont prises dans l'intervalle [2,25]. Le deuxième groupe (*Gr2*) est composé aussi de 100 instances, dont le nombre type de pièce est compris dans l'intervalle [10,35] et les dimensions sont prises dans l'intervalle [25,80].

La borne supérieure sur chaque type de pièce est générée dans [1,6] pour le premier groupe, et dans [1,15] pour le second. Le coût de chaque types de pièces est exactement sa surface.

Groupes	Gr1	Gr2	Moyenne
Temps Moyen (sec) pour la (version1)	131.32	457.52	294.42
Temps Moyen (sec) pour la (version2)	98.47	249.69	174.08
% Gain en Temps moyen (version 2/version 1)	25.01	45.58	40.87
Moy du Nb Sommets Exploités (version 1)	13237.58	63239.30	38238.44
Moy du Nb Sommets Exploités (version 2)	9835.25	37202.16	23518.71
% Gain en Moy Nb Sommets Exploités	25.70	41.17	38.49
Rapp d'appr Moy (Opt/Borne Sup)	0.603	0.681	0.642

Table 1. performance de l'algorithme IA*RP: version1 et 2.

Sur la table 1, nous pouvons constater que la version 2 est nettement meilleur que la première. Le gain moyen sur le temps d'exécution pour l'ensemble des instances traitées est de l'ordre de 40.87%. Ce rapport est plus important pour les instances de grande taille, puisqu'il passe de 25.01% en moyenne pour le groupe 1 à 45.58% pour le groupe 2. L'introduction de la stratégie de coupe des branches engendre un gain considérable sur le nombre de sommets exploités. Nous enregistrons à cet effet un gain en moyenne sur le total des sommets exploités de l'ordre de 38.49%, précisément 25.70% pour la version1 et 40.17% pour la version 2.
Nous pouvons dire à cet effet, que l'efficacité constatée notamment sur les instances de moyenne et grande taille reflète les effets conjugués, de l'introduction d'une bonne borne initiale, du choix de la stratégie de branchement, ainsi que de la procédure de stérilisation.

5.6.2.Performance de l'approche heuristique

La performance de l'approche heuristique est évaluée sur les mêmes groupes d'instances (*Gr 1*) et (*Gr 2*) que l'algorithme exact. L'heuristique utilise les stratégies du hill climbing qui consiste à rejeter certains assemblages spécifiques. Cette stratégie se résume dans les équations en les points (5) et (6) énoncées dans le paragraphe 5.5., l'objectif recherché étant de déterminer les ajustements judicieux sur les paramètres qui sont incorporés dans l'algorithme afin d'obtenir le meilleur rapport entre les qualités des solutions et les temps d'exécution.

Les ajustements des paramètres sont établis comme suit : δ_1 et δ_2 répartis uniformément dans [1,2], i.e. $\delta_1 = \delta_1^l = ... = \delta_n^l$ et $\delta_2 = \delta_1^h = ... = \delta_n^h$. De plus les valeurs de $|P_l|$ et $|Q_h|$ sont limités dans l'intervalle de variation [158,265].

Valeurs de β_1 et β_2	(0.85,1.1)	(0.9,1.2)	(0.95,11)	(0.99,1.1)	(0.999,1.1)	(0.938,1.12)
Temps .Moy (sec)	11.31	59.67	22.64	47.35	59.29	40.05
Nb. Moy. Sommets	4002.25	12361.47	6808.82	10232.37	12287.85	9138.55
Rapp Appr. Moy	0.763	0.796	0.869	0.911	0.942	0.856
(%) solutions Optimales	11	23	29	41	55	31.8

Table 2. performance de l'heuristique HIA*-RP en utilisant des variations sur β_1 et β_2

La table 2 nous montre les performances de l'approche pour les différentes variations des paramètres β_1 et β_2. Nous relevons que les meilleures solutions sont obtenues pour β_1 = *0.999* et β_2 = *1.1*, pour ces valeurs le taux d'approximation moyen est de 0.942 en un temps moyen égale à 59.29 sec, ce qui représente seulement 34.06% du temps requis par l'algorithme *IA*-RP*. A noter dans ce cas le pourcentage de solutions optimales atteintes par l'approche *HIA*RP* est de l'ordre de 55%.

5.7. Conclusion

Nous avons développé un algorithme exact pour la résolution du problèmes d'assemblage rectangulaire. L'algorithme repose sur une méthode de séparation et d'évaluation ascendante, avec des stratégies de développement bien définies d'une part et l'introduction de quelques critères d'arrêt (stérilisation) d'autre part. Nous avons par ailleurs proposé une approche heuristique dans le but de réduire l'espace de recherche, par l'introduction de stratégies de coupe branche sur l'arborescence et certains raffinement sur les solutions en cours. Les résultats numériques recueillis sur l'ensembles des instances indiquent que l'algorithme exact est capable de résoudre des instances de petite et moyenne taille de manière efficace en un temps d'exécution raisonnable. Les modifications apportées à l'algorithme par l'heuristique de recherche allège considérablement l'arborescence de recherche et les ajustements appropriés sur les paramètres de variation de l'heuristique augmente de manière significative l'efficacité de l'algorithme pour des solutions de qualité comparées aux solutions optimales.
Nous avons montré par ailleurs que l'algorithme est adaptable pour la résolution du problème d'assemblage sur bande. Par contre, il serait intéressent de concevoir un algorithme général (ou une heuristique) sans l'hypothèse de construction guillotine capable de d'assembler des pièces dans une enveloppe convexe.

Chapitre 6

Problème de découpe de pièces trapézoïdales

6.1 Introduction :

Nous étudions le problème de découpe sans contraintes de pièces de formes trapézoïdales sur un support rectangulaire, introduit par A.Khelladi et noté *PDT* [45]. Il s'agit d'une variante du problème de découpe non orthogonale. Le *PDT* possède de nombreuses applications dans les processus de fabrication de diverses industries: La conception de pipe line (pétrochimie), le design d'aileron (aéronautique) ou les découpes des composants de produits textiles. La section 2, comporte une description détaillée du problème, notamment la notion de fonction de chute utilisée comme critère d'optimalité. Dans la section 3, nous présentons les formes de construction des modèles de bandes homogènes (composé d'un seul type de pièce). Dans la section 4, nous développons une méthode approchée pour le *PDT* basée sur une procédure constructive, permettant d'obtenir le meilleur plan de découpe à partir de la combinaison des bandes homogènes optimales horizontales et verticales.

