Les problèmes de placement : étude et résolution de quelques problèmes réels

Julien Antonio

N° d'ordre :

THESE

présentée à

L'UNIVERSITE DE METZ
FACULTE DES SCIENCES
U.F.R. Mathémathique, Mécanique et Automatique

pour obtenir le titre de

DOCTEUR

Spécialité

SCIENCES DE L'INGENIEUR

par

Julien ANTONIO

Sujet de la thèse :

LES PROBLEMES DE PLACEMENT :
ETUDE ET RESOLUTION DE QUELQUES PROBLEMES REELS

Soutenue le 14 février 1997 devant le Jury composé de :

M.	**Bernard MUTEL**	Rapporteurs
Mle	**Marie-Claude PORTMANN**	
M.	**Chengbin CHU**	Examinateurs
M.	**Patrick HIRSCH**	
M.	**Jean-Marie PROTH**	
Mle	**Nathalie SAUER**	

N° d'ordre :

THESE

présentée à

L'UNIVERSITE DE METZ
FACULTE DES SCIENCES
U.F.R. Mathémathique, Mécanique et Automatique

pour obtenir le titre de

DOCTEUR

Spécialité

SCIENCES DE L'INGENIEUR

par

Julien ANTONIO

Sujet de la thèse :

LES PROBLEMES DE PLACEMENT :
ETUDE ET RESOLUTION DE QUELQUES PROBLEMES REELS

Soutenue le 14 février 1997 devant le Jury composé de :

M.	**Bernard MUTEL**	Rapporteurs
Mle	**Marie-Claude PORTMANN**	
M.	**Chengbin CHU**	Examinateurs
M.	**Patrick HIRSCH**	
M.	**Jean-Marie PROTH**	
Mle	**Nathalie SAUER**	

AVANT-PROPOS

Cette thèse a été réalisée au sein du proget SAGEP de l'INRIA-Lorraine sous la direction scientifique de Monsieur Jean-Marie PROTH, Directeur de Recherche à l'INRIA. Je tiens à lui exprimer toute ma reconnaissance pour m'avoir permis de faire cette recherche dans son équipe, pour la confiance qu'il m'a témoignée, pour son encadrement efficace et pour le temps passé à l'amélioration de ce document. Je le remercie aujourd'hui de me faire l'honneur de participer au jury.

Je voudrais remercier Monsieur Patrick HIRSCH, Président Directeur Général du Groupe Jean d'HUART & Cie, et tout son personnel pour les facilités qu'ils m'ont données afin de mener à bien cette thèse. Je suis particulièrement honnoré de sa présence dans le jury.

Je remercie vivement Mademoiselle Marie-Claude PORTMANN, Professeur à l'Ecole des Mines de Nancy d'avoir accepté la charge d'évaluer ce travail en qualité de rapporteur et de membre du jury.

Je remercie tout particulièrement Monsieur Bernard MUTEL, Professeur à l'Ecole Nationale Supérieure des Arts et de l'Industrie de Strasbourg, d'avoir accepté d'être rapporteur et d'être membre du jury de cette thèse.

Je désire exprimer ma plus profonde gratitude à Monsieur Chengbin CHU, Professeur à l'Université de Technologie de Troyes, pour l'aide qu'il m'a apportée par sa grande expérience en recherche opérationnelle et dont les conseils, les encouragements et le suivi ont été déterminants dans l'élaboration de ce travail. Je suis sensible à l'honneur qu'il me fait de participer au jury.

J'adresse mes plus vifs remerciements à Mademoiselle Nathalie SAUER, Maître de Conférence à l'Ecole des Mines de Nantes, pour sa constante bienveillance, son soutien et sa présence à ce jury.

Enfin, que les nombreuses personnes qui m'ont accompagnées tout au long de cette thèse, tant au point de vue personnel que professionnel, se reconnaissent et reçoivent toute ma sympathie.

A ma famille

A mes amis

TABLE DES MATIÈRES

Chapitre I :

INTRODUCTION AUX PROBLÈMES DE PLACEMENT

CHAPITRE I

Un problème de placement ou de découpe (ou *"bin packing"* dans la littérature anglo-saxonne) consiste à localiser des entités appelées *"entités filles"* sur un ensemble d'entités de taille supérieure dites *"entités mères"* de façon à optimiser un critère donné.

Les applications sont diverses et peuvent intégrer des contraintes multiples comme nous le verrons par la suite. En conséquence, il n'est pas possible de formaliser le problème d'une manière unique. C'est pourquoi les recherches dans ce domaine se sont focalisées sur des problèmes spécifiques que nous détaillerons.

Dans cette introduction, nous allons en premier lieu fournir une typologie des problèmes de découpe. Puis nous verrons les différentes façons d'effectuer les comparaisons entre les solutions. Enfin, nous présenterons les objectifs et l'organisation de cette thèse.

1. DIVERSITÉ DES PROBLÈMES DE DÉCOUPE

Les industries sont très souvent confrontées aux "problèmes de découpe". Leur objectif est de réduire les chutes dans les découpes à une ou deux dimensions (cas $\Re^1$ et $\Re^2$), ou de remplir au mieux des containers avec des objets (cas $\Re^3$). Dans ce dernier cas, les études réalisées concernent le plus souvent des objets parallélépipèdiques. L'enjeu économique des problèmes de découpe (ou de placement) est certain. Le gain sur la chute générée, ou sur le temps nécessaire pour réaliser cette découpe, se répercute sur le prix de revient du produit fabriqué.

1.1. Classifications

1.1.1. Classification de Dyckhoff

Dyckhoff [DYC 23] propose une classification des problèmes de découpe. Cette classification fait intervenir les paramètres suivants :

1. les dimensions :
   - **1,2,3** ou **N**-dimensions avec N>3 ;
2. le type d'allocation :
   - **B** *(Beladeproblem)* : placer un nombre donné d'objets dans un nombre non limité de boîtes ;
   - **V** *(Verladeproblem)* : remplir un nombre donné de boîtes avec un nombre non limité d'objets ;
3. les types de boîtes :
   - **O** *(One)* : une boîte ;
   - **I** *(Identical)* : des boîtes de forme identique ;
   - **D** *(Different)* : des boîtes de formes différentes ;

4. les types d'objets :
   - **F** *(Few)* : peu d'objets de formes différentes ;
   - **M** *(Many)* : beaucoup d'objets avec beaucoup de formes différentes ;
   - **R** *(Relatively)* : beaucoup d'objets avec peu de formes différentes ;
   - **C** *(Congruent)* : objets de forme similaire.

Cette classification est plus fine que le classement par dimension uniquement. Le nombre de combinaisons possibles est alors de $4 \times 2 \times 3 \times 4 = 96$. On peut la retrouver dans le recueil bibliographique [DYC 24]. Cette classification permet de cerner avec précision le domaine intéressant le lecteur. Le recueil lui fournit de façon exhaustive les références des publications propres à son problème. Cette précision est utile pour situer le problème et éclairer le chercheur en recherche opérationnelle sur les antécédents de ce type de problèmes.

Trop contraignante, cette classification est peu utilisée dans la littérature. Généralement, les problèmes exposés dans la littérature concernent des applications : ils sont donc définis de façon spécifique (découpe de temps, de moquettes, de plaques de verre, de tissus...). Elles ont néanmoins un point commun simple : les dimensions.

1.1.2. Classification courante

La classification communément utilisée pour les problèmes de placement est basée sur la dimension de l'espace dans lequel on cherche à les résoudre. De plus, nous pouvons distinguer deux types de problèmes de découpe :

1. les découpes ***"en-ligne"*** pour lesquelles les éléments à découper sont placés suivant leur ordre d'arrivée sans tenir compte des éléments futurs à découper (des études statistiques sont parfois mises en œuvre pour anticiper les commandes futures) ;
2. les découpes ***"hors-ligne"*** pour lesquelles nous connaissons tous les éléments à découper.

Dans notre étude, nous considérons uniquement le deuxième type de découpe. Il faut remarquer que les méthodes utilisées pour résoudre les problèmes de découpe "en ligne" sont issues des techniques permettant d'appréhender les problèmes de découpe "hors ligne".

1.2. Types de problèmes de découpe

Les problèmes de découpe peuvent concerner différents domaines. Nous en donnons quelques exemples. Les caractéristiques de ces problèmes sont résumées dans le tableau I.1.

a. La ***découpe proprement dite*** *(cutting, trim-loss)* [BIS 7], [MOR 43], [COS 14] et [COS 15]

L'objectif est d'utiliser une quantité minimale de matière : barres, surfaces (découpes de tubes, de bobines, de planches de bois, de verre, de papier...) pour réaliser un ensemble donné d'éléments.

Les découpes de bobines peuvent revêtir un caractère unidimensionnel lorsqu'elles sont découpées :

- dans toute leur ***longueur*** [GOU 36], [MOR 43], [COS 13] et [COS 14]
  Les bobines (de papier) mères sont découpées sur toute leur longueur en bobineaux de plus petites largeurs et de même longueur. La chute de matière est la bande perdue après l'obtention des bobineaux.
- dans toute leur *largeur* [COS 13] et [COS 14]
  Des rectangles sont découpés dans des rouleaux de moquettes. La largeur des rectangles est celle du rouleau. La chute est obtenue en bout de rouleau.

b. Le ***placement de tâches dans le temps*** [COS 14] et [BAB 4]

Il s'agit de générer un emploi du temps qui peut être, par exemple, l'organisation de la semaine d'un artisan. Les tâches sont de valeurs et de durées variées.

c. L'***utilisation de surfaces*** (pallet loading problem)

Des palettes [DOW 19] de tailles et de géométries variées sont à disposer sur une aire. L'objectif est alors de limiter les surfaces inutilisées. Par simplification, les palettes sont généralement considérées comme rectangulaires. Dans le cas des cellules électroniques [DAR 16], il faut aussi faire intervenir le coût des liaisons entre les cellules. Ce coût augmente lorsque deux cellules qui doivent être connectées s'éloignent l'une de l'autre.

d. Le ***remplissage de volumes*** [COS 14]

Des articles de volumes connus sont à placer dans des boîtes identiques de volume plus important. L'objectif est de réduire l'espace non utilisé dans les boîtes. Le problème dans $\Re^3$ se réduit à un problème dans $\Re^1$ lorsque les articles et les boîtes ont des bases identiques.

Un problème similaire se pose lors du remplissage des fours de recuit dans la sidérurgie, où les pièces sont empilées les unes au-dessus des autres. Il faut alors minimiser l'espace perdu entre la dernière pièce et le haut du four [COS 13].

Notons que dans les problèmes de remplissage, les entités peuvent être en contact, contrairement aux problèmes de découpe dans $\Re^1$ et $\Re^2$ où un espace entre les entités doit être laissé pour le "trait de scie" (cf. chapitre II, section 2.1).

e. La ***maximisation du poids transporté*** [COS 13] et [COS 14]

Il s'agit là aussi d'un problème de remplissage. Nous sommes cette fois-ci limité par la charge des containers qu'il faut remplir avec des articles de poids donné. Ceci est fréquent dans les problèmes de logistique où le coût de transport d'un container est fixe.

f. Le ***remplissage de rayons*** [COS 13] et [COS 14]

Les bibliothèques, les grandes surfaces, les entrepôts, doivent disposer sur des étagères un nombre important de produits. Ceci doit s'effectuer sans chevauchement et avec le souci de limiter l'espace inutilisé en bout de rayon.

Types de problèmes	Dimensions	Critères	Domaines d'application
Découpes proprement dite	$\Re 2$ $\Re 1$	*Minimiser :* - les chutes - la matière utilisée	Découpes de pièces dans - l'aéronavale - la confection - l'ébénisterie - la maroquinerie - la papeterie - la sidérurgie - la verrerie...
Placement de tâches dans le temps	$\Re 1$	*Maximiser :* - la rentabilité, *Optimiser :* - l'utilisation des ressources.	Emploi du temps - d'un artisan - d'une école - d'un atelier...
Utilisation de surfaces	$\Re 2$ $\Re 1$	*Minimiser :* - les surfaces inutilisées	- Étalages en grande surface - Machines outils dans un atelier - Puces sur une tranche de silicium - Composants électroniques sur une plaque époxite - Palettes ou étagères dans un entrepôt - Palettes sur une machine automatique...
Remplissage de volumes	$\Re 3$ $\Re 2$ $\Re 1$	*Minimiser :* - l'espace inutilisé - les déplacements pour atteindre les colis	- Objets dans un entrepôt - Caisses dans un wagon, un camion, un cargo - Colis dans une caisse - Pièces dans un four en sidérurgie...
Maximisation du poids transporté	$\Re 1$	*Maximiser :* - le poids embarqué	Remplissage de cargos ou camions par des objets lourds (le critère volume est secondaire)
Remplissage de rayons	$\Re 1$	*Maximiser :* - l'utilisation des rayons	- Grandes surfaces, - Bibliothèques - Entrepôts...
Gestion de mémoire par zones	$\Re 1$	*Minimiser :* - les zones inaccessibles	- Informatique
Gestion de l'unité centrale	$\Re 1$	*Minimiser :* - le temps de traitement total (makespan)	- Informatique
Choix des investissements	$\Re 1$	*Maximiser :* - les bénéfices	Investissements des - banques - sociétés - portefeuille des particuliers...

Tableau I.1 : Caractéristiques des problèmes de placement

g. La ***gestion de mémoire par zones*** [COS 14]

Il faut répartir les programmes entre différentes zones de mémoire disponibles dans un ordinateur de façon à optimiser l'utilisation de la mémoire.

h. La ***gestion de l'unité centrale*** d'un ordinateur utilisé en temps partagé

Il s'agit de minimiser le temps total de traitement lorsqu'il faut affecter à un nombre donné de processeurs, soit une liste de tâches non-interruptibles, sans ordre entre les tâches (gestion du multitâche [DAR 16]), soit le déroulement d'un programme parallélisable (gestion de processeurs [COS 13] et [COS 14]).

i. Le ***choix des investissements (capital budgeting problem)*** [DYC 23]

Le problème est de sélectionner, à partir d'un ensemble de projets d'investissement possibles, un sous-ensemble qui maximise un profit ou une satisfaction (bénéfices actualisés par exemple) sous certaines limitations budgétaires, humaines, matérielles, etc.

2. DIFFICULTÉS RENCONTRÉES DANS LES PROBLÈMES DE DÉCOUPE

La résolution des problèmes de découpe passe par deux grandes étapes. Il faut d'abord poser le problème, puis le résoudre.

2.1. Définition des problèmes

2.1.1. Positionnement du problème

La découpe de matière n'est, le plus souvent, qu'une partie du problème posé. S'y ajoutent plusieurs autres problèmes :

a. le problème de **disposition des produits** sur les aires de stockage. Ce problème d'agencement des stocks est un problème statistique dont l'étude se basera sur les historiques d'utilisation des stocks ;

b. le problème de **décisions de découpe** qui consiste, connaissant les commandes à réaliser (en fonction des dates de livraison chez les clients), les stocks (disponibles et approvisionnements futurs), et les moyens disponibles en machines, en ouvriers et en moyens de transport, à décider de la manière dont la découpe doit être effectuée de façon à minimiser un coût donné ;

c. le problème de **gestion de commandes** qui consiste à décider de la manière dont l'activité (b) sera appliquée (décision de découpe par commande, décision de découpe sur des regroupements de commandes...) ;

d. le problème de **gestion des découpes**, i.e. de regroupement de découpe de façon à augmenter la productivité ; notons que ce problème est très lié au problème précédent (gestion des commandes) ;

e. enfin, le problème de **gestion des réapprovisionnements** en tenant compte, bien entendu, des demandes et des stocks, mais aussi des contraintes des fournisseurs.

Nous voyons qu'il est ardu de vouloir modéliser l'ensemble des éléments intervenant dans l'économie d'une entreprise afin d'optimiser globalement son coût. Il est en effet illusoire de prendre en compte en même temps tous les éléments intervenant dans le coût final du fonctionnement de l'entreprise. Le décideur doit donc cerner un ou plusieurs problèmes à résoudre et comme le fait remarquer Faure [FAU 26] : "*la définition des objectifs et la détermination du critère d'optimisation sont du ressort de l'entrepreneur* ".

Deux remarques doivent être faites sur les problèmes élémentaires de l'entreprise :

- le problème b est ***indépendant*** des problèmes a, c, d et e quelle que soit la manière de les aborder ;
- les problèmes a, c, d et e sont ***bien résolus*** statistiquement. Le problème b est un problème combinatoire et il n'existe pas de méthode générale pour l'appréhender.

Nous avons décidé de nous limiter à la résolution des problèmes de décision de découpe.

2.1.2. Critères

Des critères (ou fonctions économiques) sont utilisés pour évaluer la qualité du placement réalisé. On peut soit maximiser le profit ou la satisfaction de l'entreprise, soit minimiser l'ensemble des coûts ou la consommation de ressources.

Nous avons vu que le seul critère considéré jusqu'ici dans la littérature est la minimisation de la perte de matière, d'espace, de temps ou d'argent. Dans la réalité industrielle, de nombreux autres critères doivent être considérés.

Certains des critères à prendre en compte sont prioritaires par rapport à d'autres. En outre, ils ne sont pas indépendants. Par conséquent, il faut les optimiser simultanément en favorisant les plus importants du point de vue de l'entreprise.

Notons que nous ne pouvons comparer les coûts retenus que s'ils sont évalués dans la même unité de mesure.

2.1.3. Contraintes

Les contraintes sont de deux ordres : celles dues aux modes de découpe (ou contraintes techniques) et celles dues aux méthodes de gestion des découpes.

2.1.3.1. Contraintes techniques

Les formes, les dimensions et les propriétés (fragilité, rigidité...) des entités mères et des entités filles restreignent souvent le nombre de configurations (façons de placer sur une entité mère) admissibles de placement.

Les techniques employées pour réaliser un placement réduisent encore l'ensemble des configurations retenues de placement. Ces contraintes sont par exemples :

- la **circulation** d'un ou plusieurs outils de manutention :
  dans un entrepôt, des palettes sont rangées au mieux afin de minimiser l'espace utilisé. Cependant, il faut laisser entre les palettes un espace suffisant pour le passage des transporteurs de palettes ;
- la **capacité** des machines de coupe :
  les machines de coupe sont limitées en dimensions et/ou en poids total des entités mères à couper ;
- les **types de découpe** à réaliser qui dépendent d'outils spécifiques :
  les types de découpe sont imposés par les outils de coupe et par la matière à découper. Dans le cas du placement de palettes ou de la découpe de tissus, métal, "l'outil de coupe" suit la forme des entités filles, ce qui n'est plus le cas dans les découpes de bois (scie circulaire) et de verre (diamant) , où l'outil coupe l'entité mère d'un bord à l'autre de façon rectiligne.

2.1.3.2. Contraintes de gestion des découpes

La plupart des algorithmes ne considèrent comme étant disponibles en stock que des entités mères identiques (cf. les méthodes utilisant les règles de priorité de la section 3.1 du chapitre II). Seules les méthodes du type des plans de coupe de Gilmore et Gomory ([GIL 32], [GIL 33]) prennent en compte la diversité des stocks disponibles. Ces méthodes font appel à la programmation linéaire appliquée à des plans de coupe (cf. chapitre II, section 4.1).

Cependant, dans ces méthodes, une relaxation des **contraintes de production** est autorisée si les commandes sont très fréquentes ([COS 14], [GIL 32] et [GIL 33]). Il est alors possible de faire de la surproduction. Les entités filles produites en trop sont alors stockées en attendant une prochaine commande de même type, ou encore elles sont vendues avec les autres si le client les accepte. L'excédent réalisé peut aussi être considéré comme une chute lorsque son coût est faible [COS 14]. Les approches sont alors très différentes suivant que la contrainte de production s'applique ou non.

Ces différentes contraintes sont "évidentes" pour le responsable de découpe, mais elles ne sont pas toujours exprimées clairement. Dans la plupart des cas, elles sont affinées au fur et à mesure de l'avancement de l'étude.

2.2. Choix de l'heuristique

Pour un critère d'évaluation donné, un placement est optimal si la valeur du critère est optimale (minimale ou maximale selon le critère).

Si on ne peut prouver qu'un algorithme conduit à un placement optimal pour toutes les données du problème considéré, alors le placement trouvé est une solution approchée. Un tel algorithme est une **heuristique**. Dans le cas contraire, on dira que l'algorithme est optimal ou que la **méthode est exacte**.

Il existe deux approches possibles pour évaluer les algorithmes approchés. On peut en effet mesurer leurs performances en considérant :

- le **pire des cas** *(worst-case analysis)* ;
- le **cas moyen** *(average case behavior)*.

En pratique, on veut avoir une idée du type de comportement d'un algorithme dans les cas les plus défavorables. Cependant, il est difficile de connaître tous les cas typiques de découpe et donc de trouver les cas les plus défavorables. En conséquence, les analyses des cas les plus défavorables sont très ardues même pour les algorithmes simples [GAR 27]. Ceci est encore plus vrai pour les découpes dans $\Re^3$, où toutes les approches sont entièrement composées d'heuristiques spécifiques issues du bon sens.

On peut en outre, souligner qu'un algorithme adapté à une application peut donner de mauvaises solutions dans d'autres applications. La croissance de la complexité au-delà des problèmes à deux dimensions est telle qu'il n'est pas toujours possible de comparer les différentes approches [DOW 18]. En appliquant différentes heuristiques sur plusieurs exemples, Bischoff et Mario [BIS 7] ont montré que les **résultats** sont très **variables**. La meilleure solution serait d'appliquer toutes les heuristiques et de conserver le meilleur résultat. Mais ceci n'est pas envisageable. En effet, leur nombre n'est limité que par l'imagination et le temps de traitement augmente d'autant. De plus, il est difficile de savoir lesquelles conviendraient le mieux à chaque type de problèmes. Les méthodes développées informatiquement donnent en générale des résultats réalisables meilleurs que ceux obtenus "manuellement", et en un temps moindre. Cependant, il existe toujours des cas où les résultats obtenus par le praticien dépassent en qualité ceux des heuristiques.

De plus, la prise en compte d'autres coûts dans la fonction objectif, autre que la chute, complexifie les méthodes de résolution. Les cas représentatifs de découpe ne sont alors plus énumérables, même pour les applications de petites tailles. La seule façon de déterminer le meilleur algorithme pour une application donnée est de le tester de manière empirique sur des jeux d'essai soigneusement choisis. Nous développerons donc des heuristiques que nous comparerons aux solutions réelles actuelles. Puis, nous sélectionnerons la (ou les) plus **efficace(s)** et la (ou les) plus **rapide(s)** en temps d'exécution (car le facteur temps est capital dans ce type d'application).

3. STRUCTURE DE LA THÈSE

L'objectif de cette étude est : analyser et modéliser les problèmes de découpe rencontrés dans l'industrie en utilisant les techniques mathématiques, et proposer un ensemble de méthodes susceptibles de répondre à l'éventail le plus large de ces problèmes.

Nous examinons l'ensemble des approches existantes dans le second chapitre. Les problèmes de placement sont souvent classés dans la littérature suivant les dimensions dans lesquelles on les traite. Nous étudierons plus précisément les découpes à une dimension (barres, tubes...).

Au troisième chapitre, nous présentons précisément les **problèmes de découpe** rencontrés par notre partenaire industriel et expliquons pourquoi les approches décrites au second chapitre ne sont pas applicables. Nous décrivons les besoins particuliers de notre partenaire industriel dans le domaine de la découpe à une dimension, puis des méthodes utilisées pour résoudre les **sous-problèmes** de placement sur une seule barre mère. Ces méthodes seront utilisées pour la résolution du problème global.

La résolution des problèmes de découpe fait appel à la solution de plusieurs sous-problèmes présentés dans le chapitre IV. Les méthodes de résolution sont regroupées en deux grandes catégories correspondant aux besoins particuliers des deux filiales de notre partenaire industriel. Ces deux catégories de méthodes sont décrites aux sections 2 et 3 du chapitre V. Dans la section 2, nous présentons les **méthodes par construction**. Les placements sont étudiés de manière locale ; en d'autres termes, on ne s'intéresse pas aux placements futurs. Dans la section 3, nous développons des **méthodes inspirées de la programmation dynamique**. Elles permettent d'avoir une vue d'ensemble du problème, et par conséquent de tenir compte, dans une certaine mesure, des possibilités futures de placement.

Chapitre II :

MÉTHODES CLASSIQUES

CHAPITRE II

1. INTRODUCTION

Les problèmes de découpe pouvant être résolus à l'aide de méthodes exactes dans un temps raisonnable sont rares.

Par contre, elles peuvent servir à obtenir des solutions optimales pour des problèmes de petite taille ou pour résoudre des sous-problèmes d'un problème de grande taille de manière optimale. Nous présentons dans la suite les méthodes exactes les plus classiques, puis quelques techniques de placement, et enfin quelques méthodes de résolution dans des cas particuliers.

2. MÉTHODES EXACTES

2.1. Modélisation par la programmation linéaire

Généralement, les problèmes combinatoires peuvent se formaliser comme des problèmes de programmation linéaire. Par définition, les méthodes de résolution de programmation linéaire ([GON 35] et [CAR 8]) permettent de trouver la solution optimale d'un problème si sa fonction coût et ses contraintes sont également linéaires. Actuellement, il existe de nombreux algorithmes permettant de résoudre les problèmes d'optimisation lorsque les variables sont réelles.