6.2 Présentation du problème :

Le *PDT* peut être simplement formulé comme : le découpage du maximum de pièces trapézoïdales sur un support rectangulaire, de manière à minimiser le total de la chute. Une instance du *PDT* est représentée par un support rectangulaire *R* de dimension *(L,H)* avec *L* et *H* étant respectivement la longueur et la hauteur de *R*, un ensemble fini de pièces $S = \{t_1, \cdots, t_n\}$ de forme trapézoïdale. A chaque type de pièce t_i est associé un poids c_i représentant la surface de la pièce t_i pour *i=1,…,n.* Le *PDT* consiste à découper le support *R* en stock en petites pièces t_i, sans aucune limitation sur le nombre de pièces produites, de manière à minimiser la valeur de la chute sur l'entité en stock définie par la fonction:

$$C(R, t_1, \cdots, t_n, X) = L.H - \sum_{i=1}^{n} s_i.x_i$$

Où : $X = (x_1, \cdots, x_n)$ désigne un modèle de découpe, x_i étant le nombre d'occurrences de la pièce t_i sur l'entité *R*.

6.2.1 Hypothèses de base :

La recherche du meilleur moyen de placement des pièces à découper est étroitement liée à la forme d'orientation choisie pour les pièces à découper. Pour l'ensemble des trapèzes, nous avons à spécifier les orientations permises, certaines orientations sont typiquement impossibles pour des considérations pratiques. Nous supposons que les rotations et les translations permises sur les trapèzes sont celles qui engendrent des trapèzes stables horizontalement, donc non inclinés. Ainsi, les pièces peuvent être pivotées de 180° (dans les deux sens) uniquement. Il est clair qu'a cet effet, d'une part on suppose que le support est homogène sur ses deux faces et que les découpes effectuées sont du type non orthogonal. D'autre part, on suppose que les dimensions du support et des pièces à découper sont des entiers.

6.2.2 Caractéristiques des pièces à découper :

Les seules caractéristiques à prendre en considération par le *PDT* au sujet des pièces à découper tiennent compte de la forme géométrique de celles-ci. Un trapèze *t=(a,b,c,d)* de forme irrégulière est entièrement spécifié par la donnée des paramètres $(\alpha, \beta, h, \theta, \omega)$ avec:

α : la longueur de la base inférieure ab

β : la longueur de la base supérieure dc

h : la hauteur de *t*

$\theta = \hat{dab}$

$\omega = \hat{cba}$, où $\theta, \omega \in \left]0, \frac{\pi}{2}\right[$. Ceci suppose évidemment que $\beta < \alpha$.

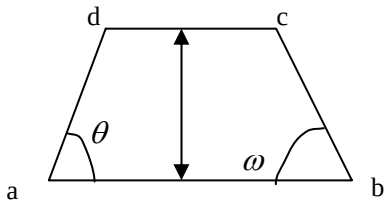


Figure 1

La figure 1 représente un trapèze avec les paramètres associés dont on a besoin pour spécifier entièrement la pièce.

6.2.3 Paramètrisation des formes trapézoïdales

Des considérations détaillées (section 2.1) sur les orientations permises et les positions potentielles des trapèzes résulte certaines formes particulières pour une même pièce $t=(\alpha,\beta,h,\theta,\omega)$. Dans la définition qui suit nous introduisons les notions des formes dupliquées de *t* ainsi que leur paramètrisation.

Définition 6.1 : Le transposé du trapèze est le trapèze $t=(\alpha,\beta,h,\theta,\omega)$ est le trapèze $t^r=(\beta,\alpha,h,\pi-\omega,\pi-\theta)$ obtenu partir de *t* par une rotation de *180°* dans le sens des aiguilles de la montre (figure 2.1).

Le symétrique du trapèze *t* est le trapèze $t^s=(\alpha,\beta,h,\omega,\theta)$ (figure 2.3).

Toutes ces formes de trapèzes dupliquées ont la même surface:

$$s(t)=s(t^r)=s(t^s)=\frac{1}{2}.(\alpha+\beta).h$$

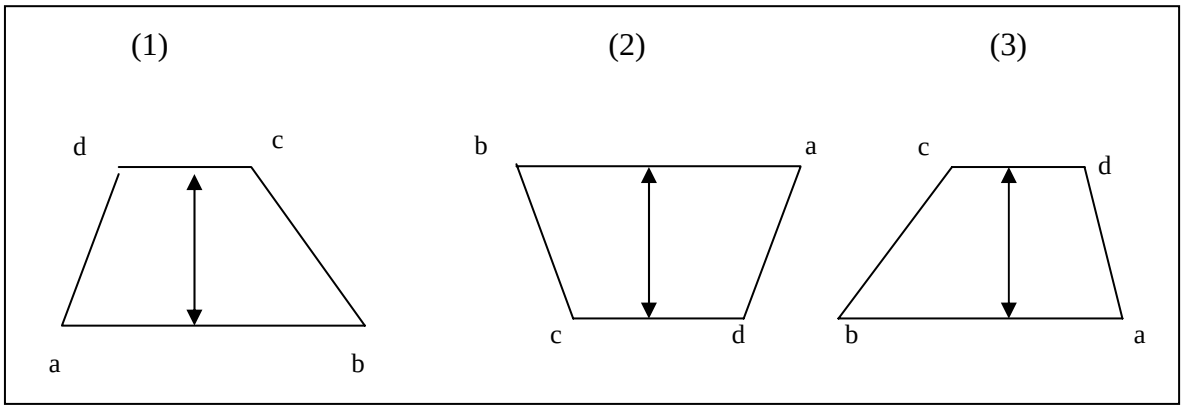


Figure 2

6.2.4 Regroupement des trapèzes

L'expression de la surface du trapèze *t* est obtenue à partir de la surface de la paire de trapèzes $p=(t,t^t)$, *p* représente un regroupement horizontal des deux trapèzes contigus *t* et t^r qui engendre un parallélogramme (figure 3) de dimension $(\alpha+\beta,h)$ et dont les angles sont $(\theta,\pi-\theta)$. Nous adoptons cet aspect constructif basé sur la notion de regroupement contigu des pièces identiques dans la construction des bandes homogènes de trapèzes. Nous donnons la définition suivante pour la construction des différents blocs formants une bande homogène horizontale.

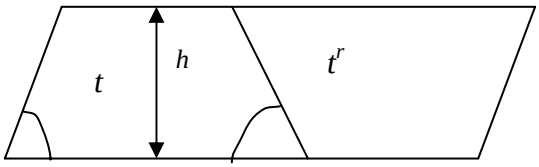


Figure 3

Définition 6.2 : On dit qu'un plan de découpe non orthogonale forme une construction homogène horizontale associée à la pièce *t* si la combinaison des deux trapèzes *t* et t^r engendre un parallélogramme $p = (t, t^t)$ de dimensions $(\alpha + \beta, h)$.

6.3 Modèles de bandes

Le problème de placement consiste à placer un ensemble de formes irrégulières, pouvant être utilisées un certain nombre de fois, dans une surface de dimension fixe, souvent rectangulaire, en optimisant l'occupation de la surface et en respectant certaines contraintes spécifiées dans les formes ou par les conditions de découpe. Les problèmes de placement constituent de ce fait, des sous problèmes disjoints pour le problème de découpe, leur résolution par une méthode appropriée tenant compte des contraintes de choix de superposition de pièces identiques tel que présenté dans la section précédente décrivant la bande homogène, nous permettent de diviser l'ordre de coupe. On ne réalisera pas un seul placement optimal, mais une succession de sous placements (eux-mêmes sous optimaux) que l'on appellera par convention: bande optimale, contrairement au problèmes de découpe de pièces rectangulaires où différents types de modèles de bandes pouvaient être considérés (homogène, uniforme et générale).

Définition 6.3 : Une bande homogène horizontale est une bande où il n' y a qu' un seul type de pièce participante. Une bande homogène d'ordre *k* associée à la pièce *t* est le module rectangulaire de surface minimale contenant une construction homogène horizontale composée de *k* pièces équivalentes à la pièce *t.*

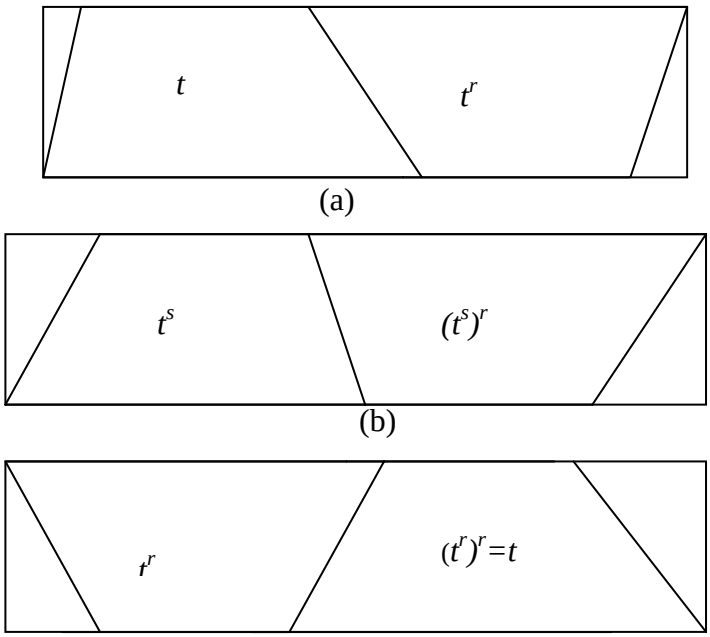


Figure 4 . Représentation des différentes bandes homogènes d'ordre 2.

(a) $R_{t,2}$, (b) $R_{t^s,2}$, (c) $R_{t^r,2}$

Comme nous pouvons le constater sur la figure 4, le regroupement de trapèzes contigus peut s'effectuer de différentes manières (selon le sens de disposition de la pièce considérée). Nous distinguons trois types de bandes homogènes qu' on note par $R_{t,k}, R_{t^r,k}$ et $R_{t^s,k}$ associées respectivement à $t=(\alpha,\beta,h,\theta,\omega)$ et à ses formes équivalentes $t^r=(\beta,\alpha,h,\pi-\omega,\pi-\theta)$ et $t^s=(\alpha,\beta,h,\omega,\theta)$ obtenues par les rotations permises sur la pièce *t*. Nous montrons dans les résultats suivants que parmi toutes les possibilités de regroupement des pièces trapézoïdales dans la génération des bandes homogènes, équivalentes en terme de composition de pièces rentrantes, il existe une configuration optimale en terme de chute dans une bande.

Proposition 6.1 : Etant donné un trapèze $t=(\alpha,\beta,h,\theta,\omega)$. Soient

$$t^r=(\beta,\alpha,h,\pi-\omega,\pi-\theta) \text{ et } t^s=(\alpha,\beta,h,\omega,\theta)$$

les formes dupliquées obtenues respectivement par rotation et par symétrie de la pièce *t*, alors:

1. $(t^r)^s=(t^s)^r$
2. Si : $p=(t,t^t)$ et $p^s=(t^s,(t^s)^r)$, alors :

$$s(p)=s(p^s)$$

Preuve : (1) On a $t^r = (\beta, \alpha, h, \pi - \omega, \pi - \theta)$, ainsi ;

$$(t^r)^s = (\beta, \alpha, h, \pi - \theta, \pi - \omega)$$

D'autre part, $t^s = (\alpha, \beta, h, \omega, \theta)$, il en résulte

$$(t^s)^r = (\beta, \alpha, h, \pi - \theta, \pi - \omega)$$

D'où :

$$(t^r)^s = (t^s)^r$$

(2) Il suffit de constater que le parallélogramme *p* engendré par le regroupement de la paire de trapèzes contigus (t, t^t) la longueur de sa base est $(\alpha + \beta)$, sa hauteur est *h* et ses angles sont $(\theta, \pi - \theta)$. D'autre part, $p^s = (t^s, (t^s)^r)$ est de dimensions $(\alpha + \beta, h)$, toutefois ses angles sont $(\omega, \pi - \omega)$.

Comme conséquence de ce résultat, nous pouvons constater que les bandes $R_{t,k}$, $R_{t^r,k}$ et $R_{t^s,k}$ sont toutes composées de *k* pièces contigus d'égales surfaces. Toutefois les chutes sur ces bandes ne sont pas équivalentes, dans ce qui suit, nous montrons un résultat permettant de caractériser la bande homogène optimale.

Proposition 6.2 : Considérons les bandes $R_{t,k}$, $R_{t^r,k}$ et $R_{t^s,k}$. Les chutes enregistrées sur ces bandes étant respectivement $C(R_{t,k})$, $C(R_{t^r,k})$ et $C(R_{t^s,k})$, on a :

1. $C(R_{t,k}) = \begin{cases} h^2.tg(\frac{\pi}{2} - \theta) & \text{si } k = 2n \\ \frac{h^2}{2}.tg(\frac{\pi}{2} - \theta) + \frac{h^2}{2}.tg(\frac{\pi}{2} - \omega) & \text{si } k = 2n+1 \end{cases}$
2. $C(R_{t^r,k}) = C(R_{t^s,k})$

Preuve : (1) Soient *(l,h)* les dimensions de $R_{t,k}$, nous avons :

$$l = \frac{k}{2}.(\alpha + \beta) + c$$

La chute sur $R_{t,k}$ s'écrit sous la forme suivante :

$$C(R_{t,k}) = l.h - \frac{k}{2}.(\alpha + \beta) = c.h$$

- Si *k=2.n*,

$$c = 2.\frac{h}{2}..tg(\frac{\pi}{2} - \theta) = h.tg(\frac{\pi}{2} - \theta)$$

ainsi ;

$$C(R_{t,k}) = h^2.tg(\frac{\pi}{2} - \theta)$$

- Si *k=2n+1,*

$$c = c' + c''$$

avec :

$$c' = \frac{h}{2}..tg(\frac{\pi}{2} - \theta)$$

$$c'' = \frac{h}{2}..tg(\frac{\pi}{2} - \omega)$$

D'où :

$$C(R_{t,k}) = \frac{h^2}{2}.\left(tg(\frac{\pi}{2} - \theta) + tg(\frac{\pi}{2} - \omega) \right)$$