La méthode de résolution de problèmes de programmation linéaire en nombres réels la plus connue est la méthode du simplexe, due à Dantzig (1959). Elle consiste, en partant d'une solution réalisable, à améliorer la valeur de la fonction objectif en explorant les solutions voisines (c'est-à-dire en faisant évoluer la base). L'optimum est atteint lorsqu'il n'y a pas de meilleure solution voisine.

Cependant, le problème devient, dans le cas général, difficile si les variables deviennent entières. La plupart des problèmes de découpe peuvent se formuler sous forme de programmes linéaires en nombres entiers ou mixtes et nous ne disposons pas d'algorithmes performants pour leur résolution : nous parlerons de problèmes NP-difficiles (cf. chapitre IV, section 3.1.1.).

Un exemple de modélisation par la programmation linéaire en nombre entier est donné par Costa [COS 15] pour la découpe de panneaux de bois. Vues les contraintes techniques et esthétiques (i.e. sens du bois), l'algorithme doit parcourir toutes les configurations de coupe et retenir les quatre qui minimise la chute, et qui respectent au mieux les proportions de chaque type de rectangles commandés. Il ne reste qu'à déterminer, à l'aide d'une méthode de résolution de problèmes de programmation linéaire en nombres entiers (cf. chapitre II, section 2.1), le nombre de panneaux découpés suivant chaque configuration de coupe pour réaliser les commandes.

2.2. Procédure par séparation et évaluation (Branch and Bound)

2.2.1. Principe

Le principe de la procédure par séparation et évaluation, **PSE** ([CEL 11], [COS 13] et [SOU 43]), est une méthode d'énumération implicite, c'est-à-dire qu'elle ne parcourt pas toutes les solutions. Nous supposons dans cette section que l'on veut minimiser un critère donné. Tout d'abord, l'ensemble S des solutions admissibles est partitionné ou décomposé en des sous-ensembles S_i plus petits. Ces sous-ensembles seront eux-mêmes décomposés en de plus petits sous-ensembles, et ainsi de suite tant que les sous-ensembles peuvent être décomposés. La ***séparation*** des solutions est représentée par un arbre dont la racine constitue l'ensemble de départ, et les nœuds les différents sous-ensembles (cf. figure II.1).

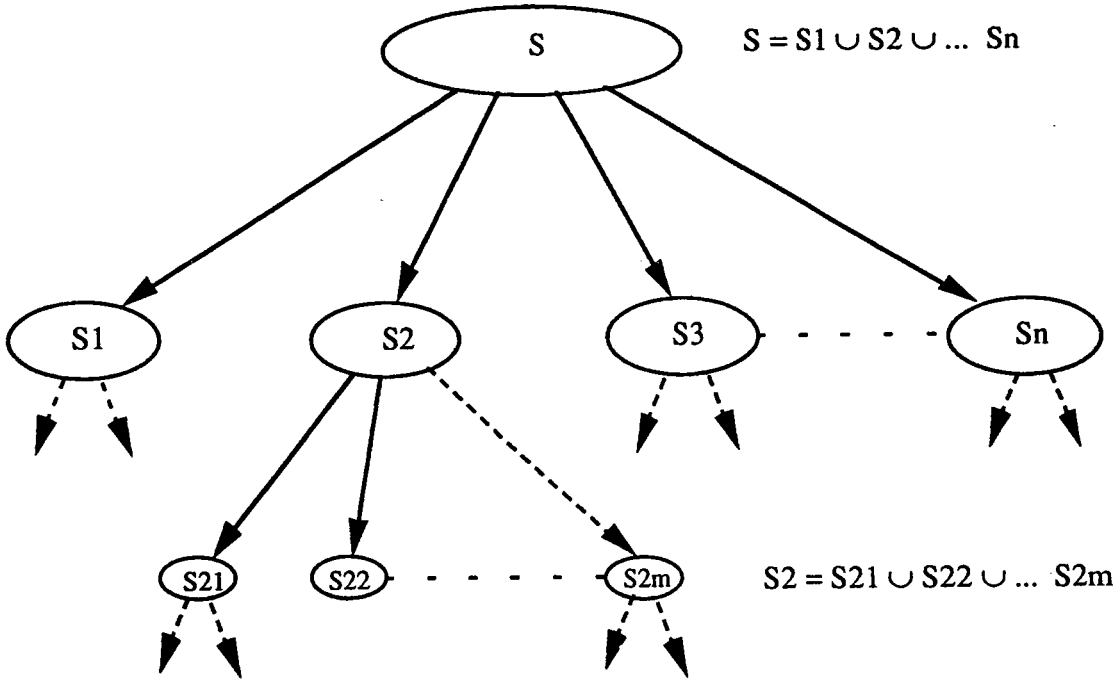

Figure II.1 : Arbre des solutions

Pour éviter de passer en revue tous les sous-ensembles, il est nécessaire d'effectuer une ***évaluation***, pour chacun des sous-ensembles créés d'une ***borne inférieure*** et d'une ***borne supérieure*** de la fonction objectif à minimiser. La valeur de la fonction objectif de la meilleure solution du sous-ensemble considéré est comprise entre ces deux bornes. Pour déterminer une borne inférieure, il est possible de relaxer certaines contraintes. Une borne supérieure du sous-ensemble est la valeur de la fonction objectif obtenue à l'aide d'une heuristique. Elle est supérieure ou égale à la valeur du critère de la solution optimale.

La borne supérieure peut être calculée au début de l'algorithme. Si, au cours de son déroulement, une meilleure solution réalisable a été trouvée, la borne supérieure est alors remise à jour avant de poursuivre l'exploration des autres sous-ensembles.

Si la borne inférieure calculée pour un sous-ensemble (au niveau d'un nœud), est plus grande que la valeur d'une solution réalisable obtenue, alors le sous-ensemble correspondant est supprimé *(cut)*. En effet, la meilleure solution que l'on peut obtenir à partir de ce sous-ensemble (ou de ce nœud) est moins bonne que la solution réalisable déjà obtenue. Nous parlerons d'***énumération implicite***.

2.2.2. Méthodes d'exploration

Les méthodes d'exploration des branches de l'arbre des solutions sont de deux types :

- les Procédures par Séparation et Évaluation **Séquentielle** (PSES) où les nœuds sont explorés :
  - en **largeur d'abord** *(width first)* : les nœuds sont séparés niveau par niveau ;
  - en **profondeur d'abord** *(deepth first)* : le nœud à séparer est le dernier nœud créé. S'il ne peut être séparé alors on revient au nœud précédent *(backtracking)*.
- les Procédures par Séparation et Évaluation **Progressive** (PSEP) ou **Mixte** (*branch and bound with jumptracking*) : la séparation s'effectue en premier par les nœuds les plus prometteuses (de borne inférieure la plus faible par exemple).

Le nombre de solutions augmente exponentiellement avec la taille du problème. Le temps d'exécution de la procédure par séparation et évaluation dépend donc de la qualité des méthodes de calcul des bornes. En effet, plus elles sont proches de l'optimum, plus le nombre de sous-ensembles à explorer est restreint.

2.2.3. Application

Costa [COS 14] utilise avec succès une PSE pour des petits problèmes de minimisation de chute dans les découpes de bobines en bobineaux de même longueur et de largeurs quelconques. Il s'agit d'une procédure par séparation et évaluation en profondeur d'abord en **variables bivalentes** (0 ou 1, i.e. on considère à chaque niveau si un élément est ou non compris dans le sous-ensemble des solutions).

Dans la pratique, le nombre de solutions à explorer est qu'il est impossible de les considérer toutes. Par conséquent, on se limite dès le départ à un sous-ensemble de configurations possibles. Ces configurations sont choisies parmi celles qui génèrent le moins de chute et dont le nombre d'éléments de chaque type réalisés est proportionnel à la demande.

Costa conseille de ne pas découper plus de 7 bobineaux de même largeur dans une seule bobine pour avoir un temps de traitement raisonnable. Le nombre maximal de bobineaux réalisés dans une bobine, toutes largeurs confondues, est de 16.

Partant du principe qu'on ne retient qu'une partie des configurations possibles, la PSE est contrainte à produire plus pour réaliser au moins toutes les commandes. Cette méthode présente donc l'inconvénient de créer des ***"surlongueurs"*** importantes des bobineaux commandés ou

"surproduction" dans le cas d'éléments de formes figées (barres ou panneaux de dimensions précises). Les produits en surnombre devront alors être stockés.

2.3. Programmation dynamique ou récursive

La programmation dynamique ([GON 35] et [CAR 8]) est une classe de méthodes décomposant un problème en sous-problèmes plus petits de même nature. Tous les sous-ensembles de solutions sont évalués. Ils sont éliminés de la recherche lorsqu'ils sont moins intéressants que d'autres.

La méthode, développée par P. Massé en 1944, est destinée à la résolution de problèmes de cheminement. Dans ce type de problèmes, il s'agit de trouver le parcours le plus court (respectivement le plus long) entre deux points A et B. Soit C un point appartenant au chemin optimal entre A et B. Bellman (1957) a souligné que les distances entre le point C et les deux autres points sont aussi optimales. En effet, si tel n'était pas le cas, il existerait une solution meilleure constituée des chemins optimaux de A à C et de C à B.

Le corollaire du principe de Bellman est le suivant : si le chemin de A à C est imposé, alors le plus court (ou le plus long) chemin de A à B utilisant la partie imposée AC est obtenu en complétant la partie imposée par un sous-chemin optimal allant de C à B.

Nous voyons que le problème à résoudre doit être ***décomposable*** en sous-ensembles complémentaires (ou niveaux) pour permettre une ***recherche séquentielle*** des solutions. La formulation de la fonction objectif d'une solution est récurrente : en effet, elle s'écrit à partir des valeurs des fonctions objectifs des sous-ensembles dont elle est composée.

Même si cette méthode ne considère pas tous les choix de placements possibles (énumération implicite), le nombre de solutions considérées reste élevé, même pour de petites applications.

3. MÉTHODES APPROCHÉES

Pour les problèmes de grande taille, on a alors recours à des méthodes approchées **(heuristiques)** qui sont rapides et qui donnent des solutions pour lesquelles on espère que les valeurs de la fonction objectif sont aussi proche que possible de celle des solutions exactes. L'intérêt des méthodes approchées réside dans la possibilité d'employer des algorithmes bien ***moins coûteux en temps de calcul et en mémoire.***

Dans cette partie, nous présentons d'abord les méthodes généralement issues du bon sens suivant les dimensions dans lesquelles sont plongés les problèmes de découpe, puis des techniques plus générales comme le recuit simulé et des méthodes de type systèmes experts.

CHAPITRE II

3.1. Méthodes utilisant des règles de priorité

3.1.1. Découpes en une dimension

3.1.1.1. NEXT FIT (NF) [COF 12] et [GAR 27]

Soit une liste **L** d'éléments unidimensionnels l_i de longueurs quelconques. Ces éléments sont, par exemple, des commandes de barres d'acier. Elles sont à découper dans des barres mères de même longueur.

Les éléments de la liste **L** sont dans un **ordre quelconque**. Les barres à réaliser sont prises dans cet ordre pour être découpées dans une **barre quelconque** du stock. Lorsque la barre l_i à découper est plus longue que la chute résultant de la découpe précédente, on passe à une nouvelle barre mère. On débite ainsi les barres du stock les unes après les autres.

Cet algorithme est très rapide. Le temps d'exécution est une fonction linéaire du nombre d'éléments à placer. Mais son défaut est qu'il n'effectue ses essais que dans la dernière barre entamée avant de passer à une nouvelle barre du stock. On constate une perte de matière réutilisable dans les barres mères déjà entamées : ceci a conduit à construire l'algorithme suivant.

3.1.1.2. FIRST FIT (FF) [COF 12], [JOH 39] et [GAR 27]

Les données initiales sont identiques aux précédentes. La liste L comporte, dans un ordre quelconque, des barres filles l_i de longueurs quelconques. Les éléments l_i sont pris dans l'ordre de la liste **L** pour être découpés dans des barres du stock qui comporte, outre les barres de même longueur, les **chutes** provenant **des découpes précédentes**. Chacun des éléments l_i est placé sur la première chute qui lui convient, ces chutes étant explorées dans l'ordre de leur création. Une nouvelle barre est tirée du stock si l'élément l_i n'entre dans aucune des chutes.

La qualité des solutions diminue lorsque les éléments l_i les plus longs sont en fin de liste **L**.

3.1.1.3. FIRST FIT DECREASING (FFD ou Sac à dos) [COF 12] et [GAR 27]

La liste **L** des éléments l_i à découper est triée dans l'ordre **décroissant de leur longueur** dans la liste. Il ne reste plus qu'à appliquer l'algorithme First Fit pour obtenir le placement. Ainsi les éléments l_i les plus longs sont traités en premier.

3.1.1.4. BEST FIT (BF) [COF 12], [JOH 39] et [GAR 27]

Les éléments l_i sont de longueurs variées. Ils figurent dans un ordre quelconque dans la liste **L** des éléments à découper dans des barres stockées de même longueur.

Le placement est du même type que celui du First Fit. Les éléments l_i sont découpés au fur et à mesure suivant l'ordre de la liste **L**. Cependant, les chutes sont considérées dans l'ordre

croissant de leur longueur. Une nouvelle barre est tirée du stock pour être découpée si l'élément l_i à placer ne convient à aucune des chutes générées précédemment.

3.1.1.5. WORST FIT (WF) [COF 12], [JOH 39] et [GAR 27]

Des éléments l_i de longueurs variées figurent dans la liste L dans un ordre quelconque. Les barres en stock ont des longueurs identiques. Comme dans la méthode précédente, cet algorithme place les éléments l_i, suivant l'ordre de la liste L, dans la chute **la plus courte** : les chutes sont considérées dans l'ordre **décroissant de leur longueur**. Si aucune des chutes déjà créées ne convient à l'élément l_i, alors on découpe l'élément l_i dans une nouvelle barre du stock.

3.1.1.6. ALMOST WORST FIT (AWF) [COF 12] et [GAR 27]

Des éléments l_i à découper sont de longueurs quelconques. Ils sont rangés au hasard dans la liste L. Ils sont à découper dans un stock de barres d'égales longueurs. Les éléments l_i sont à réaliser suivant l'ordre de la liste L. On essaie d'abord de placer l'élément l_i dans la **deuxième chute la plus courte** avant de reprendre le déroulement de l'algorithme Worst Fit.

Le résultat obtenu peut-être différent du résultat donné par l'heuristique précédente. Cependant, il n'est pas possible d'affirmer qu'elle est meilleure qu'une autre car les résultats sont variables suivant les ensembles des problèmes traités.

3.1.1.7. WORST FIT DECREASING (WFD) [COF 12] et [JOH 39]

Le stock de départ contient des barres de longueur identique. Les commandes de barres l_i sont de différentes longueurs. Dans l'algorithme Worst Fit Decreasing, la liste L des éléments l_i à placer est triée dans l'**ordre décroissant des longueurs** des éléments avant de reprendre la démarche de l'algorithme Worst Fit.

3.1.1.8. NEXT-k FIT (NkF) [COF 12], [JOH 39] et [GAR 27]

Des éléments unidimensionnels l_i de longueurs variées sont à découper dans des barres mères de même longueur. L'algorithme Next-k Fit, avec $k \geq 1$, ressemble à l'algorithme Next Fit à ceci près que l'élément l_i n'est placé dans une nouvelle barre mère que s'il ne peut être placé dans aucune des **k dernières chutes** créées (Next-1 Fit est identique à Next Fit).

3.1.1.9. ANY FIT (AF) [COF 12], [JOH 39] et [GAR 27]

Le stock contient des barres de même longueur. Il s'agit d'y découper des commandes l_i de longueurs variées. L'ordre des éléments l_i à découper est quelconque. La chute dans laquelle l'élément l_i va être découpé est choisie au **hasard parmi toutes les chutes** de plus grandes longueurs. L'élément l_i est affecté à une barre tirée du stock s'il ne peut être placé dans aucune des chutes déjà générées.

3.1.1.10. ALMOST ANY FIT (AAF) [COF 12] et [GAR 27]

Des commandes l_i de longueurs variées sont à découper dans des barres en stock de même longueur. Cet algorithme est une modification de l'algorithme précédent : l'élément l_i n'est **jamais placé sur la chute créée la plus longue**, sauf si celle-ci est la seule possible.

3.1.1.11. ANY FIT DECREASING (AFD) [COF 12] et [GAR 27]

Les éléments l_i à découper sont de longueurs variées et on dispose en stock de barres de même longueur.

La démarche est la même que pour l'algorithme Any Fit, mais les éléments l_i de la liste **L** sont classés dans l'**ordre décroissant de leur longueur**. Le premier élément à placer est alors le plus long de ceux qui restent dans la liste.

3.1.1.12. ITERATED LOWEST FIT DECREASING (ILFD) [COF 12] et [GAR 27]

Nous découpons dans des barres mères de même longueur des commandes l_i de longueurs variées. Comme pour l'algorithme FFD, les éléments l_i de la liste **L** sont triés dans l'ordre décroissant de leur longueur.

Nous définissons un entier **q** qui sera le nombre minimal de barres mères utilisées. Ce nombre est défini à l'aide d'une heuristique grossière : par exemple, **q** est le nombre minimal de barres mères dont le poids est supérieur ou égal au poids total des commandes.

Ensuite, nous choisissons **q** barres, $B_1, B_2, ..., B_q$, dans le stock et nous plaçons les éléments l_i sur les barres mères entamées ou les barres mères $B_1, B_2, ..., B_q$ les plus longues à chaque itération.

Si, à un moment donné, un élément l_i ne peut pas être placé sur l'une des **q** barres mères ou sur l'une de leurs chutes, nous recommençons le placement sur une nouvelle barre du stock. Lorsque tous les éléments l_i de la liste **L** sont placés, nous avons le nombre de barres mères qu'il est nécessaire d'utiliser pour réaliser cette découpe, donc la chute correspondant à cette solution.

3.1.1.13. MODIFIED FIRST FIT (RFF) [COF 12] et [GAR 27]

Sur un stock de barres de même longueur, nous plaçons des commandes l_i de longueurs variées. La liste **L** des éléments l_i à découper est partagée en trois listes L_A, L_D et L_X ayant approximativement le même cardinal. Les éléments de la liste L_A sont d'abord placés par la méthode FFD. Après cette première découpe, les parties restantes des barres mères entamées sont appelées des "***barres-A***". Il s'agit alors de placer les éléments de la liste L_D sur les "***barres-A***" par la procédure suivante :

1. *Début de boucle*
   - 1.1. *Soit B_j la "barre-A" la plus longue. Si les deux plus petites barres commandées non réalisées de L_D ne peuvent pas être découpées dans B_j alors Fin de boucle*
   - 1.2. *Soit l_i le plus petit élément de L_D non placé. Placer l_i sur B_j : pour obtenir des solutions différentes du simple FFD, cette méthode place à cette étape le plus petit élément de L_D et non pas comme dans l'algorithme FFD le plus grand élément*
   - 1.3. *Soit l_k le plus grand élément de L_D non placé qui convient à B_j. Placer l_k sur B_j*
2. *Fin de boucle*

Le placement des éléments $\mathbf{l_i}$ sur les barres mères se termine en plaçant le reste des éléments de $\mathbf{L_D}$ et les éléments de $\mathbf{L_X}$ sur l'ensemble des barres entamées et non entamées par la méthode FFD.

D'après Coffman [COF 12], cette méthode donne globalement de meilleurs placements que la simple méthode FFD.

3.1.2. Découpes en deux dimensions

L'objectif des algorithmes suivants est de placer des éléments rectangulaires de taille quelconque sur une bande de largeur fixe et de longueur la plus faible possible. Pour simplifier la compréhension, la bande découpée sera prise ***verticalement***. Il s'agit alors de minimiser la ***hauteur*** utilisée (cf. figure II.2).

La rotation des rectangles n'est pas autorisée. Pour simplifier la notation, les noms des côtés des rectangles à découper sont :

- la ***largeur*** pour le côté parallèle au ***bord du début*** de la bande.
- la ***hauteur*** pour le côté parallèle au ***bord latéral*** de la bande.

3.1.2.1. BOTTOM-LEFT (BL) [COF 12], [BAK 5]

Dans cet algorithme, chaque élément est d'abord placé le plus bas possible sur la bande à découper et le plus à gauche possible. Par exemple, dans la figure II.2, des rectangles à réaliser sont pris dans un ordre quelconque. Ils sont numérotés de 1 à 4 et sont placés dans cet ordre sur la bande afin de minimiser la hauteur utilisée.

Il faut remarquer qu'il y a une différence entre les placements à une dimension (i.e. First Fit et Next Fit) et ceux à deux dimensions (Bottom-Left). Dans le cas unidimensionnel, il existe toujours un ordre dans la liste **L** où les algorithmes FF ou NF donnent le placement optimal. Il suffit pour cela que les entités filles (barres) à placer soient dans l'ordre où elles devraient être placées sur les entités mères pour aboutir à l'optimum. Cela n'est pas le cas pour l'algorithme Bottom-left. Baker [BAK 5] montre qu'il n'existe pas toujours d'ordre de la liste **L** des éléments rectangulaires à placer donnant un placement optimal. Il y a cependant des ordres préférables aux autres. Ainsi, plusieurs algorithmes de type Bottom-left peuvent être considérés, ils dépendent des éléments à placer :

- BOTTOM-LEFT INCREASING WIDTH (BLIW) [COF 12] : les rectangles sont classés par ordre ***croissant*** de leur ***largeur*** ;
- BOTTOM-LEFT INCREASING HEIGHT (BLIH) [COF 12] : les rectangles sont classés par ordre ***croissant*** de leur ***hauteur*** ;
- BOTTOM-LEFT DECREASING WIDTH (BLDW) [COF 12] : les rectangles sont classés par ordre ***décroissant*** de leur ***largeur*** ;
- BOTTOM-LEFT DECREASING HEIGHT (BLDH) [COF 12] : les rectangles sont classés par ordre ***décroissant*** de leur ***hauteur***.

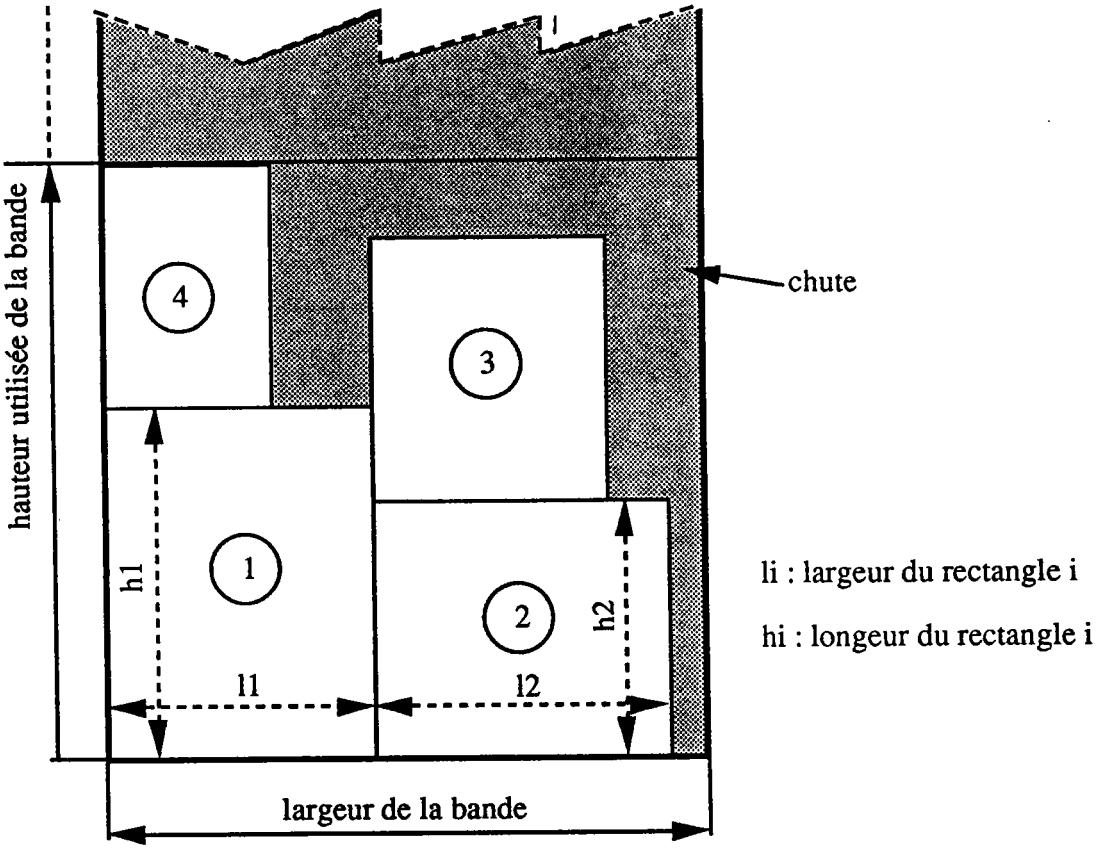

Figure II.2 : Exemple de placement par la méthode BOTTOM-LEFT

3.1.2.2. Algorithmes à niveaux [COF 12], [BAK 5], [DOW 18] et [GAR 27]

Pour certains types de découpe, comme dans les scieries et les verreries, l'outil de coupe doit aller d'un bord à l'autre du rectangle ou de la bande à découper. Ces types de découpe sont appelés découpes **guillotine** ou **bout-à-bout**. Les **algorithmes à niveaux** permettent ce type de placement.