(2) Les deux bandes $R_{t^r,k}$ et $R_{t^s,k}$ sont équivalentes en termes de pièces qui les composent et qui sont toutes équivalentes à la pièce *t*. En effet; soient $(a_1, a_2, \cdots, a_k)$ et $(c_1, c_2, \cdots, c_k)$ les pièces rentrantes dans $R_{t^r,k}$ et $R_{t^s,k}$ respectivement. Pour tout *i=1,…,k,* nous avons :

$$a_i = \begin{cases} t^r & \text{si i est pair} \\ (t^r)^r & \text{si i est impair} \end{cases}$$

et

$$c_i = \begin{cases} t^s & \text{si i est pair} \\ (t^s)^r & \text{si i est impair} \end{cases}$$

Les chutes enregistrées sur les bandes $R_{t^r,k}$ et $R_{t^s,k}$ s'obtiennent à partir de la configuration des pièces $a_1 = t^r = (\beta, \alpha, h, \pi - \omega, \pi - \theta)$ et

$$a_k = \begin{cases} t^r & \text{si k est pair} \\ (t^r)^r = t & \text{si k est impair} \end{cases}$$

pour $R_{t^r,k}$ ainsi que des pièces $c_1 = t^s = (\alpha, \beta, h, \omega, \theta)$ et

$$c_k = \begin{cases} t^s & \text{si k est pair} \\ (t^s)^r = t & \text{si k est impair} \end{cases}$$

pour la bande $R_{t^s,k}$. Il en résulte ainsi, en vertu du résultat de 1)

$$C(R_{t^r,k}) = C(R_{t^s,k}) = \begin{cases} h^2.tg(\frac{\pi}{2} - \omega) & \text{si } k \text{ est pair} \\ \frac{h^2}{2}.tg(\frac{\pi}{2} - \omega) + \frac{h^2}{2}.tg(\frac{\pi}{2} - \theta) & \text{si } k \text{ est impair} \end{cases}$$

6.3.1. Caractérisation de la bande homogène optimale

Proposition 6.3 : Soient $(b_1, b_2, \cdots, b_k)$ et $(c_1, c_2, \cdots, c_k)$ les pièces composant respectivement $R_{t,k}$ et $R_{t^s,k}$, tels que pour *i=1,…,k ;*

$$b_i = \begin{cases} t & \text{si i est pair} \\ (t^r) & \text{si i est impair} \end{cases}$$

$$c_i = \begin{cases} t^s & \text{si i est pair} \\ (t^s)^r & \text{si i est impair} \end{cases}$$

Avec $t=(\alpha, \beta, h, \theta, \omega)$ et $t^s = (\alpha, \beta, h, \omega, \theta)$. Alors,

$$C(R_{t,k}) \leq C(R_{t^s,k}) \Leftrightarrow \theta \geq \omega$$

Preuve : Nous pouvons facilement exprimer du résultat précédent les chutes sur les bandes $R_{t,k}$ et $R_{t^s,k}$ comme suit :

- Si *k=2n*,

$$C(R_{t,k}) = h^2.tg(\frac{\pi}{2} - \theta)$$

$$C(R_{t^s,k}) = h^2.tg(\frac{\pi}{2} - \omega)$$

- Si *k=2n+1*,

$$C(R_{t,k}) = C(R_{t^s,k}) = \frac{h^2}{2}.\left(tg(\frac{\pi}{2} - \theta) + tg(\frac{\pi}{2} - \omega) \right)$$

Comme la fonction tangente est une fonction croissante, le résultat en découle immédiatement.

Conséquence de ce résultat, dans tous ce qui suit, toute pièce trapézoïdale *t* sera caractérisée par $t=(\alpha, \beta, h, \theta, \omega)$, avec $\theta \geq \omega$.

6.4 Algorithme de résolution

Nous proposons un algorithme de résolution pour le *PDT* notée HDT, basé sur une procédure constructive permettant d'obtenir les bandes homogènes horizontales et verticales. La résolution de deux problèmes de sac à dos unidimensionnels permettent de construire deux modèles de découpe guillotines. Le premier est un plan de découpe horizontal obtenu comme combinaison des bandes homogènes horizontales de différentes hauteurs. Le deuxième plan est un plan de découpe vertical obtenu en combinant les bandes homogènes verticales de différentes longueurs. La solution approchée étant la valeur du meilleur modèle de découpe.

6.4.1. Principe de l'algorithme

Soit une instance du *PDT* définie par *: (R,S,c),* où: *R=(L,H)* est le rectangle initial, *L* et *H* étant sa longueur et sa hauteur respectivement. $S = \{t_1, \cdots, t_n\}$ est l'ensemble des pièces à découper. Chaque pièce t_i est caractérisée par $t_i = (\alpha_i, \beta_i, h_i, \theta_i, \omega_i)$. $c = (c_i, \cdots, c_n)$ est le vecteur poids, tel que

$$c_i = s(t_i) = (\alpha_i + \beta_i).\frac{h}{2} \quad \text{pour } i = 1, \cdots, n$$

Les étapes de l'algorithme *HDT* se résument comme suit

1. Génération des bandes homogènes horizontales. Soient $R_{i,a_i}(L)$ les bandes homogènes horizontales de longueur *L*, composées de a_i pièces t_i et ont pour valeur

$$\lambda_i = c_c.a_i$$

2. Génération des bandes homogènes verticales. Soient $R_{i,b_i}(H)$ les bandes homogènes verticales de hauteur *H*, obtenues par regroupement vertical. Chaque bande est composée de b_i pièces t_i et de valeur

$$\xi_i = c_c.b_i$$

3. Modèle de découpe horizontal. Ordonner les éléments de *S* selon un ordre décroissant tel que :

$$\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_r$$

où r est le nombre de différentes valeurs des $R_{i,a_i}(L)$. Résoudre le problème de sac à dos suivant :

$$\begin{cases} F_{hor} = \max \sum_{j=1}^{r} \lambda_i . x_j \\ sc : \\ \sum_{j=1}^{r} h_j . x_j \le H \\ x_j \in N, j = 1, \cdots r \end{cases}$$

F_{hor} est la valeur du modèle de découpe horizontal $M_{hor} = \left(x_j\right)_{j=1,\cdots,r}$, avec x_j est le nombre d'occurrences de la bande $R_{j,a_j}(L)$ dans M_{hor}.