Le principe est le suivant : d'abord, les rectangles sont triés par ordre décroissant de hauteur (pas de rotation des rectangles). Le placement est alors une succession de rangées parallèles à la

largeur de la bande. A chaque niveau, les rectangles sont calés le plus à gauche possible (cf. figure II.3). Le premier niveau est simplement le bas de la bande. Le bas du niveau suivant est délimité par le plus haut rectangle du niveau précédent. L'opération est réitérée tant qu'il reste des rectangles à placer.

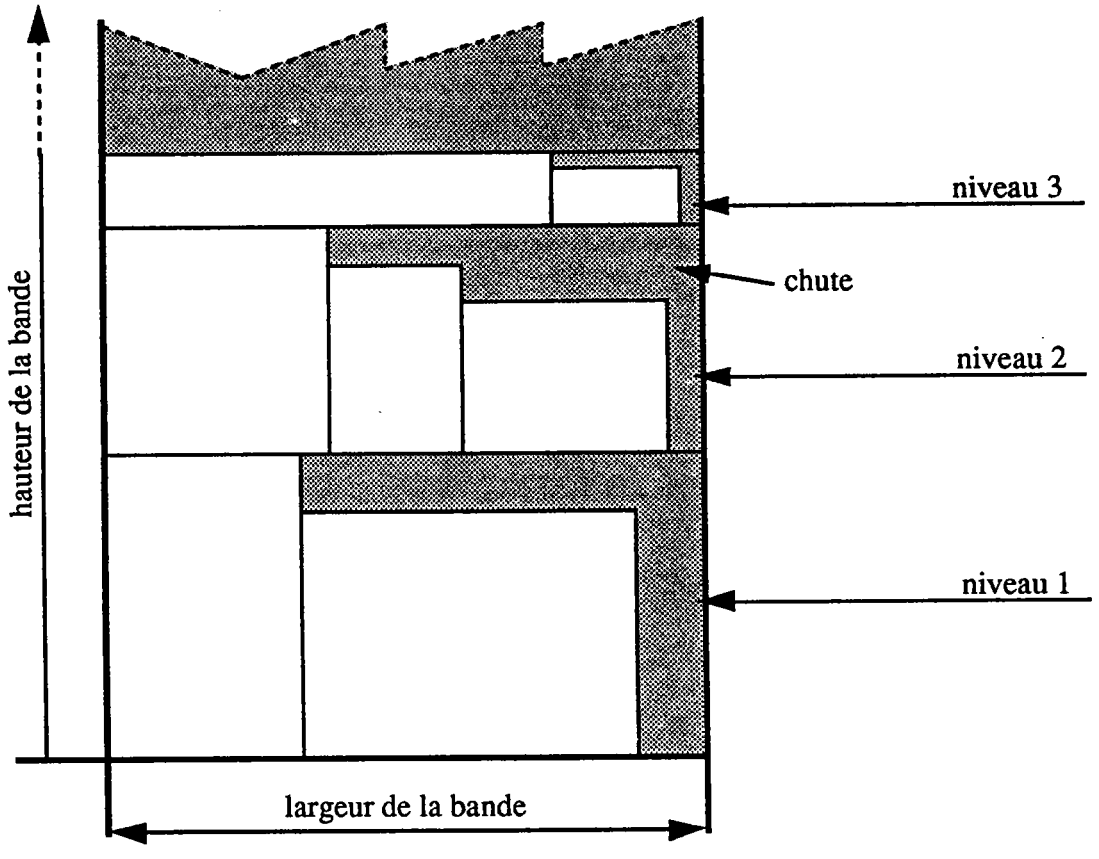

Figure II.3 : Exemple de placement par niveau

La découpe guillotine **en trois étapes** correspond au nombre de reprises de coupe. La découpe de la figure II.3 nécessite en premier un jeu de guillotines horizontales (une pour chaque niveau), puis verticales (pour séparer les éléments d'un même niveau) et enfin horizontales (pour raccourcir la hauteur de certains éléments).

Le placement à chaque niveau fait appel aux méthodes de placement dans $\Re^1$ décrites ci-après :

- NEXT FIT DECREASING HEIGHT (NFDH) [COF 12] et [GAR 27]

  Dans cet algorithme, les rectangles sont d'abord classés par hauteurs décroissantes. Le placement est réalisé niveau par niveau. L'algorithme Next Fit est utilisé pour résoudre le problème unidimensionnel consistant à placer sur un niveau les rectangles dans l'ordre du classement. Il y a changement de niveau lorsqu'un élément est plus long que la place restante.

- FIRST FIT DECREASING HEIGHT (FFDH) [COF 12] et [GAR 27]
  C'est la même démarche que la précédente pour cet algorithme, si ce n'est l'utilisation de l'algorithme First Fit pour résoudre le problème à une dimension.

3.1.3. Découpes en trois dimensions

Si les formes sont parallélépipèdiques, il est possible d'envisager une solution. Certaines méthodes proposées pour les problèmes à deux dimensions peuvent être généralisées.

Il est possible d'utiliser la méthode de placement par couches successives aussi bien en deux dimension qu'en trois dimension : on généralise alors l'algorithme Bottom-Left dans $\Re^2$ au **Bottom-Left-Up** dans $\Re^3$ [DOW 18].

George et Robinson [GEO 31] proposent un algorithme pour remplir des containers identiques de caisses de formes variables. Leur algorithme est une amélioration des algorithmes utilisant des règles de priorité pour les découpes dans $\Re^1$. Ils utilisent le principe du Best-Fit-Decreasing, tout en ajoutant des contraintes particulières au problème de placement dans un container de paquebot : les caisses sont classées suivant une série de règles de priorité pour que les caisses de même type soient regroupées. Les caisses sont d'abord placées en colonnes à l'arrière du container pour réaliser le premier niveau. On remplit ensuite au mieux les containers par niveau (le plus à l'arrière possible).

Cependant, dans la plupart des cas [DOW 18], il n'est pas possible d'empiler les entités filles en couches distinctes. Il faut dans ce cas considérer ce problème dans l'espace à trois dimensions.

Il est difficile de comparer les différents algorithmes de placement dans $\Re^3$ ([DOW 18] et [DOW 20]). En effet, il faut savoir que ***la qualité des solutions*** d'un algorithme est étroitement ***liée au type de données*** qu'on lui fournit. La difficulté réside alors dans la détermination des domaines d'utilisation d'un algorithme en fonction du type de données à traiter (y compris pour l'optimisation d'un seul critère). En général, les algorithmes sont testés avec des données réelles. Ils reflètent donc simplement les caractéristiques des données testées ([GEO 31] et [BIS 7]).

3.2. Par voisinage : recuit simulé

Les méthodes par voisinage consistent à modifier une (ou plusieurs) solution(s) courante(s) de manière à obtenir une (ou plusieurs) solution(s) "voisine(s)". Ce(s) solution(s) voisine(s) seront retenue(s) si elle(s) améliore(nt) la(es) solution(s) courante(s). Les méthodes les plus connues sont la recherche tabou *(tabu search)*, les algorithmes génétiques ou encore le recuit simulé.

La méthode du recuit simulé (*simulated annealing*, [KIR 40], [LAA 41], [PRO 45], [SIA 46] et [SOU 47]) est inspirée des traitements thermiques des métaux.

3.2.1. Analogie physique

Les traitements thermiques sont utilisés pour modifier les caractéristiques physiques et chimiques d'un produit métallurgique. Ces caractéristiques sont fonctions de l'arrangement des atomes entre eux. Le recuit a pour objectif de redonner au métal sa stabilité et ses caractéristiques "normales".

Le métal est chauffé à une **température** proche de sa température de fusion. L'énergie du système est alors élevée, et par conséquent les atomes ont suffisamment d'énergie pour se déplacer dans le métal. La température est ensuite abaissée lentement par **paliers** suffisamment longs pour que les atomes aient le temps de se réorganiser en réseaux cristallins ordonnés et stables. Cet état d'énergie est un **minimum absolu**, proche de l'**optimum théorique**.

Par opposition, lors d'une trempe, le métal est refroidi très rapidement. La structure du métal conserve l'état amorphe de l'état liquide. L'énergie du système reste élevée. C'est un **minimum local** d'énergie.

3.2.2. Simulation du recuit

Le principe du recuit utilisé en métallurgie est repris pour résoudre les problèmes combinatoires. Cette méthode cherche à se dégager des optima locaux.

Partant d'une solution réalisable s et d'une fonction économique f(s), le système est perturbé pour obtenir une nouvelle solution réalisable **s'** proche de la précédente. La valeur de la fonction économique de notre problème est calculée pour la nouvelle solution : f(s'). Cette solution est retenue si elle améliore le résultat précédent, mais aussi si la nouvelle valeur de la fonction économique calculée n'est pas "trop éloignée" de la précédente, et ceci avec une probabilité p qui diminue avec le paramètre **température T**.

$$p = \exp\left(-\frac{|f(s) - f(s')|}{T}\right) \qquad \text{(II.1)}$$

Ce paramètre, équivalent à la température dans le recuit physique, diminue lorsque le nombre d'itérations augmente prémunisant ainsi les choix dégradant trop les solutions en fin de l'exécution de l'algorithme. Souilah [SOU 47] et Proth [PRO 45] utilisent une procédure simple de diminution de la température. Soit pour la $k^{ième}$ diminution de la température :

$$T_k = rg \times T_{k-1} \text{ avec } rg \in [0,1] \qquad \text{(II.2)}$$

Van Laarhoven et Aarts [LAA 41] proposent pour **rg** des valeurs comprises entre 0,85 et 0,95.

Comme dans le recuit physique, le cycle thermique comporte aussi des **paliers** de températures constantes. Les paliers doivent être de longueur suffisante pour laisser au système le temps d'atteindre un état d'équilibre. Le contrôle de la longueur de chacun des paliers est régi par des bornes supérieures du nombre d'acceptations et du nombre de refus de configurations voisines.

Une nouvelle configuration du système est recherchée tant que la température n'a pas atteint la température finale.

3.2.3. Algorithme du recuit simulé

Dans le cas d'une minimisation de la fonction économique, l'algorithme général du recuit simulé s'écrit de la façon suivante :

ALGORITHME DU RECUIT SIMULÉ

1. Choisir une solution réalisable s , initialiser k = 0, T_0

2. Initialiser le paramètre température $T_k = T_0$, la température finale, les longueurs des paliers, la diminution de la température rg

*3. Tant que la **température finale** n'est pas atteinte, faire*

*3.1. Tant que les **conditions sur le palier** ne sont pas atteintes, faire*

3.1.1. Choisir une solution réalisable s' voisine de s

3.1.2. Calculer la probabilité d'acceptation $p = \exp\left(-\frac{|f(s)-f(s')|}{T_k}\right)$

3.1.3. Générer une variable aléatoire p' comprise entre 0 et 1

3.1.4. Si ((f(s') < f(s)) ou (p' < p)), alors

s' = s

3.2. Fin tant que

3.3. Incrémenter k de la température : $T_k = rg \times T_{k-1}$

4. Fin tant que.

3.2.4. Applications

L'efficacité de la méthode du recuit simulé dépend du problème traité et du choix des paramètres utilisés (longueurs des paliers, vitesse de refroidissement, température initiale et température finale). Cette méthode est générale et utilise peu de spécifités de notre problème. En contrepartie, le temps de traitement peut être important. C'est pourquoi elle est parfois surclassée par les heuristiques développées spécialement pour des applications particulières.

On peut utiliser très simplement le recuit simulé pour la découpe dans $\Re^1$:

- on prend comme solution initiale la liste des barres l_i à découper classées aléatoirement.
- la valeur de la fonction objectif est la mesure de la chute après avoir appliqué l'heuristique First Fit sur notre liste.

- une solution voisine sera obtenue en permutant deux barres dans la liste des éléments à découper.

Dowsland [DOW 19] donne deux applications possibles du recuit simulé pour le placement dans $\Re^2$:

- placement d'un nombre donné de pièces parallélépipèdiques de bases rectangulaires identiques sur une surface plus vaste (pallet loading problem) où il s'agit de minimiser le chevauchement des pièces ;
- placement d'un nombre donné de pièces parallélépipèdiques de ***même hauteur*** dans une boîte de base donnée. L'objectif est de minimiser la hauteur nécessaire. A chaque étape, le modèle ($\Re^3$) est ramené à un problème à deux dimensions ($\Re^2$) du même type que précédemment où les pièces rectangulaires peuvent être de dimensions différentes.

3.2.5. Parallélisation du recuit

Avec l'apparition des **architectures informatiques parallèles** (multiprocesseurs), Darema et Co [DAR 16] proposent deux méthodes de parallélisation du recuit simulé pour un problème de placement de "chips" (de circuits ou de cellules) de dimensions identiques pour IBM Yorktown Heights (comme nous l'avons vu, ce problème peut être vu comme un problème de découpe). L'objectif est de minimiser les liaisons (électriques) entre les chips sans tenir compte de l'orientation (cf. figure II.4).

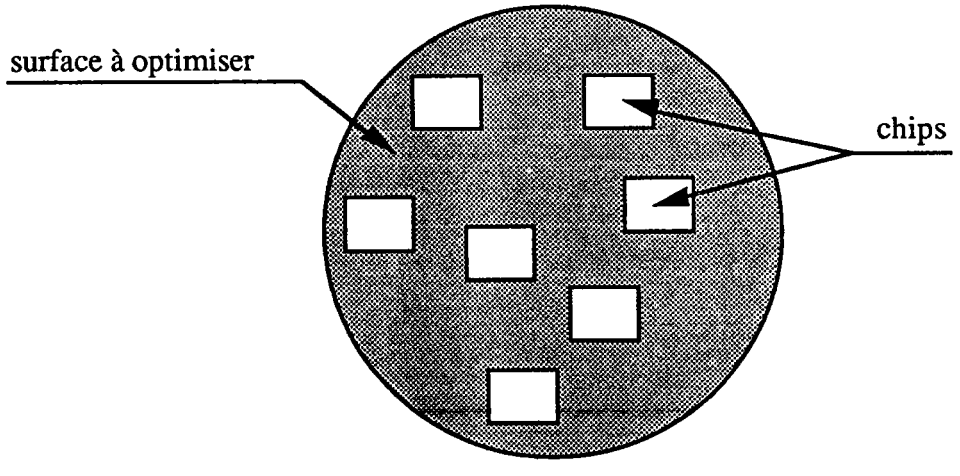

Figure II.4 : Exemple de configuration pour Darema et Co

Première méthode

Dans la première méthode, pour chaque valeur de la température, la configuration **globale** du système est gardée en mémoire et les différents processeurs du calculateur échangent en même temps des paires de chips (cf. figure II.5).

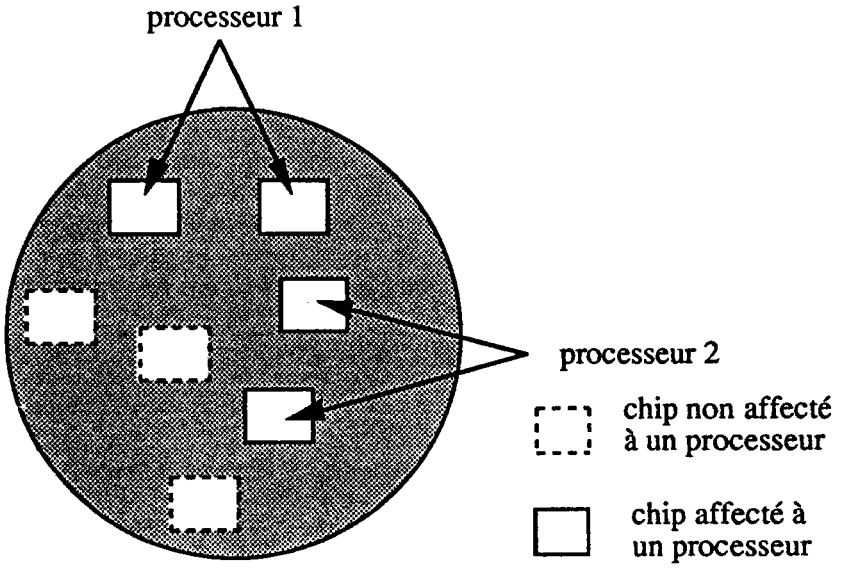

Figure II.5 : Échange de paires de chips avec les deux chips bloqués

Chaque processeur sélectionne au hasard deux chips non affectés de la configuration globale. Ces deux chips sont alors ***"affectés"*** pour les autres processeurs. Le processeur calcule alors localement l'énergie de la nouvelle configuration (sans tenir compte des permutations calculées par les autres processeurs).

Cette énergie **locale** est comparée à l'énergie globale (meilleure configuration). Si la nouvelle configuration est retenue, suivant les critères d'acceptation, le processeur remet à jour la configuration retenue ainsi que la nouvelle énergie globale.

Chaque processeur calcule une configuration et sauvegarde la configuration locale dans la configuration globale lorsqu'elle est retenue. Nous voyons qu'un problème se pose lorsqu'une nouvelle configuration globale est retenue : les configurations en cours de calcul sont toujours basées sur une ancienne configuration globale. Comme les différences de niveaux d'énergie sont faibles, on conserve toujours la dernière configuration retenue.

Deuxième méthode

La deuxième méthode est dérivée de la précédente. Pour pouvoir employer plus de processeurs, seul un des deux chips à déplacer est bloqué (cf. figure II.6).

Dans cet algorithme, les conflits au niveau des configurations retenues et celles en cours de comparaison sont plus importants. Le décalage entre le niveau d'énergie de la configuration retenue et ceux des configurations en cours de calcul reste cependant faible.

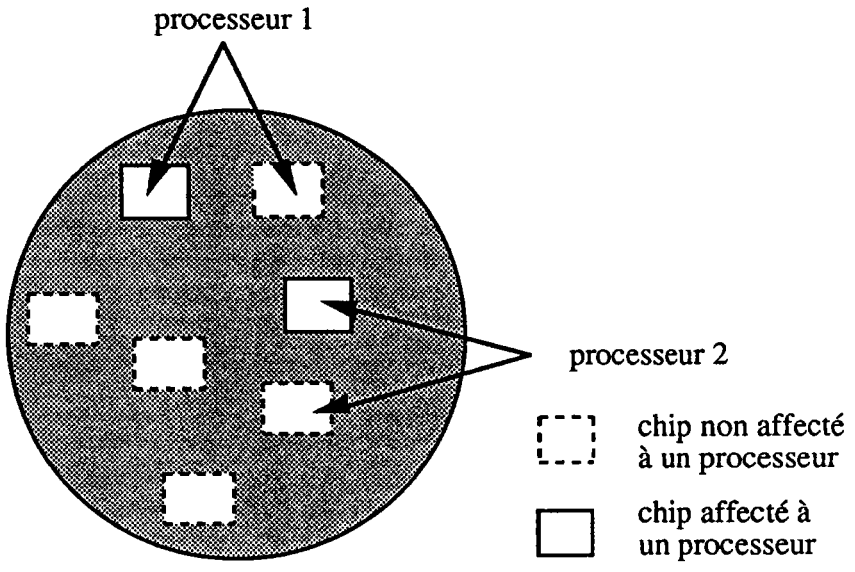

Figure II.6 : Échange de paires de chips avec un des chips bloqués

L'optimalité des placements résultants n'est pas assurée, sauf pour les cas où les structures sont très régulières. Le placement optimal est alors connu en raison de la simplicité extrême des objets. Dans d'autres cas, la décision d'accepter un placement repose finalement sur le ***jugement de l'utilisateur*** et sur la confiance qu'il accorde à cette méthode.

3.3. Systèmes experts

La complexité de certains problèmes est telle que les méthodes générales d'optimisation ne peuvent pas apporter de solution satisfaisante. On essaie alors de ***formaliser le raisonnement du spécialiste*** du domaine. Les systèmes experts permettent d'automatiser la démarche suivie par un expert. Il faut remarquer que les méthodes de l'expert aboutissent à des solutions réalisables mais non optimales. Par conséquent, les systèmes experts basés sur le raisonnement humain sont rarement des méthodes exactes. Les résultats obtenus peuvent aussi être améliorés par des moyens informatiques qui permettent de considérer un ***éventail*** plus ***important de possibilités***.

Les réalisations dans le domaine de la découpe sont limitées par le manque de règles permettant de formaliser le processus de placement. Les systèmes experts réalisés [PER 44], [HEN 37] et [WAT 48], pour les problèmes de placement de cellules électriques, intègrent des contraintes du type **adjacent** (deux cellules doivent être collées), **visibilité** (câblages rigides ou semi-rigides entre certaines cellules) ou **distance** (longueur maximale de câblages entre cellules). Cependant, ces contraintes agissent sur des objets sélectionnés et ne peuvent traiter efficacement le problème de l'optimisation des découpes où les contraintes sont communes à l'ensemble des pièces [DEL 17]. Par conséquent, les études et réalisations actuelles portent d'avantage sur l'optimisation de l'espace que sur l'optimisation de découpe.

Clerc et Sadgal [CEL 11] proposent un prototype relativement général pouvant traiter différents types d'applications (aménagement de salles, implantation des composants électroniques, partitionnement en vue de découpe...). Ils conseillent d'introduire des heuristiques génératrices de contraintes afin de limiter l'explosion combinatoire.

4. MÉTHODES SPÉCIFIQUES

4.1. Méthode des plans de coupe

Gilmore et Gomory (1963) ont largement contribué à la résolution des problèmes de découpe [GIL 32], [GIL 34] et [GIL 33]. La résolution de ce type de problème exprimé en termes de problèmes de programmation linéaire en nombre entier n'est pas envisageable en raison du nombre important de variables nécessaires. La même difficulté persiste lorsqu'on cherche une solution approchée à l'aide de la programmation linéaire en variables réelles. En effet, chaque colonne de la matrice représente les différentes façons de couper les barres, bobines ou panneaux (désignés par **plans de coupe**). Le nombre de plans de coupe (ou colonnes) est alors très important.

Comme, pour un problème pratique, il n'est pas possible de les énumérer tous, Gilmore et Gomory proposent une méthode astucieuse qui consiste à résoudre, à chaque itération du **simplexe**, un problème auxiliaire. La solution du problème auxiliaire fournira la "colonne qui entre en base" (cette colonne est un **plan de coupe** réalisable). Il peut être résolu par la programmation dynamique (méthode lente dans ce cas) ou par la méthode du sac à dos *(knapsack)*.

La solution obtenue est en valeur réelle, et ils suggèrent de la tronquer pour obtenir une solution approchée entière.

Pour des petits problèmes de découpe en une dimension où le nombre de plans de coupe réalisables est faible (moins de 600), Goulimis [GOU 36] montre qu'il est possible de trouver la (les) solution(s) optimale(s) en combinant les plans de coupe et la programmation linéaire de Gilmore et Gomory [GIL 32], ainsi que les procédures par séparation et évaluation (PSE ou branch and bound) :

- Goulimis conserve tous les plans de coupe pour chercher la meilleure combinaison de plans de coupe par la méthode du simplexe. Les variables trouvées (correspondant aux nombres de chaque type de plan) ont des valeurs réelles.
- La solution entière est obtenue en utilisant les PSE dont les branches correspondent aux valeurs des variables arrondis par excès et par défaut.

L'algorithme est optimal car Goulimis a conservé tous les plans de coupe possibles.

Les plans de coupe et l'utilisation du simplexe par Gilmore et Gomory sont à l'origine de nombreuses méthodes de découpe. Moreau [MOR 43] présente dans sa thèse les différents problèmes de découpe qui se posent dans l'industrie et propose une étude plus approfondie des plus courants. Il s'inspire des plans de coupe de Gilmore et Gomory [GIL 32] et [GIL 33] pour :

- découper des bobines de longueurs infinies en bobineaux d'une longueur donnée unique ($\Re^1$) ;
- découper "bout-en-bout" des rectangles dans des rectangles, c'est-à-dire couper à l'aide d'un outil allant toujours d'un bord à l'autre de la pièce qu'il découpe ($\Re^2$).

Dyckhoff [DYC 22] a repris l'approche des plans de coupe et de la programmation linéaire pour l'appliquer à la découpe dans $\Re^1$ en tenant compte des **résidus réutilisables** dans la fonction objectif.

4.2. Approches pour les découpes de formes complexes

La découpe de formes complexes soulève un nouveau problème dans la découpe en deux dimensions. Jusqu'à présent, les positionnements possibles relatifs entre deux éléments de formes simples (i.e. rondes, carrées, rectangulaires...) étaient limités en nombre. Ils étaient encore plus réduits si ces éléments devaient être orientés suivant le sens du matériau (bois, tissus...). En revanche, lorsque ces éléments sont de formes quelconques, le nombre de positionnement peut être illimité : tous les placements d'un élément par rapport à l'autre doit être envisagés. C'est-à-dire qu'un élément peut être positionné d'une manière quelconque sur la périphérie de l'autre par translation, rotation et même emboîtement (cf. figure II.8). Le problème étant plus complexe, les méthodes employées pour résoudre ce type de problèmes nécessitent des moyens informatiques importants, en puissance de calcul et capacité mémoire.