4. Ordonner les éléments de *S* selon un ordre décroissant tel que :

$$\xi_1 \le \xi_2 \le \cdots \le \xi_{r'}$$

où *r'* est le nombre de diff*é*rentes valeurs des $R_{i,b_i}(H)$, résoudre le problème de sac à dos suivant :

$$\begin{cases} F_{ver} = \max \sum_{j=1}^{r'} \xi_i . y_j \\ sc : \\ \sum_{j=1}^{r'} h_j . y_j \le L \\ y_j \in N, j = 1, \cdots r' \end{cases}$$

F_{ver} est la valeur du modèle de découpe vertical $M_{ver} = \left(y_j\right)_{j=1,\cdots,r'}$, avec y_j est le nombre d'occurrences de la bande $R_{j,b_j}(H)$ dans M_{ver}.

La valeur de la solution est :

$$F^* = \max\left(F_{hor}, F_{ver}\right)$$

Proposition 6.4 : le *PDT* admet un rapport d'approximation vérifiant :

$$\frac{A(I)}{Opt(I)} \ge \frac{1}{3}$$

Où : *A(I)* est la valeur de la solution partielle fournie par l'algorithme *HDT* et *Opt(I)* est la valeur de la solution optimale de l'instance *I*.

Preuve : L'heuristique *HDT* réalise le meilleur modèle de découpe homogène associée à la pièce t_i pour $1 \le i \le n$, donc vérifie :

$$A(I) \ge \left| \frac{L}{(\alpha_i + \beta_i)/2} \right| . \left| \frac{H}{h_i} \right| . c_i$$

Où :

$$c_i = (\alpha_i + \beta_i) . \frac{h_i}{2}$$

On pose :

$$\delta = \left| \frac{L}{(\alpha_i + \beta_i)/2} \right|$$

$$\delta' = \left| \frac{H}{h_i} \right|$$

Par ailleurs, la solution optimale vérifie

$$Opt(I) \le L.H$$

Ce qui nous permet d'avoir

$$\frac{Opt(I)}{A(I)} \le \frac{L.H}{\delta.\delta'.(\alpha_i + \beta).\frac{h_i}{2}_i}$$

D'autre par, on a

$$\left| \frac{L}{(\alpha_i + \beta_i)/2} \right| . \left| \frac{H}{h_i} \right| \le \frac{L.H}{(\alpha_i + \beta).\frac{h_i}{2}_i} \le \left(\left| \frac{L}{(\alpha_i + \beta_i)/2} \right| + 1 \right) . \left(\left| \frac{H}{h_i} \right| + 1 \right)$$

Ainsi

$$\delta.\delta' \le (\delta + 1).(\delta' + 1)$$

Par conséquent

$$\frac{Opt(I)}{A(I)} \le \frac{(\delta + 1).(\delta' + 1)}{\delta.\delta'}$$

Donc, pour $\delta \ge 1$ et $\delta' \ge 2$ (ou $\delta' \ge 1$ et $\delta \ge 2$), on obtient

$$\frac{A(I)}{Opt(I)} \ge \frac{1}{3}$$

6.4.2. Déroulement de l'algorithme sur un exemple

Considérons l'instance *(R,S,c),* avec *R=(7.5)* et $S = (t_1, t_2, t_3)$, où:

$$t_1 = (3,2,1), t_2 = (2,1,2), t_3 = (3,1,3)$$
$$c_1 = 5/2, c_2 = 3, c_3 = 6$$

Les bandes homogènes horizontales sont : $R_{1,2}(7), R_{2,4}(7), R_{3,3}(7)$ de valeur respectivement :

$$\lambda_1 = 5, \lambda_2 = 12, \lambda_3 = 18$$

La solution du problème de sac *à* dos :

$$\begin{cases} F_{hor} = \max 5.x_1 + 12.x_2 + 18.x_3 \\ s.c : x_1 + 2.x_2 + 3.x_3 \le 5 \\ x_1, x_2, x_3 \in N \end{cases}$$

est M_{hor} *=(0,1,1)* de valeur F_{hor} *=30.*

Les bandes homogènes verticales sont : $R_{1,1}(5), R_{2,3}(5), R_{3,2}(5)$ de valeur respectivement :

$$\xi_1 = 5/2, \xi_2 = 9, \xi_3 = 12$$

La solution du problème de sac *à* dos :

$$\begin{cases} F_{ver} = \max 5/2.y_1 + 9.y_2 + 12.y_3 \\ s.c : y_1 + 2.y_2 + 3.x_3 \le 7 \\ y_1, y_2, y_3 \in N \end{cases}$$

est M_{ver} = *(0,2,1)* de valeur F_{ver} *=30.* Il y a dans ce cas deux solutions *é*quivalentes :

$$M_{hor} = (0,1,1)$$
$$M_{ver} = (0,2,1)$$

qui désigne respectivement: le modèle de découpe horizontal composé de la bande $R_{2,4}(7)$ et de la bande $R_{3,3}(7)$ et du modèle de découpe vertical composé de deux bandes du type $R_{2,3}(5)$ et d'une bande $R_{3,2}(5)$.

6.5. Conclusion

Nous avons présenté le contexte général du problème de découpe de pièces de formes trapézoïdales sur un support rectangulaire. Nous avons étudié les formes de rotations permises dans la disposition des pièces à découper. Par ailleurs, nous avons établi les propriétés et les caractéristiques des bandes homogènes optimales. Enfin, nous avons développé une nouvelle approche de résolution du problème de découpe de trapèzes basée sur une procédure constructive permettant d'obtenir des modèles de découpe non orthogonales sur l'entité initiale au moyen de constructions horizontales et verticales qui génèrent les bandes homogènes optimales. La sélection des meilleures bandes homogènes conduit à la construction du meilleur plan de découpe généré à partir des combinaisons des éléments de l'ensemble des bandes homogènes horizontales et verticales. Nous avons démontré que l'algorithme possède un rapport d'approximation constant.

Conclusion générale et perspective de recherche

Les approches que nous avons été amenés à développer pour la résolution d'une classe difficile de problèmes de l'optimisation combinatoire, en l'occurrence les problèmes de découpe, utilise une multitude de variétés de techniques performantes issues aussi bien de la recherche opérationnelle (recherche arborescente, programmation dynamique, méthodes heuristiques, etc.) que d'autres branches telle que les méthodes d'intelligence artificielle (algorithme A*, stratégies de Hill climbing).