L'adaptation des méthodes rectangulaires aux formes complexes pose d'énormes problèmes. Delaporte [DEL 17] contourne ce problème en utilisant la notion de **modules rectangulaires**. Un module est constitué d'un nombre réduit de formes complexes associées de façon à obtenir une forme rectangulaire ayant un taux de chute acceptable. La construction de modules est de ce fait une réduction du problème général d'optimisation de découpe de formes complexes. Cette méthode a été appliquée dans la confection.

4.2.1. Méthode d'Elomri

La découpe de vêtements ($\Re^2$) aux formes quelconques dans les ***industries de confection*** est différente des problèmes de découpe que l'on rencontre généralement dans la littérature. Elomri [ELO 25] résout le problème du **bordereau de coupe** (terme utilisé dans la

confection pour les plans de coupe) en encadrant une fonction coût discontinue et linéaire par morceaux entre deux fonctions continues et linéaires. Ces fonctions prennent en compte :

- les coûts de la **matière utilisée** et de la **matière perdue** ;
- les coûts engendrés par le **temps de préparation**, le temps nécessaire pour étaler la matière (installation des coupons de tissus sur la machine), le temps pour évacuer les paquets de tissus découpés et le temps d'arrêt de la machine ;
- le coût engendré par le **temps pour "éclater" les paquets** (séparer les tissus lorsque les couleurs sont différentes en fin de découpe) ;
- le coût lié au **temps de changement de rouleau** de tissu.

La résolution du problème peut alors être obtenue en appliquant par deux fois l'algorithme de Gomory [GIL 33] aux deux fonctions économiques continues. Lorsque les valeurs des solutions retenues sont réelles, elles sont tronquées pour obtenir des valeurs entières.

4.2.2. Méthode de Moreau

Le problème pour Moreau consiste à découper des pièces de vêtements aux formes irrégulières dans des coupons de tissus rectangulaires et ce, en respectant le sens du fil et les motifs. Moreau [MOR 43] a résolut ce problème en employant une procédure inspirée des méthodes manuelles (cf. figure II.7).

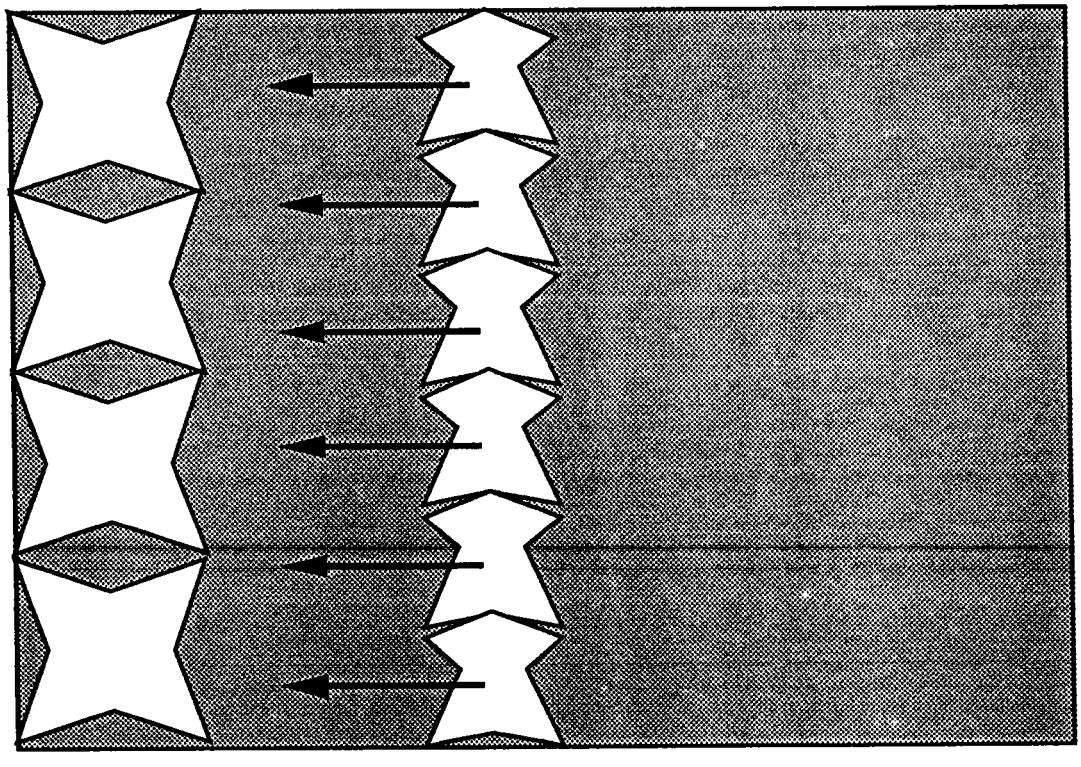

Figure II.7 : Méthode Moreau

Les pièces sont classées par ***surfaces décroissantes***. Les contours sont représentés par un ensemble de ***segments de droites***. La méthode consiste à emboîter les pièces à découper,

en colonnes, par translation de la nouvelle pièce. La nouvelle colonne réalisée est ensuite calée à côté des précédentes. Les emboîtements de pièces sont faits par translation horizontale.

Les autres types d'encastrement de pièces ne sont pas examinés (cf. figure II.8) tels ceux obtenus par translation verticale...

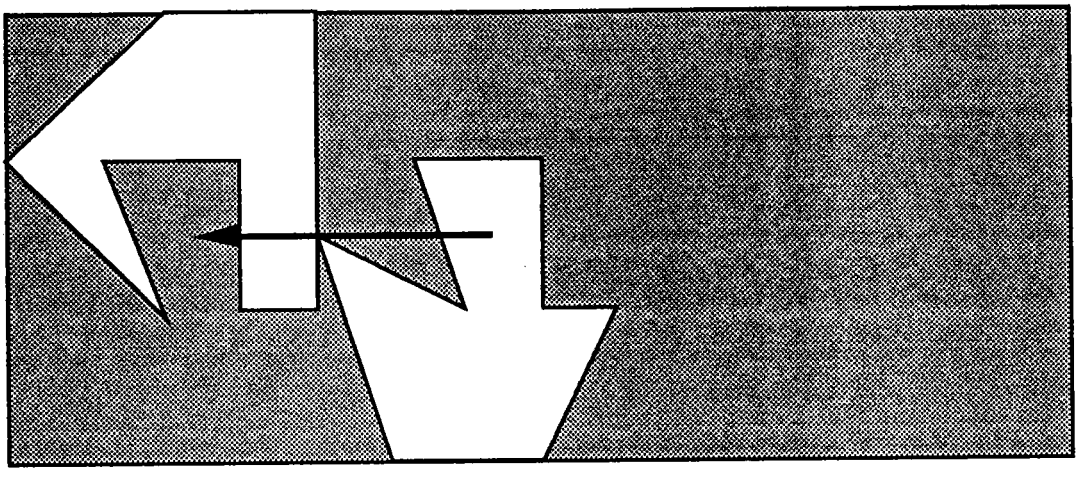

Figure II.8 : Problème d'encastrement

5. CONCLUSION

Différentes solutions de problèmes de découpe ont été fournies dans la littérature. Chen, Feiring et Cheng [CHE 9] tracent l'évolution des méthodologies utilisées pour résoudre les problèmes à deux dimensions : la programmation linéaire [GIL 32], la programmation dynamique [HER 38], les procédures par séparation et évaluation pour les pièces de même taille [AGR 1], les nouvelles techniques de l'intelligence artificielle [LIR 42], le recuit simulé [DOW 19] et le calcul parallèle [DAR 16].

Coffman et al [COF 12] et Johnson [JOH 39] exposent en détail les algorithmes pour les découpes à une et deux dimensions, et les généralisent aux dimensions supérieures. Ils donnent aussi le comportement des algorithmes simples (découpes en une dimension) dans les cas les plus défavorables.

Des exemples d'heuristiques de découpe en deux et en trois dimensions, sont données par Dowsland et al [DOW 18] comme le chargement de palettes *(pallet loading problem)* ou le remplissage de containers *(container stuffing)*.

Ce chapitre est destiné à situer les limites des méthodes existantes pour résoudre des problèmes de découpe. Dans le chapitre suivant, nous montrons que ces méthodes sont généralement insuffisantes pour répondre aux problèmes réels de découpe. Ceci est particulièrement vrai pour les problèmes posés par notre partenaire industriel.

Chapitre III :

PRÉSENTATION DU PROBLÈME

CHAPITRE III

1. INTRODUCTION

Ce chapitre introduit les méthodes [ANT 3] développées pour résoudre des problèmes de découpe unidimensionnelle rencontrés dans l'industrie. Pour cela, nous présentons d'abord ces problèmes dans les sections 2 et 3. Nous observons au passage qu'ils sont peu semblables aux problèmes rencontrés dans la littérature : en effet, les contraintes et les critères industriels liés à l'objet de ce travail ne sont pas, à notre connaissance, traités clairement dans la littérature.

2. DESCRIPTION DES PROBLÈMES INDUSTRIELS

Nous présentons ici les problèmes de découpe unidimensionnelle, c'est-à-dire les problèmes de découpe de produits longs, ou encore les problèmes multidimensionnels pouvant être ramenés à des problèmes de dimension 1 (découpes dans un seul sens de bobines, de moquettes ...). Les problèmes de découpe le plus souvent traités dans la littérature sont les découpes de barres ([MOR 43] et [COS 14]).

Nous décrivons plus précisément la découpe de barres d'acier. Ces méthodes peuvent s'appliquer à d'autres domaines.

Des barres d'acier (**barres mères**) sont entreposées en attendant d'être découpées en barres de plus petites tailles appelées **barres filles**. Il s'agit de déterminer un placement satisfaisant les commandes et minimisant les coûts engendrés par les découpes.

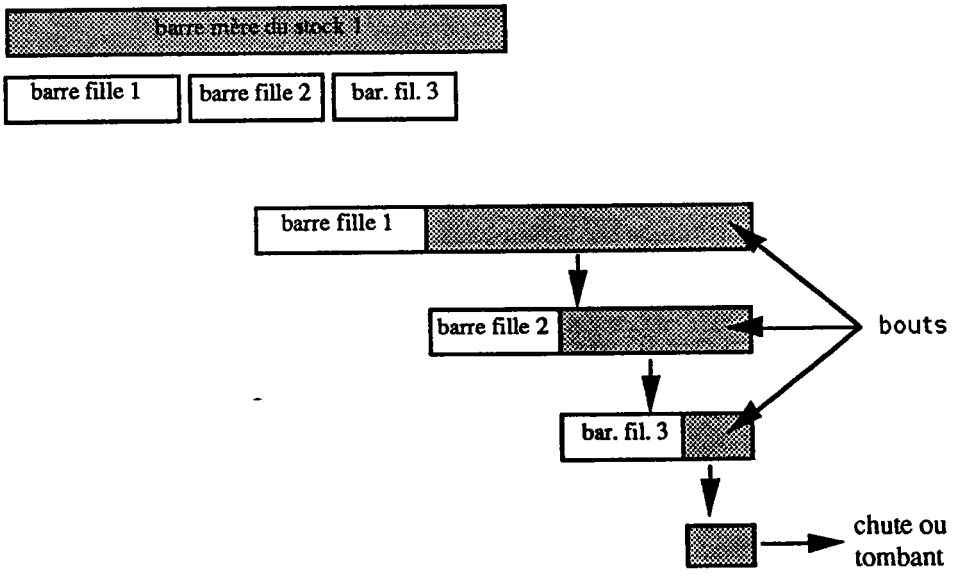

Figure III.1 : Représentation d'une découpe de barres filles dans une barre mère

Soient, par exemple, trois barres filles commandées. Elles sont découpées dans une barre mère de longueur plus importante. La partie restante non utilisée est appelée dans la suite **bout**

(cf. figure III.1). Sur ce bout, on placera à la suite la seconde barre fille, puis la troisième, jusqu'à atteindre un bout de longueur trop faible pour être réutilisable : la chute. Nous réservons le terme **chute** aux bouts non réutilisables.

Il existe, dans la littérature, différents termes techniques pour définir les problèmes industriels de découpe. Nous rappelons d'abord une notion bien connue concernant la coupe, puis nous ajoutons aux termes déjà connus dans la littérature de nouvelles notions rencontrées lors de notre étude.

2.1. Notion courante

Coupe

La notion de perte de matière due à l'outil de coupe n'intervient pas dans le cas des techniques de découpe non consommatrices de matière, comme les découpes par cisaillement. Dans d'autres cas, l'épaisseur enlevée est tellement faible qu'on peut ne pas la considérer. Cependant, dans de nombreux problèmes pratiques, la machine utilisée pour la coupe enlève une largeur de matière qui ne peut pas être négligée. Ceci est vrai pour les découpes à la scie, au laser, au chalumeau, au jet d'eau ou par oxycoupage... Un choix judicieux des placements et des positionnements des barres filles sur des barres mères peut entraîner la ***diminution du nombre de coupes***, et par conséquent une ***diminution de la perte de matière.***

L'exemple suivant permet d'illustrer nos propos. Ici, un trait de scie enlève 0,01 m de matière.

Nous disposons en stock d'une barre mère de 10,09 m et de 5 barres mères de 2,01 m. Les commandes à réaliser sont de 10 barres de 1 m. Deux solutions sont envisageables qui conduisent toutes deux à une chute nulle mais génèrent des pertes de matière différentes :

1. *les 10 barres commandées sont réalisées dans la barre mère de 10,09 m. Il faut couper 9 fois la barre mère. La perte de matière totale, due aux traits de scie, est donc de* ***0,09 m*** *;*
2. *les 10 barres commandées sont réalisées dans les 5 barres mères de 2,01 m. Chacune des barres mères est coupée en deux. Il ne faut plus que 5 coupes pour réaliser toute la commande. La chute totale, due aux traits de scie, est alors de* ***0,05 m.***

2.2. Nouvelles notions

Lors de notre étude nous avons constaté qu'il existait d'autres notions ou termes technologiques employés relevant de la découpe. Aussi il devenait nécessaire pour nous de les identifier de façon précise.

2.2.1. Tolérances et affranchissements

Le premier aspect est la tolérance. Il y a deux catégories de tolérances : la **tolérance réduite** et la **tolérance normale**. La tolérance réduite impose que les barres remises au client aient une longueur très proche de la longueur demandée et d'une qualité irréprochable (sans déformation et de composition uniforme).

En sortie de laminage, les extrémités de la barre produite sont soumises à des chocs thermiques plus rudes que les autres parties de la barre. Ainsi, il sera exigé d'éliminer les extrémités en cas de tolérance réduite (cf. figure III.2). Cette opération qui consiste à couper l'extrémité brute d'une barre mère pour éviter les imperfections est appelée **affranchissement**. Ces restrictions ne sont pas appliquées aux barres commandées à tolérance normale

Dans ce chapitre, nous considérons 2 barres commandées qui sont de même longueur mais de tolérances différentes comme des barres filles de types différents.

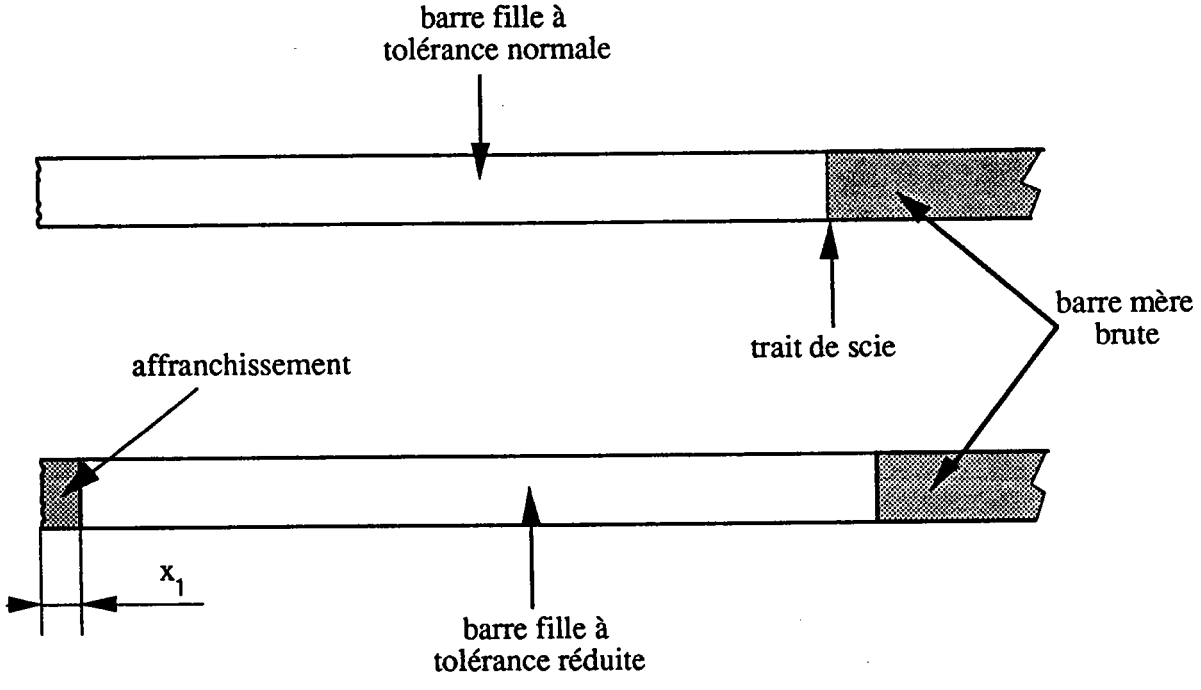

Figure III.2 : Affranchissement à l'extrémité d'une barre mère brute

Dans la figure III.2, une barre de tolérance réduite est placée sur une barre mère : si la découpe est ajustée à gauche, on perdra systématiquement une longueur $x_1 = a$ (où **a** est la longueur minimale d'un affranchissement incluant la largeur du trait de scie). Bien entendu, la partie de la barre située à droite de la découpe est d'une longueur quelconque, mais supérieure **à a .**

Pour éviter de perdre de la matière et du temps en réalisant des affranchissements, il est intéressant de placer en bouts de barres mères des barres filles à tolérance normale.

2.2.2. Paquet

La troisième notion est la notion de **paquet**. Plusieurs barres mères identiques dans lesquelles sont effectuées les mêmes découpes de barres filles peuvent être sciées en même temps. Naturellement, nous avons un gain sur le temps de coupe. La taille d'un paquet est définie par le nombre de barres mères dans ce paquet. La figure III.3 présente un exemple de paquet contenant quatre barres mères.

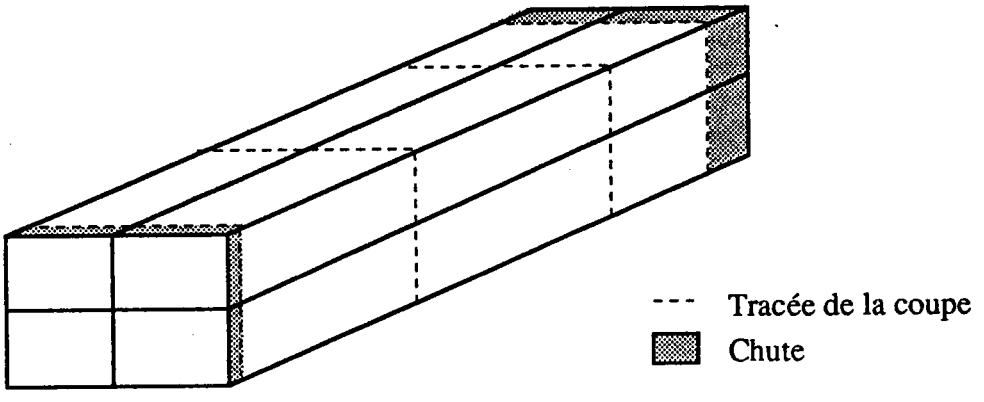

Figure III.3 : Constitution d'un paquet en prévision d'une découpe

2.2.3. Tombant

A la fin d'une découpe, si la partie restante d'une barre mère entamée est supérieure à une limite donnée **r**, on considère qu'elle est suffisamment longue pour pouvoir être réutilisée par la suite. Elle sera mise à côté de la machine de coupe (espace limité) ou remise en stock. On désigne cette partie de barres mères par **tombant**.

2.3. Problèmes abordés dans la littérature

Sous sa forme la plus simple, le problème de découpe s'écrit :

"Sachant que l'on dispose de barres de longueur **L**, et que l'on désire découper $\mathbf{n_1}$ barres de longueur $\mathbf{l_1}$, $\mathbf{n_2}$ barres de longueur $\mathbf{l_2}$, ..., $\mathbf{n_N}$ barres de longueur $\mathbf{l_N}$, comment procéder pour réaliser la demande avec un minimum de barres de longueur **L** ? (Ceci revient à minimiser les chutes). Bien entendu, $l_i < L$ pour $i = 1,2, ..., N$."

Barres mères de longueurs différentes

De nombreux algorithmes ne considèrent comme étant disponibles en stock que des barres mères de même longueur (telles les méthodes utilisant les règles de priorité de la section 3.1 du chapitre II). Seules les méthodes, issues de celles de Gilmore et Gomory ([GIL 32], [GIL 33]), qui font appel à la méthode des plans de coupe et à la programmation linéaire (cf. chapitre II, paragraphe 4.1) appréhendent la diversité des longueurs de barres disponibles en stock.

Bien entendu, le fait de ne considérer qu'une seule longueur réduit le champ des solutions et diminue la qualité du placement final.

Notons qu'il n'est pas restrictif de ne considérer qu'une seule longueur de barres mères lorsque le problème de découpe est appliqué à la pagination de la mémoire ou au disque d'un ordinateur. Ceci n'est plus le cas dans l'industrie de découpe de matière métallique. Les barres disponibles en stock sont généralement de ***différentes*** longueurs. Nous généralisons le problème de découpe au cas où l'on dispose de barres en stock (barres mères ou tombants) de longueurs différentes.

Surproduction

De nombreuses méthodes existantes ([GIL 32], [GIL 33], [COS 14], [COS 15], [MOR 43] et [DYC 22]) s'autorisent à faire de la surproduction : produire plus que ne l'exigent les commandes. Les barres produites en trop sont alors stockées en attendant une prochaine commande de même type, ou vendues avec les autres si le client les accepte. L'excédent réalisé peut aussi être considéré comme une chute [COS 14] lorsque le coût d'une barre est faible. Les approches sont alors très différentes, que l'on veuille ou non atteindre une production donnée. Dans notre étude, nous excluons toute surproduction, du fait qu'on peut se permettre de produire des tombants.

2.4. Problèmes industriels

Le critère le plus communément utilisé dans la littérature est la chute (suivant le cas, perte de matière, d'espace, de temps ou d'argent). Mais d'autres critères sont primordiaux pour les industriels : comme nous l'avons vu, nous devons prendre en compte les coûts ***génération et utilisation des tombants***. Dans les heuristiques existantes, les barres mères ne sortent qu'une seule fois du stock pour être découpées. Lorsque toutes les commandes ont été satisfaites, plutôt que de jeter les barres mères entamées, ces heuristiques génèrent du surplus en plaçant d'autres barres de mêmes dimensions que celles déjà réalisées pour réduire la chute. Ceci permet de réduire les chutes mais accroît les coupes non demandées.

En générant des tombants, les industriels produisent l'***exacte quantité demandée***, ce qui évite des pertes dues aux découpes supplémentaires, aux rabais de vente des surplus. Cependant, les tombants induisent un coût de stockage supplémentaire.

Le problème traité ici généralise les problèmes présentés dans la littérature en ce sens qu'il tient compte :

- des **longueurs différentes** des barres mères (et plus seulement des barres filles) ;
- des **affranchissements** à effectuer aux extrémités d'une barre mère avant de réaliser des barres filles à tolérance réduite ;
- du **coût de coupe** (fonction du nombre de barres et de paquets) ;

- du **coût d'installation** *(de réglages)* de chaque nouveau paquet sur la scieuse ;
- du **coût de perte de matière** liée aux traies de scie.

Modes d'utilisation

Pour la résolution de ces problèmes, nous devons développer des algorithmes qui favorisent tantôt le temps de calcul, tantôt la qualité de la solution. En effet, chez notre partenaire industriel, le placement des barres commandées sur les stocks s'opère en deux étapes.

La première est pratiquée lors de la passation de commandes par un client. L'agent commercial doit réaliser rapidement le placement des commandes sur les barres disponibles en stock (ou bientôt disponibles), pour vérifier que la commande est réalisable. Il établit alors le devis de ce placement pour le client. Le programme de découpe doit dans ce cas donner une solution réalisable de bonne qualité en un temps réduit. Nous qualifions cette étape d'étape ***temps réel***. Les commandes arrivent une à une dans la journée. Elles peuvent concerner un ou plusieurs profils, dimensions et qualités de matériaux. Pour chaque profil/qualité de barres correspond une ***référence*** de produit. Nous voyons que dans une journée, plusieurs commandes différentes peuvent concerner une même référence. Pour les distinguer dans chacune des références, chacun des groupes de commandes traitées en même temps est appelé ***affaire***.