Le schéma général que nous avons utilisé dans l'élaboration des méthodes exactes et approchées consiste en une structure d'arborescence représentant l'espace de solutions du problème considéré. L'exploration de l'arborescence s'effectue à travers un parcours transversal d'un graphe orienté dans lequel chaque sommet représente un sous problème du problème traité, et chaque arc décrit une relation directe entre sommets. Le parcours de l'arborescence nécessite la spécification du sommet initial (racine), des sommets solutions (pendants), ainsi que les règles de mouvements (cheminements) permis. Pour le problème de découpe, le rectangle initial est représenté par le sommet racine de l'arborescence, et les sous-entités sont représentés par les sommets pendants et les règles qui définissent les cheminements permis sont les découpes possibles sur le(s) rectangle(s) : découpes guillotine, découpe artificielle (qui garde le rectangle intact), les procédés de discretisation des points de découpes et les effets de symétrie afin d'éliminer les formes dupliquées des pièces produites, ainsi que les dissections inutiles.

Une recherche de solution efficace via l'exploration d'une arborescence impose l'utilisation de méthodes d'énumération implicites afin de limiter au maximum l'explosion combinatoire de celle ci, ces méthodes s'appuient sur des procédures à qui appartient la décision des règles de mouvement, c 'est à dire à quel moment et quelle opération doit être appliquée sur les sommets disponibles de l'arborescence, des résultats de ces règles intelligentes découle irrémédiablement les performances d'optimalité des algorithmes développés.

C'est ce raisonnement qui a constitué l'idée principale sur laquelle nous avons conçu les approches algorithmiques pour une classe de problèmes de découpe auxquels nous sommes intéressés dans cette thèse.

Dans [41] nous avons proposé une méthode exacte ainsi qu'une méthode spécifique pour la résolution du problème de découpe sans contraintes à deux dimensions pour les versions pondéré et non pondéré. La méthode exacte que nous avons développée est une combinaison entre une méthode de séparation et d'évaluation et une recherche locale. Le principe de la méthode consiste à limiter, d'une part la taille de l'arborescence et d'autre part, à accélérer le parcours de celle ci en se limitant à une procédure de recherche locale pour le calcul de l'évaluation sur un sommet. Par ailleurs, nous essayons d'exploiter au maximum l'information résultant de chaque calcul en un sommet, dans la génération des bandes horizontales et verticales à travers la résolution par la programmation dynamique des problèmes de sac à dos correspondants. La méthode approchée s'appuie principalement sur les points suivants :

- Heuristique de calcul d'une solution initiale.
- Réduction des ensembles des combinaisons linéaires (respectivement sur les longueurs et sur les hauteurs des pièces à découper).
- Limitation de la profondeur de chaque arborescence (une arborescence correspond à une succession de découpes sur le support et les rectangles générés).
- Application d'une recherche locale pour le calcul des bornes inférieures et supérieures sur les sommets.

En procédant de la sorte, il est apparu que l'on pouvait réduire de manière considérable le nombre de sommets à parcourir. De plus, la qualité des solutions obtenues est très proche de l'optimum. Notre approche à été comparée aux deux meilleures méthodes de la littérature (Beasley [8] et Morabito and al.[53]) sur des instances de la littérature et d'autres générées aléatoirement, nous avons enregistré des résultats de meilleure qualité et un temps d'exécution inférieur.

Pour la résolution du problème de découpe généralisé sans contraintes, nous avons proposés

1. Une méthode exacte [41] sous forme d'un algorithme séquentiel, basé sur le principe suivant :

Construire une procédure séquentielle capable de :

- Détecter quelques supports rectangulaires qui peuvent être traités simultanément.
- Négliger quelques régions de l'espace de recherche qui n'affectent en aucun cas la solution optimale du problème.

L'approche que nous avons proposée s'appuie sur quelques résultats théoriques que nous avons démontrés. Au début nous donnons la définition d'un support rectangulaire non-dominant. En utilisant cette définition, nous avons démontré que la présence de tels supports peut être omise sans affecter la solution optimale du problème. Par la suite nous avons démontré que si toutes les longueurs des supports respectaient un ordre croissant, alors nécessairement toutes les hauteurs respectaient l'ordre inverse. Finalement, en utilisant l'ordre sur les longueurs, nous avons démontré la convergence de l'algorithme.

Notre approche à été comparée à une adaptation de l'algorithme de Beasley [8], sur un ensemble d'instances générées aléatoirement (500 instances). Sur l'ensemble des instances, notre algorithme était sensiblement plus rapide en moyenne.

2. Une méthode spécifique basée sur le même principe du plus rectangle fictif. Au départ, nous utilisons une réduction du problème initial sous forme d'une série de problème de sac à dos unidimensionnels. Ensuite, nous effectuons un tri par ordre croissant sur les longueurs des supports rectangulaires. Finalement, nous appliquons une procédure basée sur la programmation dynamique pour générer un ensemble de bandes horizontales. Cet ensemble est généré progressivement, selon l'ordre initialement appliqué sur les longueurs des supports. Par la suite, nous utilisons le même principe pour la génération progressive des bandes verticales. Finalement, la solution approchée de chaque support est représentée par la combinaison de quelques bandes horizontales (ou verticales) disponible de dimensions inférieures ou égales à celle du support considéré.

Nous avons, démontré que lorsque les problèmes de sac à dos unidimensionnels utilisés sont résolus par l'utilisation d'une procédure gloutonne, l'algorithme garantissait un rapport d'approximation polynomial constant.

Nous avons effectué une série de tests numériques afin d'analyser le comportement de cette approche sur différentes instances générées aléatoirement Nous avons également testé la performance de cette approche en la comparant à l'algorithme exact. Sur l'ensemble des instances traitées, l'approche heuristique donnait des résultats satisfaisants. A titre d'exemple sur un groupe d'instance de taille moyenne, l'approche heuristique réalisait un rapport d'approximation expérimental proche de 1 en réduisant de manière significative le temps de calcul requis.

Nous avons également étudié le problème d'assemblage [42] qui consiste à regrouper un nombre de pièces rectangulaires dans la plus petite surface rectangulaire possible. Nous proposons un algorithme exact et une heuristique générale. Ces algorithmes sont basés principalement sur l'algorithme A* et les techniques de la programmation dynamique pour résoudre le problème d'assemblage orthogonal et le problème d'assemblage sur bande. Nous proposons plusieurs modifications sur l'algorithme original A* en introduisant des bornes inférieures au niveau de l'arborescence et d'une stratégie de coupe – branche, qui permet de stopper le processus de développement de l'algorithme. Par ailleurs, nous introduisons une borne supérieure obtenue par l'application d'une heuristique gloutonne pour le problème d'assemblage rectangulaire et d'une autre heuristique se basant sur le problème de sac à dos borné pour le problème d'assemblage sur bande [43]. Les résultats théoriques présentés justifient les choix effectués sur les évaluations. La performance de nos méthodes est évaluée sur un nombre d'instances générées aléatoirement. Les tests numériques montrent que les approches développées sont capables de résoudre des instances de petites et moyenne taille en un temps de d'exécution raisonnable.