La seconde étape consiste à réaliser, en fin de journée, les placements pour préparer les découpes du lendemain. Toutes les commandes reçues sont alors rassemblées et triées suivant les références et les qualités des matériaux. Nous savons qu'une solution admissible existe : c'est la concaténation des solutions établies par les vendeurs lors des prises de commandes. Cependant, en traitant globalement toutes les commandes de même profil en fin de journée, il est possible d'améliorer la solution précédente, et ainsi de réduire encore les frais. Dans ce cas, le programme de découpe n'est plus excessivement contraint au niveau du temps de traitement. En effet, le temps d'exécution, toutes affaires confondues (100 par jour), peut se prolonger toute la nuit. Il faut dans ce second placement rechercher le "meilleur" placement possible avec toutes les commandes réunies afin de préparer les fiches de travail pour le lendemain. Cette étape est appelée **planification** par la suite du chapitre.

3. ÉNONCÉ DU PROBLÈME

Dans la section précédente, nous avons donné une description globale du problème. Dans cette section, nous formalisons cette description. Pour cela, nous introduisons les éléments nécessaires à cette formalisation.

CHAPITRE III

3.1. Données

Le placement de barres filles sur des barres mères n'est possible que si elles correspondent à la même ***référence*** (profil/dimension/qualité). Sans perte de généralité, il n'est donc pas nécessaire d'optimiser simultanément toutes les références de barres. Par conséquent, nous considérons par la suite **affaire par affaire** (les commandes ponctuelles de la journée sont considérées référence par référence).

Le problème de découpe de stock s'énonce comme suit dans sa généralité. Pour une référence donnée de produit, **N** types de barres (barres mères) sont disponibles en stock. La longueur des barres mères de type **i** ($1 \leq i \leq N$) est L_i. Le nombre de barres de type **i** disponibles en stock est M_i. La variable T_i permet de préciser si une barre mère de type **i** est un tombant ou non :

$$T_i = \begin{cases} 1 & \text{si les barres mères de type i sont des tombants ;} \\ 0 & \text{sinon.} \end{cases}$$

La distinction entre les tombants et les autres barres du stock est importante lors des découpes de barres filles à tolérance réduite : le premier affranchissement n'est plus nécessaire.

Par extension, on peut concevoir des tombants découpés aux deux extrémités : une barre mère brute est retirée deux fois du stock pour réaliser deux barres filles à tolérance normale (une à chacune des extrémités). Cette distinction n'est pas nécessaire pour les raisons suivantes :

- Le fait que les extrémités d'une barre mère soient déjà coupées ***ne dispense pas*** de l'affranchissement en fin de barre mère sauf cas exceptionnel. En effet, les dimensions des barres filles à tolérance réduite sont très précises aussi il est peu probable que les dimensions des barres filles à découper tombent juste sur celle de la barre mère. Il faut noter aussi que les dimensions des barres en stock sont légèrement supérieures à celles connues au niveau informatique ;
- Cette distinction nécessiterait une ***nouvelle aire de stockage*** et, par conséquent, engendrerait des coûts supplémentaires de stockage.

L'ensemble $\{(L_i, T_i, M_i) / 1 \leq i \leq N\}$ décrit l'ensemble des barres en stock.

Les commandes des clients sont de **n** types différents. La longueur des barres filles de type **j** ($1 \leq j \leq n$) commandées est l_j. La tolérance des barres filles de type **j** est τ_j.

$$\tau_j = \begin{cases} 1 & \text{si les barres mères de type j sont à tolérance réduite ;} \\ 0 & \text{sinon.} \end{cases}$$

Pour chaque type **j** de barres filles, m_j est le nombre d'unités commandées.

$\{(l_j, \tau_j, m_j) / 1 \leq j \leq n\}$ décrit l'ensemble des barres à réaliser.

Notons qu'un sous-ensemble **B** de barres mères (resp. **C** de barres filles) peut être défini de manière unique par un vecteur de dimension **N** (resp. **n**). Le $i^{ième}$ élément de ce vecteur avec

$1 \leq i \leq N$ (resp. $1 \leq i \leq n$), représente le nombre de barres mères (resp. barres filles) dans ce sous-ensemble. Dans la suite de ce chapitre, un sous-ensemble de barres (B, C, ...) est désigné par des lettres majuscules. Quant aux éléments du vecteur, ils sont notés en lettres minuscules ($(b_1, ..., b_N)$, $(c_1, ..., c_n)$, ...).

Dans la pratique, il peut y avoir des **découpes biaisées**, c'est-à-dire que les coupes ne sont pas perpendiculaires à la longueur de la barre mais avec un certain angle par rapport à cette médiatrice (cf. figure III.4).

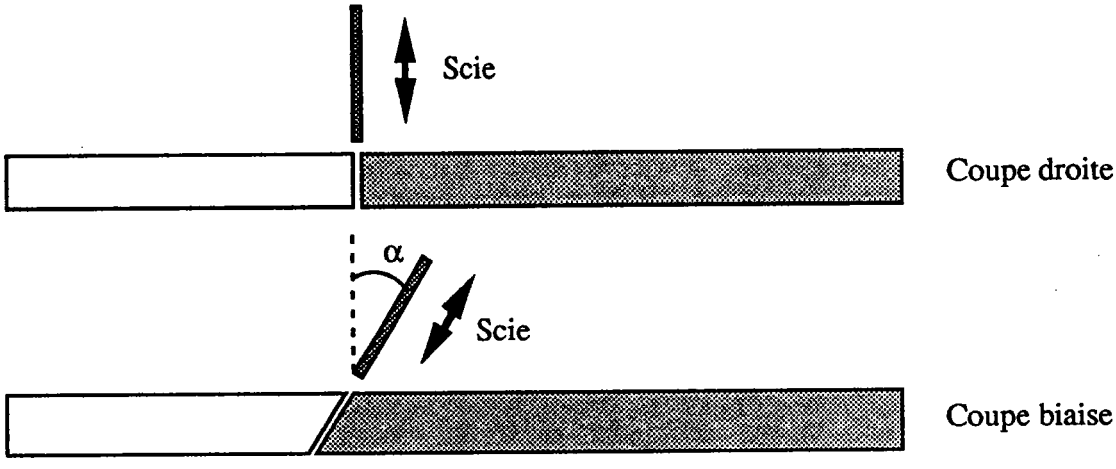

Figure III.4 : Coupe biaise

Ce genre de découpe n'est pas pris en compte dans notre étude, pour les deux raisons suivantes :

- d'abord les découpes avec biais sont ***peu fréquentes*** ;
- ensuite, elles peuvent être considérées comme des découpes sans biais : suite à une découpe biaisée, le tombant peut être ***affranchi de la partie biaisée*** ; pour l'optimisation de la découpe, on prendra en compte la longueur maximale de la barre biaisée.

3.2. Paramètres

Nous rappelons que pour nos problèmes de découpe en dimension un, il faut tenir compte de toutes les notions que nous avons évoquées à la section 2.

Les paramètres suivants sont à prendre en considération :

- **r** est la longueur minimale d'un tombant : en dessous de r, le bout est considéré comme une chute ;
- γ est la largeur d'un trait de scie ;
- **a** est la largeur d'un affranchissement (la largeur du trait de scie est incluse) ;
- n_s est la tolérance de sciage. Il s'agit du nombre de traits de scie qu'il est possible de ne pas comptabiliser lorsque l'on réalise les découpes à tolérance normale. Ceci est autorisé

car les longueurs réelles des barres en stock sont légèrement supérieures à celles connues au niveau informatique.

3.3. Coûts

Définissons à présent les critères entrant dans le choix des placements. Le coût est décomposé en quatre parties : les coûts des chutes, les coûts résultant de la préparation des paquets, les coûts induits par la durée des coupes et les coûts générés par les tombants. Sans perte de généralité, on suppose que les coûts correspondant aux différentes parties sont ramenés à la même unité : le coût d'une unité de longueur de chute. En normalisant, on peut ramener le coût d'une unité de longueur de chute à un.

3.3.1. Perte de matière : trait de scie et affranchissement éventuel

Les chutes entraînent une perte financière pour l'entreprise. Le coût d'une chute de longueur λ est normalisé et égal à λ. Les autres coûts sont ramenés à l'équivalent en longueur de chute.

3.3.2. Préparation d'un paquet

Chaque nouveau paquet à découper occasionne un coût correspondant au temps nécessaire pour **positionner les barres**, pour **cercler les barres** en un seul paquet, et pour **installer le paquet** sur la machine. Le temps d'installation peut être considéré comme constant quelle que soit la taille du paquet et quelle que soit la référence. Soit π le coût moyen de mise en place d'un paquet sur la scieuse. Le coût de préparation de l'ensemble des barres à découper est proportionnel au nombre de paquets à découper : si le nombre de paquets à réaliser est faible, le coût correspondant est faible.

3.3.3. Durée de coupe

La durée d'une coupe dans un paquet dépend de la **forme**, des **dimensions** et de la **matière** des barres ou des tubes à découper, ainsi que de leur nombre dans ce paquet. Pour une référence donnée, ce coût ne dépend que de la taille du paquet. Cette fonction est connue et elle est définie par p(s) pour un paquet de taille **s** :

$$\begin{cases} p(0) = 0 & \text{si } s = 0 ; \\ p(s) > 0 & \text{si } s \in \mathbb{N}^* . \end{cases}$$

Il est possible que certaines tailles de paquets ne soient pas réalisables pour des raisons techniques (stabilité et capacité). Par exemple, pour une raison de stabilité, il n'est pas possible de mettre 5 barres mères au lieu de 4 dans le paquet de la figure III.3 (au chapitre III, paragraphe 2.2.2). Les capacités en hauteur, en largeur et/ou en poids sont également limitées. Ainsi, lorsqu'un paquet de taille s n'est pas réalisable, on lui affecte un coût de découpe infini, i.e. $p(s) = +\infty$.

Notons que les coûts de coupe pour les tailles réalisables s satisfont la relation suivante :

$$p(s+s') \leq p(s) + p(s') \qquad \text{(III.1)}$$

où s, s' et (s+s') sont des tailles réalisables de paquets.

En d'autres termes, découper un paquet est moins coûteux que de le séparer en deux paquets plus petits que l'on coupe en séquence.

3.3.4. Génération de tombants

Nous devons prendre en compte les problèmes liés à la génération de tombants. Le fait de considérer les tombants peut mener à la situation indésirable suivante : si le stock est très important, la solution optimale consisterait à couper chaque barre fille dans une barre mère différente, de longueur importante puisque cela conduit à des tombants, donc à une chute nulle ! Cette situation doit être évitée, pour les raisons suivantes :

1. Le **coût de manutention** augmente : de nombreuses barres mères doivent être déplacées jusqu'à la scieuse. De plus, les parties restantes (tombants) doivent être transportées vers la réserve où elles pourront être stockées pour des commandes ultérieures.
2. Puisque les tombants ne sont pas de longueur standard, ils sont souvent regroupés dans une aire spéciale limitée en capacité de stockage. De plus, on ne sait pas quand ces tombants seront utilisés (par exemple pour des demandes occasionnelles de barres de ce profil et de cette qualité de matériaux). Ceci implique que la production de tombants augmente le **capital immobilisé** et la **gestion du stock**, ce qui entraîne des coûts supplémentaires.
3. L'existence d'un tombant n'exclut pas l'apparition d'une **chute** dans le futur.

Par conséquent nous définissons une fonction coût c(t) où t est la longueur du bout (partie restante) de la barre mère :

$$c(t) = \begin{cases} t & \text{si } 0 \leq t < r ; \\ \alpha + \beta t & \text{si } r \leq t \leq L ; \end{cases} \qquad \text{(III.2)}$$

où r est la longueur minimale d'un tombant et L la longueur de la barre mère.

Lorsque la longueur t du bout est inférieure à la longueur minimale r d'un tombant, le bout n'est pas conservé : nous avons une chute. La perte de matière occasionne un coût égal à la longueur de la chute.

Si le bout est de longueur au moins égale à r, il est considéré comme un tombant. Nous donnons au tombant de longueur t un coût $\alpha + \beta t$. Le paramètre α représente le coût de transport et le coût de stockage du tombant. Le paramètre β représente une estimation du coût (par unité de longueur) engendré par la chute générée lorsque le tombant sera utilisé et le coût du capital immobilisé.

Notons que la fonction coût doit avoir la propriété suivante :

$$c(r^+) \leq c(r^-). \tag{III.3}$$

Cette relation traduit l'intérêt pour l'entreprise de créeer des tombants plutôt que de faire une chute importante. Elle signifie que le coût associé au plus petit des tombants est inférieur au coût associé à la plus grande des chutes. Le coût de la partie restante de la barre mère suit la figure III.5.

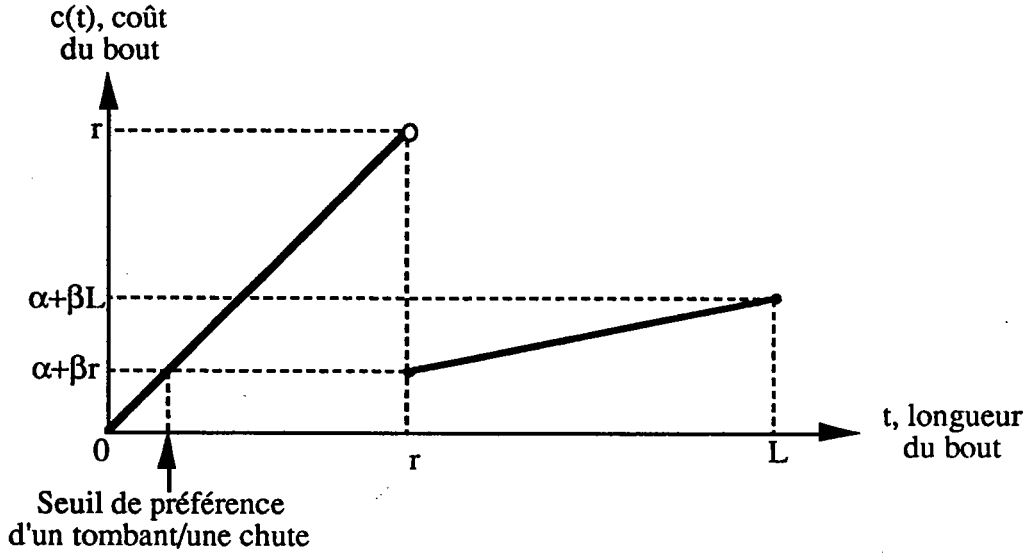

Figure III.5 : Coût du bout

Pour résumer, si on réalise une découpe d'un paquet de taille s, en effectuant **d** coupes, avec une chute (autre que le bout restant) **w** et une partie restante de longueur **t**, le coût correspondant à ce paquet est :

$$\sigma(w,t,s) = s \times (w + c(t)) + \pi + d \times p(s) \tag{III.4}$$

où :

- π est le coût de préparation d'un paquet ;
- **p(s)** est le coût d'une coupe dans un paquet de taille s.

3.3.5. Utilisation des tombants

Un tombant est réutilisable parce qu'il est suffisamment long. En revanche, cette matière génère un coût de stockage. Ce coût n'est actuellement pas chiffrable. Nous introduisons donc un paramètre pour favoriser l'utilisation des tombants lors du calcul des placements. La fonction coût est alors pondérée par le paramètre θ favorisant l'utilisation des tombants. Le paramètre θ est compris entre 0 et 1 lorsqu'il s'agit d'une découpe dans un tombant, il est égal à 1 lorsque la barre mère est brute (la barre mère n'a jamais été découpée). Plus le nombre θ est petit, plus le placement sur un tombant est privilégié.

S'il s'agit d'un paquet de tombants, on a :

$$\sigma(w,t,s) = \theta \times \{s \times (w + c(t)) + \pi + d \times p(s)\} \qquad \text{(III.5)}$$

Cependant, cette option ne pourra être utile au gestionnaire de stock que lorsque les tombants seront identifiés au niveau informatique.

Notons que ceci n'est pas une tâche très difficile. En effet, il suffit de faire un inventaire une seule fois pour tous les tombants en stock. La suite est gérée automatiquement par notre programme de découpe, en supposant bien entendu que les consignes données par le programme soient suivies.

3.4. Contraintes

La faisabilité d'un placement d'un ensemble E de barres filles sur une barre mère est directement liée au ***nombre d'affranchissements*** à effectuer sur ce placement. Ce nombre dépend du nombre de barres filles à tolérance réduite et du nombre de barres filles à tolérance normale de l'ensemble E.

Soit $\Phi(E)$ (respectivement $\Psi(E)$) le nombre de barres filles à tolérance normale (respectivement réduite) de E. Donc :

$$\Phi(E) = \sum_{j \in E} (1 - \tau_j) \text{ et } \Psi(E) = \sum_{j \in E} \tau_j$$

Nous examinons les trois cas suivants. Ils ont été définis en fonction du nombre d'affranchissements à réaliser.

3.4.1. Cas 1 : $\Phi(E) = 0$ et $\Psi(E) \geq 1$

Dans ce cas, seules des barres filles à tolérance réduite sont à placer. Les extrémités de la barre mère sont brutes. Afin de pouvoir réaliser par la suite des barres filles à tolérance réduite, nous coupons d'abord une longueur a à une extrémité de la barre mère (correspondant à un affranchissement). Les barres filles sont alors coupées une à une dans un ordre quelconque. A chaque coupe réalisée, nous enregistrons une perte de matière correspondant au trait de scie de largeur γ. Comme nous ne découpons que des barres filles à tolérance réduite, il faut laisser en fin de la barre mère une longueur au moins égale à la longueur d'un affranchissement.

Deux affranchissements sont nécessaires (réels ou virtuels s'il y a génération d'un tombant). Le placement est donc réalisable si et seulement si pour un ensemble E de barres filles à tolérance réduite, la longueur de la barre mère λ est au moins égale à la somme des longueurs des affranchissements, des barres filles et des traits de scie. Plus simplement, la condition à réaliser s'écrit :

$$\lambda \geq 2a - \gamma + \sum_{j \in E} (l_j + \gamma) \qquad \text{(III.6)}$$

où :

- **a** est la largeur de l'affranchissement augmentée du trait de scie ;
- γ est la largeur d'un trait de scie ;
- **E** est l'ensemble des barres filles placées sur la barre mère de longueur λ ;
- l_j est la longueur de la barre fille de type **j**.

3.4.2. Cas 2 : $\Psi(E) = 0$ ou $\Phi(E) \geq 2$

Dans ce cas, les affranchissements ne sont pas nécessaires puisqu'on peut placer une barre fille à tolérance normale à une des extrémités de la barre mère. La condition de faisabilité du placement est :

$$\lambda \geq -\gamma + \sum_{j \in E}(l_j + \gamma) - \gamma \times \min(n_s, \Phi(E) - 1)$$

où n_s est le nombre maximal de traits de scie qu'il est possible d'ignorer lorsqu'il s'agit de découpe de barres filles à tolérance normale ;

ou plus simplement :

$$\lambda \geq \sum_{i \in E}(l_i + \gamma) - \gamma \times \min(n_s + 1, \Phi(E)) \qquad \text{(III.7)}$$

3.4.3. Cas 3 : $\Phi(E) = 1$ et $\Psi(E) \geq 1$

Dans ce cas, l'affranchissement n'est pas nécessaire à l'une des extrémités de la barre mère car la barre fille à tolérance normale est placée à cette extrémité. En revanche, si c'est une barre fille à tolérance réduite qui est placée à l'autre extrémité, la partie restante doit alors être de longueur au moins égale à **a** (ce qui est toujours le cas s'il y a génération de tombant). Par conséquent, la condition pour que le placement soit réalisable est :

$$\lambda \geq a - \gamma + \sum_{j \in E}(l_j + \gamma) \qquad \text{(III.8)}$$

avec $\Phi(E) = 1 \Rightarrow \min(n_s, \Phi(E)-1) = 0$.

3.4.4. Cas où la barre mère est un tombant

Il faut noter que ce calcul est très facilement adaptable au cas où la barre mère de type **i** est un tombant ($T_i = 1$). En effet, il suffit alors de rajouter, dans chacun des sous-ensembles de **E**, une barre fille fictive à tolérance normale et de longueur nulle. On retombe alors soit dans le cas 2, soit dans le cas 3.

3.5. Fonction objectif

Pour terminer la description des problèmes étudiés, nous définissons la fonction objectif du problème de découpe à une dimension en intégrant les critères que nous avons précédemment définis.

En résumé, si :

- **P** est le nombre de paquets réalisés ;
- e_k est le type de barres mères utilisées dans le paquet **k** ;
- w_k est la chute dans chaque barre mère du paquet **k** ;
- t_k est la longueur de la partie restante de chaque barre mère du paquet **k** ;
- s_k est la taille du paquet **k** ;
- d_k est le nombre de coupes dans le paquet **k** ;
- $c(t_k)$ est coût de la partie restante t_k qui est soit une chute, soit un tombant ;
- θ est le paramètre favorisant l'utilisation des tombants ;
- T_{e_k} est la variable indiquant si la barre mère de type e_k est un tombant. Elle est égale à 1 s'il s'agit d'un tombant et à 0 sinon ;

alors le coût à minimiser est :

$$\chi = \sum_{k=1}^{P} \left[T_{e_k}\theta + \left(1 - T_{e_k}\right) \right] \left[s_k\left(w_k + c(t_k)\right) + \pi + d_k p(s_k) \right] \qquad \text{(III.9)}$$

Nous pouvons remarquer que w_k et t_k sont fonctions de e_k et C_k, ensemble des barres filles réalisées dans chaque barre mère du paquet k.

Pour mieux saisir les éléments entrant dans cette formulation, nous allons commenter les termes et les facteurs de cette formule :

- $T_{e_k}\theta$ est la valeur que prend le paramètre θ (valeur constante) en fonction du type de barres mères. Elle est égale à θ si la barre mère e_k est un tombant et à 0 sinon ;
- $1\text{-}T_{e_k}$ est le complément à 1 de T_{e_k}, c'est-à-dire $1\text{-}T_{e_k}$ est égale à 1 si la valeur de T_{e_k} est nulle et vice versa ;
- $\left[\theta\left(T_{e_k}\right) + \left(1\text{-}T_{e_k}\right)\right]$ est la pondération des coûts (chutes, tombants, installation et coupes) associés au type de barres mères e_k. Elle est égale à θ (avec $\theta \in [0,1]$) si la barre e_k est un tombant et à 1 sinon ;
- $\left[s_k\left(w_k + c(t_k)\right) + \pi + d_k p(s_k)\right]$ est le coût des chutes, le coût des tombants, le coût d'installation et le coût des coupes du paquet **k** (cf. relations (III.4) et (III.5)).

4. CONCLUSION

Dans ce chapitre, nous avons montré les différences entre les problèmes traités dans la littérature et le problème de notre partenaire industriel. Nous avons posé le problème de placement en termes de contraintes à respecter et de critères à optimiser.

Comme il est malaisé de résoudre le problème globalement, nous résolvons le problème en deux étapes. La première étape (chapitre IV) consiste à optimiser localement les placements : il s'agit de trouver le meilleur placement de barres filles sur une barre mère donnée. La seconde étape (chapitre V) utilise les résultats de la première pour résoudre globalement le problème de placement.

Chapitre IV :
RÉSOLUTION DES SOUS-PROBLÈMES
(placement de barres filles sur une barre mère)

CHAPITRE IV

1. INTRODUCTION

Dans cette section, nous nous focalisons sur la résolution de deux sous-problèmes du problème principal qui concernent tous deux le ***positionnement de barres filles sur une seule barre mère***. Nous verrons dans la suite comment la résolution du problème principal fait appel aux solutions de ces sous-problèmes.

Dans les deux sous-problèmes, il s'agit de choisir le meilleur placement possible d'un sous-ensemble C de barres filles sur une barre mère donnée. Le meilleur placement correspond au coût minimal dû à la chute et à la génération de tombant. Bien que de dimension plus réduite que le problème général, ce sous-problème est encore combinatoire car il est nécessaire de parcourir l'ensemble des possibilités de placement des barres filles sur la barre mère.

Nous pouvons distinguer deux types de sous-problèmes. Le premier consiste à chercher, pour une barre mère donnée, la meilleure combinaison de barres filles optimisant la fonction objectif définie par la relation (III.9) du chapitre III. Le second sous-problème ressemble au premier, à la différence près que ***nous fixons au départ le nombre de barres filles à placer*** sur la barre mère.

Rappelons que dans la résolution de ces sous-problèmes, on ne considère pas pour le moment les coûts dus à la préparation de paquets et au temps de coupe. En effet, comme on ne considère qu'une seule barre mère, les coûts sont quasiment constants, c'est-à-dire idépendants du placement des barres filles sur cette barre mère donnée. Nous ne tiendrons compte de ces coûts que lors de la résolution du problème général.

Pour pouvoir choisir un sous-ensemble qui minimise le coût, nous devons être capables d'évaluer le coût pour tous les sous-ensembles extraits de l'ensemble des barres filles commandées.

2. CALCUL DU COÛT DES CHUTES ET DU TOMBANT

Soit $\rho(\lambda, E)$ le coût induit par les chutes et le tombant lorsqu'on affecte un sous-ensemble E donné de barres filles à une barre mère de longueur λ. Nous montrons d'abord comment calculer $\rho(\lambda, E)$, puis nous indiquons comment résoudre les sous-problèmes susmentionnés.

Pour un placement local donné, lorsque toutes les contraintes (cf. chapitre III, section 3.4) ne sont pas respectées, nous lui affectons un coût infini.

La valeur prise par $\rho(\lambda, E)$ est étroitement liée au nombre de barres filles à tolérance réduite et au nombre de barres filles à tolérance normale de l'ensemble E.

Rappelons que $\Phi(E)$ (respectivement $\Psi(E)$) est le nombre de barres filles à tolérance normale (respectivement réduite) de E, avec :

$$\Phi(E) = \sum_{j \in E} \left(1 - \tau_j\right) \text{ et } \Psi(E) = \sum_{j \in E} \tau_j .$$

Considérons les trois cas suivants qui se distinguent par le nombre d'affranchissements à réaliser dans le placement local considéré.

2.1. Cas 1 : $\Phi(E) = 0$ et $\Psi(E) \geq 1$

Si la condition (III.6) est vérifiée, la partie restante de la barre mère, après réalisation de l'affranchissement et des barres filles, est alors de longueur $(\lambda - a - \sum_{j \in E} (l_j + \gamma))$. Cette partie est soit une chute, soit un tombant.

Par conséquent, dans ce cas, nous avons :

$$\rho(\lambda, E) = \begin{cases} +\infty & \text{si } \lambda < 2a - \gamma + \sum_{j \in E} (l_j + \gamma) ; \\ a + |E|\gamma + c\left(\lambda - a - |E|\gamma - \sum_{j \in E} (l_j)\right) & \text{sinon ;} \end{cases} \qquad \text{(IV.1)}$$

où c(t) est le coût du tombant de longueur t si t est plus long que la longueur minimale d'un tombant, sinon c(t) est le coût d'une chute de longueur t.

Rappelons que les coûts sont exprimés en unités de longueur de chute (voir chapitre III, paragraphe 3.3).

De la même manière, nous calculons $\rho(\lambda, E)$ pour les autres cas.

2.2. Cas 2 : $\Psi(E) = 0$ ou $\Phi(E) \geq 2$

Nous pouvons déduire de la relation (III.7) que la partie non utilisée de la barre mère est de longueur :

$$\lambda - \sum_{i \in E} (l_i + \gamma) + \gamma \times \min(n_s + 1, \Phi(E)).$$

La chute due aux coupes (hors la partie restante de la barre) est alors $\gamma\left(|E| - \min(n_s, \Phi(E) - 1)\right)$.

Par conséquent, le coût du placement local est :

$$\rho(\lambda,E)=\begin{cases}+\infty & \text{si } \lambda < \left(\begin{array}{l}\sum_{i\in E}(l_i+\gamma)\\ -\gamma\times\min(n_s+1,\Phi(E))\end{array}\right);\\ \left(\begin{array}{l}\gamma\left(|E|-\min(n_s,\Phi(E)-1)\right)\\ +c\left(\begin{array}{l}\lambda-\sum_{i\in E}(l_i+\gamma)\\ +\gamma\times\min(n_s+1,\Phi(E))\end{array}\right)\end{array}\right) & \text{sinon.}\end{cases} \qquad \text{(IV.2)}$$

2.3. Cas 3 : $\Phi(E) = 1$ et $\Psi(E) \geq 1$

D'après la condition (III.8), la chute générée est de longueur $a+\gamma(|E|-1)$.

Le coût de la découpe du sous-ensemble **E** de barres filles dans la barre mère de longueur λ est :

$$\rho(\lambda,E)=\begin{cases}+\infty & \text{si } \lambda < \left(a+\sum_{j\in E}(l_j+\gamma)-\gamma\right);\\ \left(\begin{array}{l}a+\gamma(|E|-1)\\ +c\left(\lambda-a-\sum_{j\in E}(l_j+\gamma)+\gamma\right)\end{array}\right) & \text{sinon.}\end{cases} \qquad \text{(IV.3)}$$

2.1.4. Cas où la barre mère est un tombant

Comme pour la section 3.4 du chapitre III, le calcul est très facilement adaptable au cas où la barre mère de type **i** est un tombant ($T_i = 1$). Il suffit d'ajouter dans chacun des sous-ensembles de **E**, une barre fille fictive à tolérance normale et de longueur nulle. Le cas 1 n'est alors plus utile pour le calcul de $\rho(\lambda,E)$: on retombe soit dans le cas 2, soit dans le cas 3.

3. PREMIER SOUS-PROBLÈME (placement sur une barre mère sans spécification du nombre de barres filles)

3.1. Un algorithme pseudo-polynomial

Définissons ce qu'est un algorithme pseudo-polynomial avant de présenter le sous-problème et sa résolution.

3.1.1. Définition d'un algorithme pseudo-polynomial

Pour définir ce terme, et par là même éclaircir les termes employés tout au long de ce travail, nous allons présenter quelques notions de complexité.

La complexité ([CHU 10] et [GAR 28]) s'intéresse au temps nécessaire pour obtenir une solution (on peut s'intéresser également à la mémoire nécessaire). On peut considérer la durée moyenne de traitement ou la durée dans le pire des cas. Nous traitons essentiellement ce dernier cas.

La théorie de la **complexité** conduit à distinguer entre la complexité des problèmes et la complexité des algorithmes utilisés pour les résoudre.

Pour obtenir la complexité d'un algorithme, il faut exprimer sa durée en fonction de la taille, notée **n**, de l'énoncé du problème, **f(n)** (par exemple dans notre cas, le nombre de barres filles commandées). Nous dirons qu'une fonction f(n) est en O(g(n)) lorsqu'il existe une constante **c** et une fonction **g** tel que $|f(n)| \leq c.|g(n)|$ pour toutes valeurs de $n \geq 0$. La fonction complexité d'un algorithme est la "meilleure" fonction g(n) que l'on puisse trouver pour majorer **f**.

Si **f** est en O(g(n)) et si g(n) est un polynôme, on dit que l'algorithme est ***polynomial*** et il est d'autant plus complexe que le degré du polynôme est grand. Lorsque le polynôme **g** n'est pas seulement fonction de la taille du problème (n dans notre exemple) mais aussi d'un ou de plusieurs paramètres du problème (par exemple la longueur L_i de la barre mère de type **i**), i.e. $g(n,L_i)$, l'algorithme est alors dit ***pseudo-polynomial***. Les algorithmes pour lesquels il n'existe pas de fonction **g** polynomiale telle que **f** soit en O(g(n)) sont dits ***exponentiels*** (on le démontre en minorant **f** par une fonction exponentielle de **n**).

Voyons maintenant la notion de complexité d'un problème. Un problème peut être résolu par différents algorithmes et, en général, on n'est pas sûr d'avoir trouvé l'algorithme de moindre complexité. L'analyse de la complexité des problèmes consiste à les ranger dans des classes de problèmes de complexité «analogue».

On distingue la classe de problèmes de décision dont le résultat s'exprime par un booléen «vrai» ou «faux» et la classe de problèmes d'optimisation (où l'on cherche une solution optimale dans un domaine donné). Les problèmes d'optimisation sont plus difficiles que les problèmes de décision de même nature, comme nous l'expliquerons plus loin.

Considérons tout d'abord les problèmes de décision.

Il existe des listes de problèmes déjà démontrés comme étant NP-complets [GAR 28]. Pour montrer qu'un problème est **NP-complet**, il suffit de montrer qu'il est au moins aussi difficile que l'un d'entre eux.

De même, il existe des listes de problèmes polynomiaux. Pour montrer qu'un problème est polynomial, il suffit de montrer que l'un quelconque d'entre eux est au moins aussi difficile que lui (ou de savoir le résoudre par un algorithme polynomial).

Actuellement, on n'a pas encore démontré qu'il n'existe pas d'algorithmes polynomiaux pour résoudre les problèmes NP-complets. Tant que l'on n'a pas prouvré qu'un problème NP-complet peut être résolu par un algorithme polynomial, on affirme sans aucune preuve que les problèmes NP-complets qui lui sont associés ne peuvent probablement pas être résolus par des algorithmes polynomiaux.

On peut associer à tout problème d'optimisation un problème de décision :

1. le problème d'optimisation est de la forme :
   "Chercher un élément **S** d'un domaine **D** qui maximise une fonction F(S)."
2. le problème de décision de même nature est de la forme :
   "Soit **K** un paramètre. Existe-t-il une solution $S \in D$ telle que $F(S) \geq K$?"

La solution pour le problème de décision est :

- si $F(S) \geq K$ alors vrai ;
- si $F(S) < K$ alors faux.

Dans le problème de maximisation, la plus grande valeur K^* est obtenue pour l'élément S^* de **D**, c'est-à-dire $K^* = f(S^*) \geq f(S)$. Pour obtenir cette solution optimale, en utilisant la résolution du problème de décision, nous pouvons effectuer une recherche dichotomique sur l'intervalle de définition de **K**.

Soit $[K_A, K_B]$ l'intervalle de définition de **K**. L'algorithme pour obtenir la solution optimale K^* est la suivante (on suppose que $F(S) \geq K_1$) :

ALGORITHME DE DICHOTOMIE

1. Initialiser

$K_1 = K_A$ *et* $K_2 = K_B$

2. Tant que $|K_1 - K_2| > \varepsilon$

2.1. $K = \dfrac{K_1 + K_2}{2}$

2.2. $\exists$ *une solution S du problème de décision à paramètre K*

(i.e. $\exists S \in D$ *telle que* $F(S) \geq K$*)*

2.2.1. Si oui

$K_1 = K$

2.2.2. Si non

$K_2 = K$

3. Fin tant que

4. $K^* = K$ et $S^* = S$.

On dit qu'un problème d'optimisation est **NP-difficile** si le problème de décision associé est NP-complet. Un problème d'optimisation sera polynomial si on peut le résoudre par un algorithme polynomial.

La notion de complexité permet d'informer les chercheurs et leur évite de perdre du temps à essayer de construire un algorithme polynomial pour obtenir une solution optimale alors que le problème a été démontré NP-difficile.

3.1.2. Position du problème

Ce premier sous-problème consiste à affecter un sous-ensemble de barres filles de l'ensemble C à une barre mère donnée de longueur L afin de minimiser le coût induit par les chutes et le tombant. Nous posons, sans perte de généralité et par souci de commodité, que toutes les barres filles sont de types différents et que les types de barres filles sont numérotés de 1 à $|C|$. Les variables l_i et τ_i indiquent respectivement la longueur et la tolérance de la barre fille de type i.

Sans restreindre la généralité, nous supposons que les barres filles candidates au placement constituant l'ensemble C sont de types différents, et qu'elles sont numérotées de telle sorte que $\tau_i \geq \tau_{i+1}$ pour tout $1 \leq i < |C|$. Plus simplement, les barres filles à tolérance réduite sont en début de liste.

3.1.3. Résolution du problème

Nous montrons d'abord que ce problème combinatoire peut-être résolu par un algorithme pseudo-polynomial.

Si l'on ne tient pas compte du type de tolérance, d'affranchissement et de tombant, Goulimis [GOU 36] a démontré que le problème qui consiste à placer des barres filles (avec un nombre de barres placées non fixe) sur une barre mère peut-être résolu en un temps pseudo-polynomial en utilisant une approche de type programmation dynamique. Nous montrons que ceci est possible même en tenant compte des tolérances, des affranchissements et des tombants.

Nous procéderons en examinant le placement éventuel d'une barre fille supplémentaire sur la barre mère, celle-ci étant partiellement utilisée. Pour cela, nous examinerons les barres filles les unes après les autres.

Pour chaque sous-ensemble dont l'élément de plus grand indice est j, c'est-à-dire $\{1,2,...,j\}$, nous étudierons tous les cas de figures résultant du placement d'un des sous-ensembles de barres filles de $\{1,2,...,j\}$ sur une barre mère de longueur λ.

Appelons E un sous-ensemble de $\{1,2,...,j\}$ de barres filles destinées à être découpées dans la barre mère. Les découpes dans une barre mère sont caractérisées par les affranchissements à effectuer (ou non) à ses extrémités.

3.1.3.1. Calcul du placement de coût minimal d'un sous-ensemble de C de barres filles sur une barre mère

Rappelons que C est l'ensemble des barres filles candidates au placement. Les barres filles sont supposées toutes différentes et numérotées de 1 à $|C|$. Les barres filles à tolérance réduite sont classées en début de liste et leurs indices sont inférieurs aux indices des barres filles à tolérance normale. Définissons $S_{j,q}$ comme la collection de tous les sous-ensembles de $\{1,2,...,j\}$ ($j \le |C|$) tels que le nombre d'affranchissements à effectuer est q ($0 \le q \le 2$).

Définissons la fonction $f(j,\lambda,q)$ comme le coût minimal occasionné par le placement d'un sous-ensemble de l'ensemble $\{1,2,...,j\}$ de barres filles candidates au placement sur une barre mère de longueur λ et qui nécessite q affranchissements. En d'autres termes, le meilleur coût $f(j,\lambda,q)$ est calculé de la manière suivante :

$$f(j,\lambda,q) = \min_{\substack{E \in S_{j,q} \\ E \neq \varnothing}} \rho(\lambda,E) \qquad \text{(IV.4)}$$

Rappelons que $\rho(\lambda,E)$ est le coût provenant des chutes et des tombants lorsqu'on affecte un sous-ensemble E donné de barres filles à une barre mère de longueur λ.

Les formules de récurrence sont définies suivant le nombre d'affranchissements à réaliser.

a. Formule à appliquer lorsqu'aucun affranchissement n'est requis

Le coût du placement local ne nécessitant pas d'affranchissement examine quatre cas possibles (+ un cas sans solution) :

$$f(j,\lambda,0) = \begin{cases} +\infty & \text{si} \quad \tau_j = 1 ; \\ \min\left\{ \begin{matrix} f(j-1,\lambda,0), f\left(j-1,\lambda-l_j-\gamma,0\right), \\ \nu(\lambda,j), f\left(j-1,\lambda-l_j-\gamma+a,1\right)-a+\gamma \end{matrix} \right\} & \text{sinon ;} \end{cases} \qquad \text{(IV.5)}$$

où :

- $+\infty$ est le coût correspondant au cas où la barre fille de type j est à tolérance réduite. L'ensemble $\{1,2,...,j\}$ ne contient donc que des barres filles à tolérance réduite. Leurs découpes dans la barre mère brute de longueur λ nécessitent deux affranchissements, ce qui est incompatible avec l'hypothèse $q = 0$;

- **f(j-1,λ,0)** est le coût correspondant au cas où les barres filles à découper dans la barre mère de longueur λ sont choisies dans l'ensemble {1,2,...,j-1} et où aucun affranchissement n'est nécessaire ;
- **f(j-1,λ-l$_j$-γ,0)** est le coût correspondant au cas où la barre fille de type **j**, qui est à tolérance normale, est incluse dans l'ensemble des barres filles à découper dans la barre mère de longueur λ, le complément étant choisi dans l'ensemble {1,2,...,j-1} de telle sorte qu'il ne nécessite aucun affranchissement ;
- **v(λ,j)** est le coût correspondant au cas où la barre fille de type **j** est à tolérance normale et qu'elle est la seule à être découpée dans la barre mère brute de longueur λ (le dernier trait de scie est inutile si le reste de la barre mère est de longueur inférieure à la largeur **a** d'un affranchissement). Ce coût s'écrit :

$$v(\lambda,j)=\begin{cases}+\infty, & \text{si } \lambda < l_j ;\\ c\left(\lambda - l_j - \gamma\right) & \text{sinon ;}\end{cases} \qquad \text{(IV.6)}$$

- enfin, **f(j-1,λ-l$_j$-γ+a,1)-a+γ** correspond au cas où la barre fille de type **j**, qui est à tolérance normale, est incluse dans l'ensemble des barres filles à découper, mais où le complément est choisi dans {1,2,...,j-1} de telle sorte qu'un affranchissement soit nécessaire. En effet, le terme -a+γ est la modification du coût due à l'introduction de la barre fille de type **j** qui permet d'économiser un affranchissement mais qui rajoute un trait de scie.

Il est à noter qu'il est impossible de placer la barre fille de type **j** et de choisir le complément parmi {1,2,...,j-1} de telle sorte que deux affranchissements soient nécessaires. En effet, ceci conduirait à un placement qui nécessite au moins un affranchissement, ce qui est en contradiction avec l'hypothèse q = 0.

b. Formule à appliquer lorsqu'un affranchissement est requis

Pour aboutir à un placement local ne réalisant qu'un seul affranchisssement, deux cas sont envisageables (+ un cas sans solution) :

$$f(j,\lambda,1)=\begin{cases}+\infty & \text{si } \tau_j = 1 ;\\ \min\begin{cases}f(j-1,\lambda,1), \\ f\left(j-1,\lambda-l_j-\gamma+a,2\right)-a+\gamma\end{cases} & \text{sinon ;}\end{cases} \qquad \text{(IV.7)}$$

où :

- +∞ est le coût correspondant au cas où la barre fille de type **j** est à tolérance réduite. L'ensemble {1,2,...,j} est uniquement constitué de barres filles à tolérance réduite. Le

placement d'un sous-ensemble de {1,2,...,j} entraîne l'ajout de deux affranchissements, ce qui est en désaccord avec l'hypothèse q = 1 ;

- **f(j-1,λ,1)** est le coût correspondant au cas où les barres filles à découper dans la barre mère de longueur λ sont choisies dans l'ensemble {1,2,...,j-1} et où un affranchissement est nécessaire ;
- **f(j-1,λ-l$_j$-γ+a,2)-a+γ** correspond au cas où la barre fille de type **j**, qui est à tolérance normale, est incluse dans l'ensemble des barres filles à découper, mais où le complément est choisi dans {1,2,...,j-1} de telle sorte que deux affranchissements soient nécessaires. En effet, le terme -a+γ est la modification du coût due à l'introduction de la barre fille de type **j** qui permet d'économiser un affranchissement mais qui rajoute un trait de scie.

Il n'y a pas d'autres cas à envisager car les barres filles sont classées par ordre de tolérance avec, en début de classement, les barres filles à tolérance réduite.

c. Formule à appliquer lorsque deux affranchissements sont requis

Quatre cas sont envisageables pour obtenir un placement local nécessitant deux affranchisssements, avec deux cas identiques quelle que soit la tolérance de la barre fille de type **j** :

$$f(j,\lambda,2)=\begin{cases}\min\begin{cases}f(j-1,\lambda,2),\\ f\left(j-1,\lambda-l_j-\gamma,2\right)+\gamma,\sigma(\lambda,j)\end{cases} & \text{si } \tau_j=1\ ;\\ f(j-1,\lambda,2) & \text{sinon.}\end{cases} \qquad \text{(IV.8)}$$

où :

- **f(j-1,λ,2)** est le coût correspondant au cas où les barres filles à découper dans la barre mère de longueur λ sont choisies dans l'ensemble {1,2,...,j-1} et où deux affranchissements sont nécessaires ;
- **f(j-1,λ-l$_j$-γ,2)+γ** correspond au cas où la barre fille de type **j** à tolérance réduite est incluse dans l'ensemble des barres filles à découper, mais où le complément est choisi dans {1,2,...,j-1} de telle sorte que deux affranchissements soient nécessaires ;
- enfin, **σ(λ,j)** est le coût correspondant au cas où la barre fille de type **j** est à tolérance réduite et qu'elle est la seule à être découpée dans la barre mère brute de longueur λ. Ce coût s'écrit :

$$\sigma(\lambda,j)=\begin{cases}+\infty, & \text{si } \lambda<l_j+2a\ ;\\ a+c\left(\lambda-l_j-a-\gamma\right) & \text{sinon ;}\end{cases} \qquad \text{(IV.9)}$$

Il n'y a pas d'autres cas à considérer étant donné que les barres filles à tolérance réduite sont examinées en premier.

Le coût minimal correspondant à un ensemble C donné de barres filles découpées dans une barre de longueur **L** est $\min_{0 \le q \le 2} f(|C|, L, q)$.

Ces différentes formules sont regroupées dans l'algorithme suivant.

3.1.3.2. Algorithme pseudo-polynomial pour le premier sous-problème

Pour une barre mère donnée, l'algorithme s'écrit :

ALGORITHME PSEUDO-POLYNOMIAL POUR LE PREMIER SOUS-PROBLÈME
1. Les données du problème sont : • *C_0, ensemble des barres filles candidates numérotées de 1 à n* • *L, longueur de la barre mère* *2. Initialiser* • *$C = C_0$, ensemble des barres filles candidates non placées* *3. Classer les barres filles à tolérance réduite en début de liste avec j comme indice des barres filles, j varie de 1 à |C|* *4. Pour tout type j de barres filles, j variant de 0 à |C|* *4.1. Pour λ variant de 0 à L (millimètre par millimètre)* *4.1.1. Pour q variant de 0 à 2* *4.1.1.1. Calculer le coût f(j, λ, q) d'après les relations (IV.5), (IV.6) et (IV.7) définies suivant q* *4.1.2. Fin de boucle* *4.2. Fin de boucle* *5. Fin de boucle* *6. Initialiser le meilleur coût f^* à $+\infty$* *7. Parcourir q de 0 à 2* *7.1. Si f(n, L, q) < f^* alors* *$f^* = f(n, L, q)$* *8. Fin de boucle.*

Afin d'obtenir la solution optimale, on calculera f(j,λ,q) pour chaque combinaison de **j**, **λ** et **q**. Comme nous l'avons montré, le calcul de f(j,λ,q) dépend du nombre final d'affranchissements à réaliser (i.e. de **q**). Sachant que **q** varie de 0 à 2, pour chaque combinaison, au plus trois comparaisons sont nécessaires pour calculer f(j,λ,q). Le plus petit coût f(j,λ,q) est calculé à partir d'au plus quatre coûts.

3.1.3.3. Conclusion concernant l'algorithme pseudo-polynomial

Le nombre de valeurs possibles de **j**, indice des barres filles (respectivement λ, longueur restante de la barre mère) est $|C|$ (respectivement **L**). Le nombre de valeurs possibles pour le nombre **q** d'affranchissements est 3, et cette valeur est indépendante de la taille du problème. Donc, le nombre de combinaisons est au plus $9|C|L$. Par conséquent, la complexité, par l'approche de la programmation dynamique est de $O(|C|L)$. Dans notre cas, cette approche est très coûteuse en temps, surtout quand elle est utilisée comme sous-programme. En effet, la longueur de la barre mère atteint les 25 m et l'unité de mesure la plus faible utilisée est le millimètre, ce qui signifie que $L = 25\,000$.

C'est pourquoi, nous préférons utiliser des heuristiques basées sur la méthode FFD (First Fit Decreasing) ou sur une méthode de l'arborescence limitée. Cependant, la méthode de programmation dynamique reste intéressante lorsque l'unité de mesure de la longueur est grande, c'est-à-dire lorsque l'unité de mesure la plus faible utilisée n'est plus le millimètre mais le centimètre par exemple. Ce n'est pas possible dans notre cas.

3.2. First Fit Decreasing (FFD)

3.2.1. Description de l'algorithme FFD

La première heuristique est inspirée du problème du "sac à dos". Cette appellation provient de la situation suivante : un randonneur doit choisir, parmi une liste d'objets à emporter dans son sac, un sous-ensemble dont le poids n'excède pas un poids donné et qui soit le meilleur sous-ensemble possible, par exemple du point de vue de l'utilité des objets. Il choisira de placer d'abord les objets les plus difficiles à placer. D'où la transcription de la méthode dans notre cas.

Elle consiste à placer les barres filles commandées par ordre décroissant des longueurs sur la barre mère considérée (cf. figure IV.1). Cette solution présente l'avantage d'être très rapide, mais ne permet pas de connaître la distance à l'optimum.

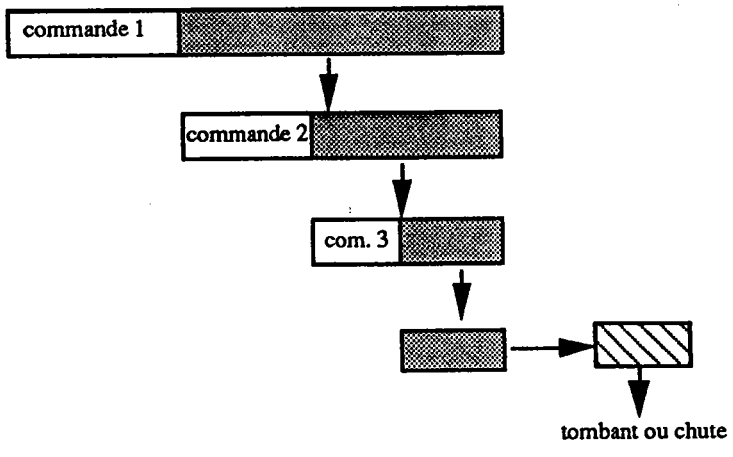

Figure IV.1 : Méthode FFD

3.2.2. Algorithme FFD

Pour une barre **L** du stock donnée, l'algorithme FFD est le suivant :

ALGORITHME FFD
1. Classer les barres filles dans une liste dans l'ordre décroissant de leur longueur l_j pour $j = 1, \ldots, n$
2. Tant que toutes les barres filles commandées ne sont pas placées et que la plus courte barre fille l_n est inférieure à la partie non utilisée de la barre mère
2.1. Placer si possible les barres filles en les prenant dans l'ordre de la liste
3. Fin tant que.