Nous avons aussi traité un problème particulier appartenant à la classe des problèmes de découpe non orthogonale [45], il s'agit du problème de découpe de pièces trapézoïdales, qui consiste à rechercher le meilleur modèle de découpe du maximum de pièces trapézoïdale de forme fixée sur un support rectangulaire, avec le minimum de chute. C'est un problème pour

lequel, il n'existe aucune approche de résolution. A cet effet, nous avons introduit un ensemble de concepts et de notions qui définissent une paramétrisation complète des pièces à découper et de détecter les formes de pièces dupliquées engendrées par les orientations permises sur les pièces (rotation et translation). Nous avons développé une méthode de résolution basée sur une procédure constructive permettant d'obtenir un modèle de découpe non orthogonale sur l'entité initiale au moyen de constructions horizontales et verticales. Nous avons démontré que cette approche possède un rapport d'approximation constant supérieure à 33%.

Compte tenu de la diversité des problèmes de découpe, il nous est impossible d'énumérer toutes les variantes que nous souhaiterons traiter. Nous citerons un certain nombre de problèmes et les objectifs que nous voudrions atteindre les concernant pour nos travaux de recherche future.

En premier lieu, Nous espérons améliorer l'algorithme exact conçu pour le problème de découpe sans contraintes, par la suite nous essayerons de voir si la méthode peut s'adapter au problème de découpe sur bande.

De plus, nous espérons que l'utilisation de machines parallèles nous permettra d'augmenter la tailles et le temps d'exécution des instances à traiter.

Par ailleurs l'introduction d'une contrainte supplémentaire appelée contrainte de niveau engendre des limitations sur le nombre de dissections horizontales et/ou verticales permises. Notre souhait est de concevoir des approches de résolution pour ces types de problèmes.

Un autre problème qui nous semble intéressant, est le problème de découpe généralisé avec contraintes. Une première direction consiste à construire un algorithme constructif basé sur des constructions horizontales et verticales en parallèles, adaptées par la suite aux différents supports disponibles.

Finalement, nous signalerons les classes de problèmes de découpes pour lesquels peu d'approches ont été conçues pour leur résolution, c'est le cas des problèmes de découpe non guillotine et des problèmes de découpe à trois dimensions [52].

Références bibliographiques

[1] M. Adamowicz and A.Albano, Two-stage solution of the cutting stock problem, Information Procc. North Holland, vol. 71, p.p. 1086-1091, 1972.

[2] R. Andonov, F. Raimbault and P. Quinton, dynamic programming parallel implementations for the knapsack problem, Preprint IRISA, No 740, July 1993, Campus de Beaulieu, 35042 Rennes cedex .

[3] M. Arenales and R. Morabito, An and/or- graph approach to the solution of two dimensional non guillotine cutting problems, European Journal of Operational Research, vol. 84, p.p. 599-617, 1995.

[4] B.S. Baker, E.G. Ciffman, and R.L. Rives, Orthogonal packing in two dimensions, Siam Jour. Comput, vol.9, No 4, p.p. 846-855, 1980.

[5] B.S Baker, D.J. Brown and H.P. Katseff, A 5/4 algorithm for two-dimensional packing, Journal of Algorithms, vol. 2, No 4, p.p. 348-368, 1981.

[6] J.P. Barthelemy, G. Cohen and A. Lobestein, Complexité algorithmique et problèmes de communications, Masson, Collection CNET-ENST, Paris, 1992.

[7] J. E. Beasley, An exact two-dimensional non-guillotine cutting tree search procedure, Operations Research, vol. 33, p.p. 49-64, 1985 .

[8] J. E. Beasley, Algorithms for unconstrained two-dimensional guillotine cutting , Journal of the Operational Research Society, vol. 36, p.p. 297-306, 1985 .

[9] M. Biro and E. Biros, Network flows and non guillotine cutting tree search procedure, European Journal of Operational Research, vol. 16, p.p. 297-306, 1985.

[10] G. Chen, M. Chern and J. Jang, Pipeline architectures for dynamic programming algorithms, Parallel Computing, vol. 13, p.p. 111-117, 1990.

[11] N. Christofides and C. Whitlock, An algorithm for two-dimensional cutting problems, Operations Research, vol. 2, p.p. 31-44, 1977.

[12] C.G. Codd, Multiprogram sheduling, Comm. of the ACM, vol. 3, N°6, p.p. 347-350 and N° 7, p.p. 413-418, 1960.

[13] S.A. Cook, The complexity theorem providing procedure, Proc. 3rd ACM Symp. On theory of computing, p.p. 151-158, 1971.

[14] J. Daniels and P. Ghandforoush, An improved algorithm for the non-guillotine constrained cutting-stock problem, Journal of operation research society, vol. 41, N°2, p.p. 141-150, 1990.

[15] R. Dechter and J.Pearl, The optimality of A*, search in Artificial Intelligence, Spinger, 1988.

[16] K. Dowsland and W. Dowsland, Solution approaches to irregular nesting problems, European Journal of Operational Research, vol. 84, p.p. 506-521, 1995.

[17] G. Dueck and T. Scheurer, Threshold accepting: A general purpose optimisation algorithm appearing superior to simulated annealing, Journal of computational physics, vol. 90, p.p. 161-175, 1990.

[18] R.G. Dyson and A.S. Gregory, The cutting stock problem in the flat glass industry, Operation Research Quart., vol. 25, p.p. 41-53, 1974.

[19] Z. Drezner, DISCON: A new method for the layout problem, Operations Research, vol. 25, No 6, p.p. 1375-1384, 1980.

[20] Z. Drezner, A heuristic procedure for the layout of a large number facilities, Management Science, vol. 33, No 7, p.p. 907-915, 1987.

[21] A. J. W. Duijvestijn, Simple perfect squared square of lowest order, Journal of Combinatorial Theory, vol. 25 (B), p.p. 260-263, 1978.

[22] J. Edmond, Paths, trees and flows, Canad. Journal. Math, vol. 17, p.p. 449-467, 1965.

[23] D. Fayard and V. Zissimopoulos, An approximation algorithm for solving unconstrained two-dimensional knapsack problems, European Journal of Operational Research, vol. 84, p.p. 618-632, 1995.

[24] P. J. Federico, J.A. Bondy, and U.S.R. Murty, Squaring rectangles and squares, Graph theory and related topics, Editions Academic Press, 1979.

[25] T.A. Feo and M.G.C. Resende, A probabilistic heuristic for computationally difficult set covering problem, Operational Research Letters, vol. 8, p.p. 67-71, 1989.

[26] W. Fernandez de la Vega, and G.S Lueker, Bin packing can be solved within 1+ε in linear time, Combinatoria, vol. 1, No 4, p.p. 349-355, 1981.