3.2.3. Amélioration du principe

Nous voyons que plusieurs barres filles commandées de même type (même longueur et même tolérance) peuvent être placées les unes après les autres sur la barre mère. Pour éviter de répéter ce placement, nous plaçons à chaque étape le maximum de barres filles du même type.

Soit **n** le nombre de types différents de barres filles. La longueur des barres filles de type **j** $(1 \leq j \leq n)$ est l_j. La tolérance des barres filles de type **j** est τ_j. Pour chaque type **j** de barres filles, m_j est le nombre de barres filles commandées.

Soit C_0 (resp. **E**), le vecteur des barres filles à réaliser (resp. placées), les éléments de ce vecteur correspondent aux nombres m_j (resp. e_j) de barres filles de chacun des types de barres filles à réaliser (resp. placées).

Les types de barres filles sont classés de telle sorte que les barres filles soient dans l'ordre décroissant de leur longueur, c'est-à-dire $l_j \geq l_{j+1}$.

Nous pouvons à présent calculer le nombre n_b de barres filles de type **j** qui peuvent être placées sur la barre mère de longueur λ. Ce nombre est majoré par le nombre de fois qu'une barre fille de type **j** peut être placée sur la barre mère et par le nombre m_j de barres filles de ce type à réaliser. Nous avons donc :

$$n_b = \min\left(\left\lfloor \frac{\lambda}{l_j} \right\rfloor, m_j\right). \qquad \text{(IV.10)}$$

Le nombre n_b de barres filles à placer est calculé sans tenir compte des pertes de matière dues aux traits de scie. Après avoir défini le nombre exact de traits de scie nécessaires à la découpe, nous vérifions si ce nombre n_b de barres filles à placer est réalisable, sinon le nombre

n_b est réduit d'une unité. La relation est la suivante :

$$n_b \Leftarrow \begin{cases} n_b & \text{si } \tau_j = 0 \text{ et } \lambda \geq n_b \times l_j + \max(n_b - n_s - 1, 0) \times \gamma ; \\ n_b & \text{si } \tau_j = 1,\ 1^T E > 0 \text{ et } \lambda \geq n_b \times (l_j + \gamma) - \gamma + a ; \\ n_b & \text{si } \tau_j = 1,\ 1^T E = 0 \text{ et } \lambda \geq n_b \times (l_j + \gamma) - \gamma + 2a ; \\ n_b - 1 & \text{sinon.} \end{cases} \qquad \text{(IV.11)}$$

où :

- $n_b - n_s - 1$ est le nombre de traits de scie à comptabiliser sachant que le dernier trait de scie n'est pas nécessaire si le reste de la barre mère est petit ;
- $n_b \times (l_j + \gamma) - \gamma$ est la longueur totale des barres filles découpées augmentée du trait de scie correspondant (le dernier trait de scie est pris en compte dans l'affranchissement) ;
- $1^T E$ est le nombre de barres filles déjà placées sur la barre mère ;
- a est l'affranchissement nécessaire si la dernière barre fille placée est à tolérance réduite.

Nous avons montré comment calculer le nombre n_b de barres filles à placer à chacune des itérations de l'algorithme FFD. Nous modifions l'algorithme FFD en conséquence.

3.2.4. Algorithme FFD Révisé

Pour réduire le nombre d'itérations, nous ne parcourons plus les barres filles les unes après les autres, mais nous procédons par types de barres filles. L'algorithme FFD révisé s'écrit :

ALGORITHME FFD RÉVISÉ

1. Données
- *L, longueur de la barre mère brute*
- *n_s, nombre de traits de scie pouvant être négligés dans le cas d'une découpe de barres filles à tolérance normale*
- *γ, largeur du trait de scie*
- *C_0, vecteur des barres filles candidates (pour n types de barres filles, les n composantes de C_0 sont les nombres de barres filles de chaque type)*

2. Initialiser
- *$\lambda = L$, longueur non utilisée de la barre mère*
- *$C = C_0$, vecteur des barres filles non placées (les composantes m_j de C sont les nombres de barres filles de type j, avec $j = 1,...,n$)*
- *$E = \varnothing$, vecteur des barres filles placées sur la barre mère (chacune des composantes e_j de E correspond au nombre de barres filles de type j placées)*

3. Classer les types de barres filles dans l'ordre décroissant des longueurs l_j des barres filles pour $j = 1, ..., n$. Les composantes des vecteurs C et E sont réarrangées dans cet ordre

4. Tant que toutes les barres filles commandées ne sont pas placées et que la plus courte barre fille l_n est de longueur inférieure ou égale à la partie de la barre mère non affectée, faire varier j de 1 à n

4.1. Calculer le nombre n_b de barres filles de type j à placer (voir relations (IV.10) et (IV.11))

4.2. Si $n_b > 0$ alors

4.2.1. Retenir le placement dans E

$$e_j \Leftarrow e_j + n_b \qquad (IV.12)$$

4.2.2. Mettre à jour les variables :

- *longueur non utilisée de la barre mère*

(nombre de traits de scie comptabilisés)

$$n_s' \Leftarrow \begin{cases} \max(n_s - n_b, 0) & \text{si } \tau_j = 0 \\ 0 & \text{sinon} \end{cases} \qquad (IV.13)$$

(dernier trait de scie négligé si reste de la barre mère < γ)

$$\lambda \Leftarrow Max(\lambda - n_b \times l_j - (n_b - n_s')\gamma, 0) \qquad (IV.14)$$

- *reste du nombre de traits de scie que l'on peut négliger*

$$n_s \Leftarrow n_s - n_s' \qquad (IV.15)$$

- *nombre de barres filles de type j restant à réaliser*

$$m_j \Leftarrow m_j - n_b \qquad (IV.16)$$

4.2. Fin si

5. Fin tant que.

3.2.5. Exemple de placement par l'algorithme FFD

Soit une barre mère de longueur L = 14 m dans laquelle il faut découper au mieux :

- 2 barres filles de longueur 6 m à tolérance normale ;
- 1 barre fille de longueur 4 m à tolérance normale ;
- 1 barre fille de longueur 2 m à tolérance normale.

La largeur γ du trait de scie est 0,01 m.

Le nombre n_s de traits de scie pouvant être négligés dans le cas d'une découpe de barres filles à tolérance normale est 2.

1. Les données de départ sont :
   $L = 14$; $n_s = 2$; $\gamma = 0{,}01$; $C_0 = (2,1,1)$.
2. Les variables sont initialisées comme suit :
   $\lambda = 14$; $C = (2,1,1)$; $E = (0,0,0)$.

3. Classons d'abord les types de barres filles suivant la longueur des barres filles correspondantes (avec n = 3) :
   - $l_1 = 6$ m ; $\tau_1 = 0$; $m_1 = 2$;
   - $l_2 = 4$ m ; $\tau_2 = 0$; $m_2 = 1$;
   - $l_3 = 2$ m ; $\tau_3 = 0$; $m_3 = 1$;

   $C = (2,1,1)$.
4. L'algorithme parcourt tous les types **j** de barres filles en commençant par le type j = 1. L'algorithme place le maximum possible de barres filles de type 1 sur λ.
   4.1. D'après la relation (IV.10), le nombre n_b de barres filles du premier type que l'algorithme peut placer est :
   $$n_b = \min\left(\left\lfloor \frac{\lambda}{l_j} \right\rfloor, m_j\right) = 2.$$
   4.2. La condition (IV.11) confirme que $n_b = 2$.
      4.2.1. L'algorithme retient le placement dans E. D'après la relation (IV.12) :
      $e_j = 0 + 2 = 2$; $E = (2,0,0)$.
      4.2.2. Les variables sont mises à jour :
      - d'après la relation (IV.13), le nombre n_s' de traits de scie à comptabiliser est : $n_s' = 2$.

        d'après la relation (IV.14), la longueur λ non utilisée de la barre mère après avoir découpé les deux barres filles de type 1 est :
        $\lambda = 14 - 2\times6 - (2-2)\times0{,}01 = 2$;
      - d'après la relation (IV.15), le reste n_s du nombre de traits de scie que l'on peut négliger est alors : $n_s = 0$;
      - d'après la relation (IV.16), le nombre de barres filles de type **j** restant à réaliser devient : $m_1 = 2 - 2 = 0$; $C = (0,1,1)$.

      La fin de la boucle 3 est atteinte, l'algorithme remonte au début de la boucle.

4.(1) Il reste encore des barres filles à placer et la plus courte barre fille l_3 est égale à la longueur λ non utilisée de la barre mère. L'algorithme peut continuer.
   4.1. L'algorithme teste s'il peut découper la barre fille de type 2 dans le reste de la barre mère appelée à présent λ, ce qui n'est pas possible car d'après la relation (IV.10) le nombre de barres filles de type 2 à réaliser est $n_b = 0$.

   L'algorithme continue pour le type de barres filles suivant : j = 3. Il calcule le nombre de barres filles de type 3 qu'il peut placer sur λ, d'après la relation (IV.10) et (IV.11) : $n_b = 1$.
   4.2. Le placement de $n_b = 1$ barres filles de type 3 est réalisable. L'algorithme peut continuer.
      4.2.1. L'algorithme retient le placement dans E. D'après la relation (IV.12) :
      $e_3 = 0 + 1 = 1$; $E = (2,0,1)$.

4.2.2. Les variables sont mises à jour :

- d'après les relations (IV.13) et (IV.14), la longueur λ non utilisée de la barre mère après avoir découpé les deux barres filles de type 1 est : $\lambda = \text{Max}(2 - 1\times2 - (1-0)\times0,01,0) = 0$;
- n_S reste nul ;
- d'après la relation (IV.16), le nombre de barres filles de type **j** restant à réaliser devient : $m_3 = 1 - 1 = 0$; $C = (0,1,0)$.

La longueur non utilisée de la barre mère est alors $\lambda = 0$. Le placement par l'algorithme FFD est terminé avec la découpe de 2 barres filles de longueur 6 m et d'une autre barre fille de 2 m dans une barre mère de 14 m. La chute (ou le coût de la chute) est nulle pour ce placement. Le placement final est donné dans la figure IV.2.

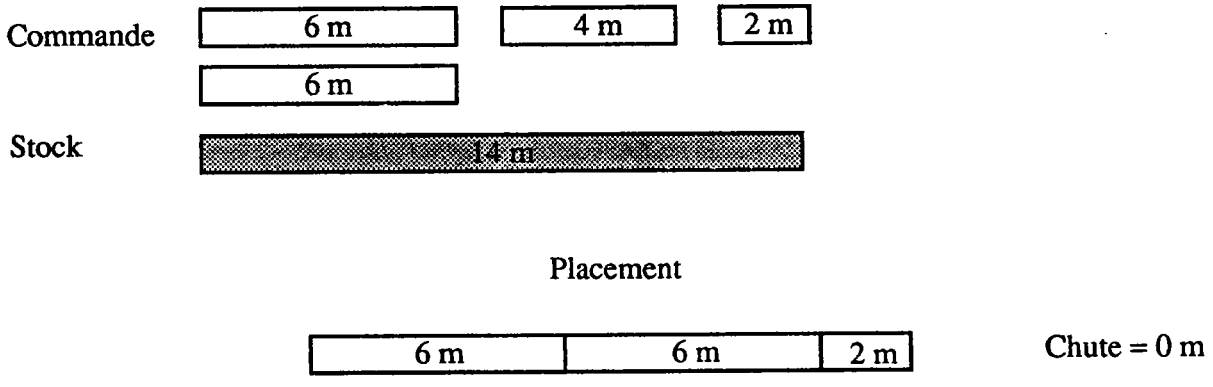

Figure IV.2 : Résulat du placement par l'algorithme FFD

3.2.6. Conclusion concernant l'algorithme FFD

La qualité du placement par cet algorithme est très variable mais son temps de traitement est très court (moins d'une seconde[1] pour un nombre important de types de barres filles (>10)). Cette observation est primordiale pour déterminer l'aptitude de l'algorithme à être utilisé par d'autres algorithmes de placement agissant non plus localement mais globalement. La rapidité de cet algorithme autorise un nombre important d'itérations dans les algorithmes généraux.

3.3. Méthode de l'arborescence limitée

3.3.1. Description de la méthode de l'arborescence limitée

La méthode utilise le principe de la Procédure par Séparation et Évaluation en profondeur d'abord décrite à la section 2.1 du chapitre II. Pour réduire le temps de traitement, nous avons introduit un paramètre qui permet d'arrêter la recherche dès que le pourcentage du bout restant de la barre mère découpée par rapport à sa longueur initiale est inférieur à un pourcentage choisi, noté δ, de la longueur totale de la barre. Dans la pratique, nous choisissons $\delta = 2$ %.

Afin de conserver des données similaires entre algorithmes, nous reprenons les notations de l'algorithme FFD :

- **n** est le nombre de types différents de barres filles. La longueur des barres filles de type **j** $(1 \leq j \leq n)$ est l_j. La tolérance pour les barres filles de type **j** est τ_j. Pour chaque type **j** de barres filles, m_j est le nombre de barres filles commandées.
- C_0 (resp. E) est le vecteur des barres filles à réaliser (resp. placées), les éléments de ce vecteur correspondent aux nombres m_j (resp. e_j) de barres filles de chacun des types de barres filles à réaliser (resp. placées).

Un schéma du parcours par arborescence est donné à la figure IV.3.

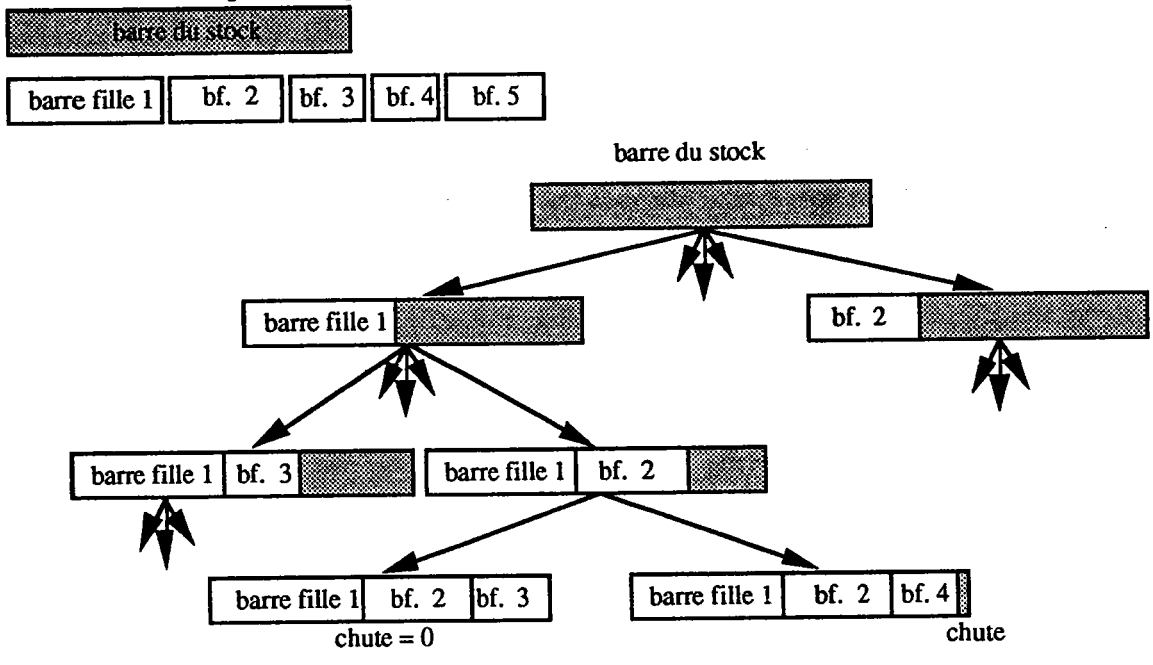

Figure IV.3 : Méthode de l'arborescence limitée sur une barre mère

3.3.2. Algorithme de l'arborescence limitée

Pour une longueur de barre mère donnée, la procédure récurrente s'écrit de la manière suivante :

ALGORITHME D'ARBORESCENCE LIMITÉE

1. Données

- *L, longueur de la barre mère brute*
- *n_s, nombre de traits de scie pouvant être négligés dans le cas d'une découpe de barres filles à tolérance normale*
- *γ, largeur du trait de scie*
- *a, largeur d'un affranchissement*
- *C_0, vecteur des barres filles candidates (les composantes de C_0 sont les nombres de barres filles de chaque type. Au type j de barre fille correspond une longueur l_j et une tolérance τ_j, avec $j = 1,...,n$)*

2. Initialiser

- $c_b^* = +\infty$ *(tout au long de l'algorithme,* c_b^* *contiendra le coût minimal)*
- $\lambda = L$, *longueur non utilisée de la barre mère*
- $C = C_0$, *vecteur des barres filles non placées (chacune des composantes* m_j *de C est le nombre de barres filles de type j à placer)*
- $E^* = \varnothing$, *vecteur du placement retenu (les composantes sont les nombres de barres filles de chaque type placées)*
- $E = \varnothing$, *vecteur des barres filles placées sur la barre mère (les composantes* e_j *de E sont les nombres de barres filles de chaque type j placées)*
- $n_s(E) = n_s$, *nombre de traits de scie pouvant encore être négligés dans le cas d'une découpe de barres filles à tolérance normale en fonction des barres filles déjà placées*
- *Fin = Faux*

3. Procédure ARBO(λ, *E,* $n_s(E)$, *C, Fin)*

3.1. Si ($\lambda \times 100/L < \delta$*) (c'est-à-dire que le pourcentage de la longueur restante par rapport à la longueur initiale de la barre mère (*$\lambda \times 100/L$*) est inférieur à un seuil* δ*) ou que la plus courte des barres filles restant à réaliser est de longueur supérieure à la longueur non utilisée de la barre mère*

3.1.1. Calculer le coût $c_b = \rho(L,E)$ *du placement*

3.1.2. Si c_b *est inférieur à* c_b^*, *alors*

$c_b^* = c_b$ *et* $E^* = E$

3.1.3. Fin = Vraie

3.2. Sinon

3.2.1. Si Fin = Faux

3.2.1.1. Garder en mémoire le meilleur placement E^* *donnant le meilleur coût* c_b^*, *i.e. si* $c_b < c_b^*$ *alors* $c_b^* = c_b$ *et* $E^* = E$

3.2.1.2. Pour tout ***type j de barres filles****, j variant de 1 à n*

3.2.1.2.1. Si le placement de la barre fille de type j est possible

(si $\tau_j = 0$ *et* $l_j \leq \lambda$*) ou (si* $\tau_j = 1$ *et* $l_j + a \leq \lambda$*)*

3.2.1.2.1.1. Retenir le placement et les variables correspondantes

$E' = E$ *; (vecteur des barres filles placées dont les composantes* e'_j *sont les nombres de barres filles de type j placées)*

$e'_j = e_j + 1$;

$C' = C$ *; (vecteur des barres filles non placées dont les composantes* m_j' *sont les quantités de barres filles de chaque type j')*

$m_j' = m_j - 1$

Si $\tau_j = 0$ *et* $n_s(E) > 0$, *alors*

$\lambda' = \lambda - l_j$ *et* $n_s(E') = n_s(E) - 1$

Sinon

$\lambda' = \lambda - l_j + \gamma$

$j \Leftarrow j-1$ *(pour revenir à ce type de barre fille à la boucle suivante)*

3.2.1.2.1.2. Appel procédure ARBO(λ' , E', $n_s(E')$, C', Fin)

3.2.1.2.2. Fin si

3.2.1.3. Fin pour

3.2.2. Fin si

3.3. Fin si

4. Fin procédure ARBO.

3.3.3. Exemple d'application de l'algorithme de l'arborescence limitée

Le problème est de découper de façon optimale dans une barre mère de longueur L = 14 m :

- 3 barres filles de longueur 6 m à tolérance normale ;
- 3 barres filles de longueur 4 m à tolérance normale.

La largeur du trait de scie γ est 0,01 m.

Fixons le pourcentage δ de chute pour l'arrêt des recherches à 2 %.

1. Les données de départ sont :
   L = 14 ; n_s = 2 ; γ = 0,01 ;
   C_0 = (3,3), avec :
   - n = 2 ;
   - l_1 = 6 ; τ_1 = 0 ; m_1 = 3 ;
   - l_2 = 4 ; τ_2 = 0 ; m_2 = 3.
2. Les variables sont initialisées comme suit :
   c_b^* = +∞ ; λ = 14 ; C = C_0 = (3,3) ; E^* = E = (0,0) ; Fin = Faux ; $n_s(E)$ = 2.
3. L'algorithme entre dans la procédure ARBO.
   3.1. Comme ($\lambda \times 100/L < \delta$), nous passons à la deuxième partie de la procédure récurrente : étape 3.2.
   3.2.1. La condition est vraie, l'algorithme entre dans les niveaux inférieurs de cette étape.
   3.2.1.1. Comme il n'y a pas encore eu de placement, le meilleur placement E^* reste vide et le coût infini.
   3.2.1.2. L'algorithme commence par le type de barres filles j = 1.
   3.2.1.2.1. L'algorithme teste s'il est possible de poser une barre fille de type 1 à tolérance normale (τ_1 = 0). La relation est vraie, il peut continuer à ce niveau.
   3.2.1.2.1.1. Le placement et les variables correspondantes sont retenus. L'algorithme ajoute donc une barre fille de type 1 au placement précédent :
   E' = E + (1,0) = (1,0).

Les nouvelles variables sont :

$C' = C$; $m_j' = m_j - 1 = 2 \Rightarrow C' = (2,3)$;

$\lambda' = \lambda - l_j = 8$;

$n_s(E') = n_s(E) - 1 = 1$.

3.2.1.2.1.2. Nous allons maintenant au niveau 2 de la récurrence en renommant les variables :

$E = E'$; $\lambda = \lambda' = 8$; $C = C'$; $n_s(E) = n_s(E')$.

3.(1) Au niveau 2 de la procédure ARBO, l'algorithme teste la condition d'arrêt ($\lambda \times 100/L < \delta$). Il en déduit qu'elle n'est pas vérifiée. En plus $l_1 = 6 < \lambda = 8$. L'algorithme passe à la deuxième partie de la procédure récurrente.

3.2.1.1. L'algorithme teste tout d'abord si le placement correspondant au vecteur E est de meilleure qualité que celui du vecteur E^*, i.e. $\rho(L,E) < c_b^*$. La relation est vraie, aussi il retient ce placement et le coût correspondant :

$E^* = E = (1,0)$ et $c_b^* = 8$.

3.2.1.2. L'algorithme parcourt ensuite la liste des types de barres filles à réaliser en commençant par $j = 1$.

3.2.1.2.1. L'algorithme teste s'il est possible de poser une barre fille de type 1. La relation est vraie. Il ajoute une barre fille de type 1 au placement précédent. Les nouvelles valeurs des variables sont :

$E' = (2,0)$; $\lambda' = 2$; $n_s(E') = 0$; $m_1' = 1$; $C' = (1,3)$.

Nous allons maintenant au niveau 3 de la récurrence en renommant les variables comme précédemment.

3.(2) Au niveau 3 de la procédure ARBO, la condition d'arrêt ($\lambda \times 100/L < \delta$) n'est toujours pas vérifiée. Dans la deuxième partie de la procédure récurrente, ce placement est retenu comme étant le meilleur jusqu'à présent : $E^* = (2,0)$ et $c_b^* = 2$.

3.2.1.2. En parcourant toutes les barres filles restantes, l'algorithme constate que plus aucune d'entre-elles ne peut être placée sur ce qui reste de la barre mère. Par conséquent, l'algorithme remonte au niveau précédent, c'est-à-dire au niveau 2.

3.2.1.2. Au niveau 2, comme l'indice $j = 1$ des types de barres filles vient d'être exploré, nous passons à l'indice $j = 2$. Une barre fille de type 2 est alors placée. Les valeurs des variables sont :

$E' = (1,1)$; $\lambda' = 4$; $n_s'(E') = 0$; $m_2' = 2$;

3.(3) Au niveau 3 de la procédure ARBO, la condition d'arrêt ($\lambda \times 100/L < \delta$) n'étant toujours pas vérifiée, l'algorithme passe à la seconde partie de la procédure récurrente. Ce nouveau placement, ayant un coût c_b de 4, n'améliore pas la solution déjà retenue. L'algorithme passe à l'étape 3.2.1.2.

3.2.1.2.1. La condition n'est pas satisfaite, le placement de la barre fille de type 1 n'est pas possible.

L'algorithme passe au second type. Le placement est possible. Les valeurs des variables deviennent :

$E' = (1,2)$; $\lambda' = 0$; $n_s^{\cdot}(E') = 0$; $m_2^{\cdot} = 1$; $C' = (2,1)$.

3.(4) Au niveau 4 de la procédure ARBO, la condition d'arrêt est vérifiée. Le meilleur placement retenu est alors :

$E^* = E = (1,2)$ et $c_b^* = \rho(L,E) = 0$.