[27] R.J. Fowler, R.M. Paterson, and S.T. Tanimoto, Optimal packing and covering in the plane are NP-complete, Information Processing Letters, 12, p.p. 133-137, 1981.

[28] M.R. Garey and R.L. Graham, Bounds on multiprocessing scheduling with resource constraints, SIAM, Journal on computing, vol. 4, N°2, p.p. 187-200, 1975.

[29] M.R. Garey and D.S. Johnson, Computers and intractability. A guide to the theory of NP-completeness, W.H. Freeman and Company, San Francisco, 1979.

[30] P. Gilmore, Cutting stock, linear programming, knapsacking, dynamic programming and integer programming, some interconnections, Annals of Discrete Mathematics, vol. 4, p.p. 217-235, 1979.

[31] P. Gilmore and R. Gomory, Multistage cutting problems of two and more dimensions, Operations Research, vol. 13, p.p. 94-119, 1965.

[31’] P. Gilmore and R. Gomory. A linear programming approach to the cutting stock problem. Operations Research, 9:848--859, 1961.

[31’’] P. Gilmore and R. Gomory, "A linear programming approach to the cutting stock problem - part II," Operations Research, vol. 11, 1963.

[32] P. Gilmore and R. Gomory, The theory and computation of knapsack functions, Operations Research, vol. 14, p.p. 1045-1074, 1966.

[33] F. Glover, The future paths for integer programming and links to artificial intelligence, Computers and operations research, vol. 13, N°5, p.p. 533-549, 1986.

[34] F. Glover, and H. J. Greenberg, New approaches for heuristic search: A bilateral linkage with artificial intelligence, European Journal of Operational Research, vol. 39, p.p. 119-130, 1989.

[35] E. Hadjiconstantinou and N. Christofides, An optimal algorithm for general orthogonal 2-D cutting problems, Technical Reports MS-91/2, Imperial College, London, 1991.

[36] S.G. Hahn, On the optimal cutting of defective sheets, Operations Research, vol. 16, p.p.1100-1114, 1968.

[37] R.W. Haessler, Controlling cutting changes in one dimensional trim problem, Operations Research, vol. 23, p.p.483-493, 1975.

[38] S.S. Heragu, and A. Kusiak, Machine layout problem in flexible manufacturing systems, Operations Research, vol. 36, No 2, p.p. 258-268, 1988.

[39] S.S. Heragu, and A. Kusiak, Efficient models for the facility layout problem, European Journal of Operational Research, vol. 53, p.p. 1-13, 1991.

[40] J. C. Herz, A recursive computing procedure for two-dimensional stock cutting, IBM Journal of Research and Development, vol. 16, p.p. 462-469, 1972 .

[41] M. Hifi and R. Ouafi, Best first search and dynamic programming methods for cutting problems: The cases of one or more stock plates, Computers and industrial Engineering, vol. 32, N°1, p.p. 187-205, 1997.

[42] M. Hifi and R. Ouafi, A best first branch-and-bound algorithm for orthogonal rectangular packing problems, International Transactions in Operational Research, vol. 5, N°5, p.p. 345-356, 1998.

[43] M. Hifi, A. Khelladi and R. Ouafi, The A* algorithm for orthogonal rectangular packing problems and strip packing problems. Proceeding of the fifth International Symposium on Programming System p.p. 253-263, 2001.

[44] M. Hifi, Contribution à la résolution de quelques problèmes difficiles de l'optimisation combinatoire, Thèse d'habilitation à diriger des recherches, PRiSM, Université de Versailles St.-Quentin en Yvelines, 1999.

[45] A. Khelladi, R. Ouafi, Problème de découpe non orthogonale: Cas de pièces trapézoïdales, à paraître dans Revue Maghrébine de Mathématiques.

[46] R. M. Jorgensen, R. M. Thomson, and R. V. Vidal, The afforestation problem: A heuristic method based on simulated annealing, European Journal of Operational Research, vol. 56, p.p. 184-191, 1992.

[47] L. K. Kantorovich, Mathematical methods of organising and planning production, Management Science, vol. 6, p.p. 363-422, 1960.

[48] R.M. Karp, Reducibility among combinatorial problems, in R.E. Miller, J.W. Thatcher, Complexity of computer computation, Press, p.p. 85-103, 1972.

[49] T. C. Koopmans, T. C. and Beckmann, M. Assignment problems and the location of economic activities, Econometrica, vol. 25, No 1, p.p. 53-76, 1957.

[50] R. F. Love, and R. F. Wong, On solving one-dimensional space allocation problem with integer programming, Information Processing and Operations Research (INFOR), vol. 14, No 2, p.p. 139-143, 1976.

[51] O.B.G. Madsen, References concerning the cutting stock problem, IMSOR, The Technical University of Denmark, DK-2800, Lyngby, 1980.

[52] S. Martello, D. Pisinger and D. Vigo, The three dimensional bin packing problem, Operation Research, 48, p.p. 256-267, 2000.

[53] R. Morabito, M. Arenales and V. Arcaro, An and/or-graph approach for two-dimensional cutting problems, European Journal of Operational Research, vol. 58, No 2, p.p. 263-271, 1992.

[54] H. Muller, Modelling techniques and heuristics for combinatorial problems, in B. Roy, Combinatorial programming: Methods and applications, Proceeding of the NATO Advanced Study institute, p.p. 1-26, 1974.

[55] J. Pearl, Heuristics : Intelligent search strategies for computer problem solving, Addison-Wesley, Reading, MA, 1984 .

[56] A.H.G. Rinnoy Kan, J.R. De Wit and R.T. Wijmenga, Non-orthogonal two-dimensional cutting patterns, Management Science, vol. 33, p.p. 670-684, 1987.

[57] M. Sakarovitch, Optimisation combinatoire, programmation discrète, Hermann Editeurs des sciences et des Arts, 1984.

[58] M. Syslo, N. Deo, and J. Kowalik, Discrete optimisation algorithms, Prentice-hall, New Jersey, 1983.

[59] P. Toth, Dynamic programming algorithms for zero-one knapsack problem, Computing, vol. 25, p.p. 29-45, 1980.

[60] F. J. Vasko, A computational improvement to Wang's two-dimensional cutting stock algorithm, computers and industrial Engineering, vol. 16, p.p. 109-115, 1989.

[61] K. V. Viswanathan, and A. Bagchi, Best-first search methods for constrained two-dimensional cutting stock problems, Operations Research, vol. 41, No 4, p.p. 768-776, 1993.

[62] P. Y. Wang, Two algorithms for constrained two-dimensional cutting stock problems, Operations Research, vol. 31, No 3, p.p. 573-586, 1983.

[63] V. Zissimopoulos, Heuristic methods for solving (un) constrained two-dimensional cutting stock problems, Methods of Operations Research, vol. 49, p.p. 345-357, 1984.