L'algorithme remonte tous les niveaux en mettant la variable "Fin" à Vraie et la procédure se termine. Le placement final est donné dans la figure IV.4.

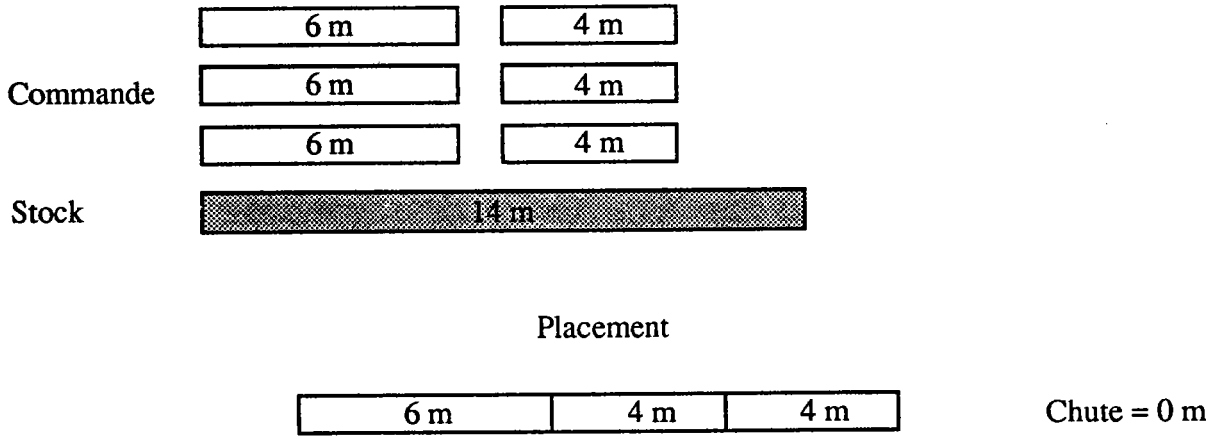

Figure IV.4 : Résulat du placement par l'algorithme de l'arborescence limitée

3.3.4. Conclusion relative à l'algorithme de l'arborescence limitée

Le temps de traitement de cet algorithme est fonction de l'ensemble des barres filles candidates et du nombre d'éléments de cet ensemble pouvant être découpés dans la barre mère. Ce temps est d'environ une seconde pour un nombre moyen (5) de barres filles placées sur une barre mère et un nombre (7) de types de barres filles.

Cette remarque fait que son utilisation future par d'autres algorithmes traitant toutes les barres disponibles en stock est plus limitée que la méthode FFD, et ceci malgré la garantie d'être proche du placement optimal local. Son utilisation par les algorithmes itératifs doit donc être évitée.

4. SECOND SOUS-PROBLÈME (placement de b barres filles sur une barre mère)

Ce deuxième sous-problème est différent du précédent en ce sens que nous imposons le nombre de barres filles à découper dans la barre mère. La résolution de ce sous-problème sera

utilisée lors de la description des algorithmes inspirés de la programmation dynamique (cf. chapitre II, section 6).

4.1. Un algorithme pseudo-polynomial

Ce sous-problème consiste à affecter **b** barres filles d'un ensemble **C** donné (évidemment, nous devons avoir $b \leq |C|$) à une barre mère de longueur **L** de sorte que le coût total induit par les chutes et le tombant soit minimal. En utilisant un raisonnement similaire à celui utilisé pour le premier sous-problème, nous pouvons montrer que ce second sous-problème peut être résolu par une méthode pseudo-polynomiale utilisant l'approche de la programmation dynamique.

4.1.1. Formulation de l'algorithme pseudo-polynomial

Soit $g(b',j,\lambda,q)$ le coût de placement le plus faible sur une barre mère donnée de longueur λ d'un sous-ensemble non vide appartenant à $S_{j,q}$, tel que le nombre total de barres filles dans ce sous-ensemble soit **b'** et que le nombre d'affranchissements requis soit **q**. Cela revient à écrire :

$$g(b',j,\lambda,q) = \min_{\substack{E \in S_{j,q} \\ |E| = b'}} \rho(\lambda,E) \; ; \qquad \text{(IV.17)}$$

avec $\forall b',\ 0 < b' \leq b$.

Les conditions initiales sont :

$$g(b',0,\lambda,q) = +\infty \;,\ \forall b' > 0 \qquad \text{(IV.18)}$$

La démarche est légèrement différente de celle de l'algorithme pseudo-polynomial du premier sous-problème : cette fois-ci, nous cherchons à chaque itération le placement d'un nombre précis de barres filles. Pour obtenir les formules de récurrence, considérons tout d'abord le placement d'une seule barre fille (b' = 1) sur la barre mère de longueur λ.

4.1.1.1. Placement d'une seule barre fille sur la barre mère

Nous retrouvons ici les trois cas déjà rencontrés dans le premier sous-problème, un pour chaque nombre **q** possible d'affranchissements à réaliser.

a. Formule à appliquer lorsqu'aucun affranchissement n'est requis

Pour aboutir au meilleur placement local ayant q = 0 affranchissement, il faut choisir le coût minimal entre l'option de ne pas placer la barre fille **j** (ce qui conduit à un coût $g(1,j-1,\lambda,0)$) et l'option de la placer, dont le coût est $v(\lambda, j)$ (cf. formule (IV.6)).

$$g(1,j,\lambda,0) = \begin{cases} +\infty & \text{si } \tau_j = 1 \; ; \\ \min\{g(1,j-1,\lambda,0), v(\lambda,j)\} & \text{sinon.} \end{cases} \qquad \text{(IV.19)}$$

où :

- $+\infty$ est le coût correspondant au cas où la barre fille de type **j** est à tolérance réduite. L'ensemble {1,2,...,j} ne contient donc que des barres filles à tolérance réduite. Par conséquent, leur découpe dans la barre mère brute de longueur λ nécessite deux affranchissements, ce qui est incompatible avec l'hypothèse q = 0 ;
- **g(1,j-1,λ,0)** est le coût correspondant au cas où les barres filles à découper dans la barre mère de longueur λ sont choisies dans l'ensemble {1,2,...,j-1} et où aucun affranchissement n'est nécessaire ;
- enfin, **v(λ,j)** est le coût (IV.6) correspondant au cas où la barre fille de type **j** est à tolérance normale et qu'elle est découpée dans la barre mère brute de longueur λ.

Il n'y a pas d'autres possibilités car on ne peut placer une barre fille sur la barre mère de telle sorte qu'aucun affranchissement ne soit nécessaire (q = 0).

b. Formule à appliquer lorsqu'un affranchissement est requis

Considérons à présent le coût induit par le placement d'une seule barre fille :

$$g(1,j,\lambda,1) = +\infty. \qquad \text{(IV.20)}$$

Dans le cas où une seule barre fille (b' = 1) doit être placée, le coût g(1,j,λ,1) est infini. Ceci s'explique par le fait que le placement d'une unique barre fille sur une barre mère brute entraîne soit zéro affranchissement dans le cas d'une tolérance normale, soit deux dans le cas d'une tolérance réduite.

c. Formule à appliquer lorsque deux affranchissements sont requis

Le coût du placement d'une seule barre fille sur la barre mère doit prendre en compte trois cas, parmi lesquels deux sont identiques quelle que soit la tolérance de la barre fille de type **j** :

$$g(1,j,\lambda,2) = \begin{cases} \min\{g(1,j-1,\lambda,2), \sigma(\lambda,j)\} & \text{si } \tau_j = 1 ; \\ g(1,j-1,\lambda,2) & \text{sinon.} \end{cases} \qquad \text{(IV.21)}$$

où :

- **g(1,j-1,λ,2)** est le coût correspondant au cas où la barre fille à découper dans la barre mère de longueur λ est choisie dans l'ensemble {1,2,...,j-1} et où deux affranchissements sont nécessaires ;
- **σ(λ,j)** est le coût (IV.9) correspondant au cas où la barre fille de type **j** est à tolérance réduite et qu'elle est découpée dans la barre mère brute de longueur λ.

Il n'y a pas d'autres cas à considérer étant donné que les barres filles à tolérance réduite sont examinées en priorité.

4.1.1.2. Placement de b' barres filles sur la barre mère

De la même manière, les formules de récurrence sont régies par le nombre **q** d'affranchissements à pratiquer dans la découpe d'une barre mère, avec $1<b'\leq b$.

a. Formule à appliquer lorsqu'aucun affranchissement n'est requis

Trois cas sont envisageables pour aboutir à un placement ne nécessitant pas d'affranchissement :

$$g(b',j,\lambda,0)=\begin{cases}+\infty & \text{si } \tau_j=1\ ;\\ \min\begin{cases}g(b',j-1,\lambda,0),\\ g(b'-1,j-1,\lambda-l_j-\gamma,0)+\gamma,\\ g(b'-1,j-1,\lambda-l_j-\gamma+a,1)-a+\gamma\end{cases} & \text{sinon.}\end{cases} \qquad \text{(IV.22)}$$

où :

- $+\infty$ est le coût correspondant au cas où la barre fille de type **j** est à tolérance réduite. D'après l'ordre des barres filles, l'ensemble {1,2,...,j} ne contient donc que des barres filles à tolérance réduite. Leur découpe dans la barre mère brute de longueur λ nécessite deux affranchissements, ce qui est incompatible avec l'hypothèse q = 0 ;
- **$g(b',j-1,\lambda,0)$** est le coût correspondant au cas où les barres filles à découper dans la barre mère de longueur λ sont choisies dans l'ensemble {1,2,...,j-1} et où aucun affranchissement n'est nécessaire ;
- **$g(b'-1,j-1,\lambda-l_j-\gamma,0)+\gamma$** est le coût correspondant au cas où la barre fille de type **j**, qui est à tolérance normale, est incluse dans l'ensemble des barres filles à découper dans la barre mère de longueur λ, le reste étant choisi dans l'ensemble {1,2,...,j-1} de telle sorte qu'il ne nécessite aucun affranchissement ;
- enfin, **$g(b'-1,j-1,\lambda-l_j-\gamma+a,1)-a+\gamma$** correspond au cas où la barre fille de type **j** est incluse dans l'ensemble des barres filles à découper, mais où le complément est choisi dans {1,2,...,j-1} de telle sorte qu'un affranchissement soit nécessaire. En effet, le terme $-a+\gamma$ est la modification du coût due à l'introduction de la barre fille de type **j** qui permet d'économiser un affranchissement mais qui rajoute un trait de scie.

Il est à noter qu'il est impossible de placer la barre fille de type **j** et de choisir le reste parmi {1,2,...,j-1} de telle sorte que deux affranchissements soient nécessaires. En effet, ceci conduirait à un placement qui nécessite au moins un affranchissement, ce qui est en contradiction avec l'hypothèse q = 0.

Le coût le plus faible est retenu pour le quadruplet (b', j, λ, q).

b. Formule à appliquer lorsqu'un affranchissement est requis

Pour q = 1 affranchissement à réaliser dans le nouveau placement, trois cas sont envisageables :

$$g(b',j,\lambda,1)=\begin{cases}+\infty & \text{si } \tau_j = 1 ; \\ \min\begin{cases} g(b',j-1,\lambda,1), \\ g\left(b'-1,j-1,\lambda-l_j-\gamma+a,2\right)-a+\gamma \end{cases} & \text{sinon ;}\end{cases} \qquad \text{(IV.23)}$$

où :

- $+\infty$ est le coût correspondant au cas où la barre fille de type **j** est à tolérance réduite. D'après l'ordre de la liste des barres filles, l'ensemble {1,2,...,j} est uniquement constitué de barres filles à tolérance réduite. Le placement d'un sous-ensemble de {1,2,...,j} entraîne le placement de deux affranchissements, ce qui est incompatible avec l'hypothèse q = 1 ;
- **$g(b',j-1,\lambda,1)$** est le coût correspondant au cas où les barres filles à découper dans la barre mère de longueur λ sont choisies dans l'ensemble {1,2,...,j-1} et où un affranchissement est nécessaire ;
- **$g(b',j-1,\lambda-l_j-\gamma+a,2)-a+\gamma$** correspond au cas où la barre fille de type **j**, qui est à tolérance normale, est incluse dans l'ensemble des barres filles à découper, mais où le complément est choisi dans {1,2,...,j-1} de telle sorte que deux affranchissements soient nécessaires. En effet, le terme $-a+\gamma$ est la modification du coût due à l'introduction de la barre fille de type **j** qui permet d'économiser un affranchissement mais qui rajoute un trait de scie.

Il n'y a pas d'autres cas à envisager car les barres filles sont classées en mettant en tête les barres filles à tolérance réduite.

c. Formule à appliquer lorsque deux affranchissements sont requis

Le nouveau placement, imposant la réalisation de q = 2 affranchissements, considère trois cas dont deux sont indentiques quelle que soit la tolérance de la barre fille de type **j** :

$$g(b',j,\lambda,2)=\begin{cases}\min\begin{cases} g(b',j-1,\lambda,2), \\ g\left(b'-1,j-1,\lambda-l_j-\gamma,2\right)+\gamma \end{cases} & \text{si } \tau_j = 1 ; \\ g(b',j-1,\lambda,2) & \text{sinon ;}\end{cases} \qquad \text{(IV.24)}$$

où :

- **$g(b',j-1,\lambda,2)$** est le coût correspondant au cas où les barres filles à découper dans la barre mère de longueur λ sont choisies dans l'ensemble {1,2,...,j-1} et où deux affranchissements sont nécessaires ;

- $g(b'-1,j-1,\lambda-l_j-\gamma,2)+\gamma$ correspond au cas où la barre fille de type **j** à tolérance réduite est incluse dans l'ensemble des barres filles à découper, mais où le complément est choisi dans {1,2,...,j-1} de telle sorte que deux affranchissements soient nécessaires.

Il n'y a pas d'autres cas à considérer étant donné que les barres filles à tolérance réduite sont considérées en premier.

Le coût minimal pour ce problème est $\min_{0 \le q \le 2} g(b,|C|,L,q)$. Ces formules sont utilisées dans l'algorithme pseudo-polynomial que nous présentons maintenant.

4.1.2. Algorithme pseudo-polynomial pour le second sous-problème

Les barres filles sont considérées individuellement et sont supposées différentes les unes des autres. Pour une barre mère du stock donnée de longueur **L** et pour un nombre **b** de barres filles commandées, l'algorithme s'écrit :

ALGORITHME PSEUDO-POLYNOMIAL POUR LE SECOND SOUS-PROBLÈME

1. Les données du problème sont :
- *C_0, ensemble des barres filles candidates numérotées de 1 à n*
- *L, longueur de la barre mère*

2. Initialisation
- *$C = C_0$, ensemble des barres filles candidates non placées*

3. Classer les barres filles à tolérance réduite en début de liste avec j = 1,..., |C| comme indice des barres filles

*4. Pour tout **nombre b' de barres filles** placées, b' variant de 1 à b*

*4.1. Pour tout λ variant de 0 à L et pour tout **q** variant de 0 à 2*

4.1.1. $g(b', 0, \lambda, q) = +\infty$

*4.2.1. Pour tout **type j de barres filles**, j variant de 1 à |C|*

4.2.1.1.1. Pour λ variant de 0 à L (millimètre par millimètre)

*4.2.1.1.1.1. Pour **q** variant de 0 à 2*

4.2.1.1.1.1.1. Calculer le coût $g(b', j, \lambda, q)$ d'après les relations (IV.17) à (IV.24) définies en fonction de b' et q

4.2.1.1.1.2. Fin de pour

4.2.1.1.2. Fin de pour

4.2.2. Fin de pour

4.2. Fin de pour

5. Fin de pour

6. Initialiser g^* à $+\infty$

7. Pour q variant de 0 à 2
7.1. Si $g(b, n, L, q) < g^*$ alors
$g^* = g(b, n, L, q)$
8. Fin de pour.

4.1.3. Conclusion relative à l'algorithme pseudo-polynomial

En utilisant les mêmes arguments que pour le premier sous-problème, nous pouvons déduire que la complexité de cette approche est $O(b|C|L)$. Comme dans le cas du premier sous-problème, cet algorithme nécessite donc un temps de calcul important. Par conséquent, nous utilisons une méthode approchée au lieu de la méthode exacte.

4.2. Méthode par permutations

4.2.1. Description de la méthode par permutations

Cet algorithme procède par permutations de barres filles placées et non placées. A chaque itération, un ensemble **E** de barres filles tel que $|E| = b$, a été affecté à la barre mère donnée. Soit **C'** = C - E l'ensemble des barres filles de C non placées à l'instant considéré. Au début de l'algorithme, **E** est composé des **b** barres filles les plus courtes. A chaque itération, nous essayons d'enlever une barre fille courte de **E** et nous la remplaçons par une barre fille plus longue choisie dans **C'**. Nous examinons les éléments de **C'** dans l'ordre décroissant de leur longueur, et pour chaque élément de **C'**, nous examinons les éléments de **E** dans l'ordre croissant de leur longueur. La permutation est finie lorsque la première permutation réalisable est trouvée. Nous obtenons alors un nouvel ensemble **E** et un nouvel ensemble **C'**. Le même processus est recommencé jusqu'à ce qu'il n'y ait plus de permutation réduisant la longueur de la partie restante. Le coût correspondant aux chutes et au tombant est calculé comme dans la section 2 .

Afin de conserver des données similaires entre algorithmes et de réduire le nombre d'itérations, nous reprenons les notations de l'algorithme FFD :

- **n** est le nombre de types différents de barres filles. La longueur des barres filles de type **j** $(1 \leq j \leq n)$ est l_j. La tolérance pour les barres filles de type **j** est τ_j. Pour chaque type **j** de barres filles, m_j est le nombre de barres filles commandées.
- **C** (resp. **E**) est le vecteur des barres filles à réaliser (resp. placées), les éléments de ce vecteur correspondent aux nombres m_j (resp. e_j) de barres filles de chacun des types de barres filles à réaliser (resp. placées). C_0 est le vecteur des barres filles à réaliser au départ.

4.2.2. Algorithme par permutations

Pour une barre mère donnée du stock et un nombre **b** de barres filles à placer, l'algorithme est le suivant :

ALGORITHME PAR PERMUTATIONS

1. Données

- *L, longueur de la barre mère brute*
- *n_s, nombre de traits de scie pouvant être négligés dans le cas d'une découpe de barres filles à tolérance normale*
- *γ, largeur du trait de scie*
- *C_0, vecteur des barres filles candidates (les composantes de C_0 sont les nombres de barres filles de chaque type, avec n le nombre de types différents)*

2. Initialiser

- *$c_b^* = +\infty$ (tout au long de l'algorithme, c_b^* contiendra le coût minimal)*
- *$E = \varnothing$, vecteur des barres filles placées sur la barre mère (chacune des composantes e_j correspond au nombre de barres filles de type j placées, avec $j = 1,...,n$)*
- *$b' = b$, nombre de barres filles restant à placer*

3. Classer les types de barres filles dans l'ordre décroissant de leur longueur l_j avec $j = 1,...,n$. Soit C le vecteur des barres filles correspondant à cet ordre et dont les composantes sont les nombres de barres filles de chaque type à réaliser

*4. Pour tout **type j de barres filles** de C, j variant de n à 1 tant qu'il reste des barres filles à placer ($b' > 0$)*

4.1. Calculer le nombre de barres filles de type j pouvant être placées

$$n_b = min(m_j, b') \qquad (IV.25)$$

4.2. Mettre à jour les variables

$$e_j = n_b \; ; \; m_j \Leftarrow m_j - n_b \; ; \; b' \Leftarrow b' - n_b \qquad (IV.26)$$

5. Fin pour

6. Si le placement est réalisable (le coût $c_b^* = \rho(L,E) << +\infty$)

*6.1. Parcourir tous les **types j de barres filles** de C' (vecteur des barres filles non placées) dans l'ordre décroissant des longueurs l_j des barres filles, j variant de 1 à n*

6.1.1. "Permutation" = Faux

*6.1.2. Parcourir tous les **types j' de barres filles placées**, i.e. les composantes non nulles du vecteur E, dans l'ordre croissant de leur longueur, j' variant de n à 1, tant que "Permutation" = Faux*

6.1.2.1. Retenir le nouveau placement E' (les composantes e'_j de ce vecteur correspondent aux nombres de barres filles placées de chacun des types j)

$$E' = E \; ; \; e'_j = e_j - 1 \; ; \; e'_{j'} = e_{j'} + 1$$

6.1.2.2. Calculer le coût $c_b = \rho(L,E)$ de ce nouveau placement (∞ si permutation impossible)

6.1.2.3. Si le coût c_b est meilleur que c_b^* (la différence peut être due aux affranchissements ou à la génération d'un tombant)

6.1.2.3.1. Permuter les deux barres filles dans les deux vecteurs E et C' : E = E' et $m'_j \Leftarrow m'_j + 1$; $m'_{j'} \Leftarrow m'_{j'} - 1$

6.1.2.3.2. Mettre à jour les variables

- $c_b^* = c_b$
- *"Permutation" = Vrai*
- *j ⇐ j - 1 (d'autres permutations peuvent être possibles pour ce même type de barres filles)*

6.1.2.4. Fin si

6.1.3. Fin parcourir

6.2. Fin parcourir

7. Sinon, il n'y a pas de solution.

4.2.3. Exemple d'application de l'algorithme par permutations

Le problème consiste à découper 3 barres filles (b = 3) dans une barre mère de longueur L = 14 m. Les barres commandées sont :

- 3 barres filles de longueur 6 m à tolérance normale ;
- 3 barres filles de longueur 4 m à tolérance normale ;
- 2 barres filles de longueur 2 m à tolérance normale.

La largeur du trait de scie est 0,01 m.

Le nombre de traits de scie pouvant être négligés dans le cas d'une découpe de barres filles à tolérance normale est 2.

1. Les données de départ sont :
   $L = 14$; $n_s = 2$; $\gamma = 0{,}01$;
   $C_0 = (3,3,2)$, avec :
   - $n = 3$;
   - $l_1 = 6$; $\tau_1 = 0$; $m_1 = 3$;
   - $l_2 = 4$; $\tau_2 = 0$; $m_2 = 3$;
   - $l_3 = 2$; $\tau_3 = 0$; $m_3 = 2$.
2. Les variables sont initialisées comme suit :
   $c_b^* = +\infty$; $E = (0,0,0)$; $b' = 3$.
3. L'algorithme classe tout d'abord les types de barres filles par ordre décroissant des longueurs de barres correspondantes. Cela ne change rien dans notre cas.
   Le contenu du vecteur des barres filles pouvant être placées sur la barre mère est alors :
   $C = (3,3,2)$.

4. Il s'agit maintenant de placer b = 3 plus petites barres filles sur la barre mère. L'ensemble des barres filles placées sur la barre mère est initialisé à ∅, c'est-à-dire E = (0,0,0). L'algorithme place alors les plus petites barres filles de type j = n.

4.1. L'algorithme calcule le nombre n_b de barres filles de type **j** pouvant être placées sur la barre. D'après la relation (IV.25) :

$n_b = \min(m_j, b) = 2$.

4.2. Les variables sont remises à jour :

$e_j = 2 \Rightarrow E = (0,0,2)$;

$m_j = 2 - 2 = 0 \Rightarrow C = (3,3,0)$;

$b' = 3 - 2 = 1$.

La fin de la boucle "parcourir" est atteinte, l'algorithme revient à l'étape 4.

4.(1) L'algorithme passe au type de barres filles j = n-1 = 2. Le nombre n_b de barres filles à placer est $n_b = 1$. Les variables sont remises à jour comme pour la boucle précédente :

$e_j = 1 \Rightarrow E = (0,1,2)$;

$m_j = 3 - 1 = 0 \Rightarrow C = (3,2,0)$;

$b' = 1 - 1 = 1$.

L'étape 4 est terminée car **b'** est nulle. L'algorithme passe à l'étape 6.

6. Pour déduire si le placement est réalisable, l'algorithme calcule si le coût ρ(L,E) est inférieur à +∞. D'après la section 2, ρ(L,E) = 6. Ce coût est mis dans c_b^*.

L'algorithme peut continuer dans les niveaux inférieurs de l'étape 5 où il s'agit de permuter la plus courte barre fille de **E** (j = n, ...,1) avec la plus longue de **C'** (j' = 1, ..., n).

6.1. L'algorithme parcourt tous les éléments de **C'** en commençant par j = 1, le type de la plus longue barre fille.

6.1.1. La variable booléenne "Permutation" est mise à Faux.

6.1.2. L'algorithme parcourt tous les éléments de **E** en commençant par j' = n = 3, le type de la plus courte barre fille.

6.1.2.1. Un nouveau placement **E'** est construit :

$E' = E = (0,1,2)$; $e'_j = 2 - 1 = 1$; $e'_{j'} = 0 + 1 = 1$

$\Rightarrow E' = (1,1,1)$.

6.1.2.2. Le coût est calculé d'après la section 2 :

$c_b = \rho(L,E) = 2$.

6.1.2.3. La valeur du critère est meilleure que c_b^*, aussi $c_b^* = c_b$. L'algorithme met à jour les variables utilisées :

$E = E'$;

$m'_j = 0 + 1$; $m'_{j'} = 3 - 1 \Rightarrow C' = (2,2,1)$;

$c_b^* = 2$; "Permutation" = Vrai ; j = 0.

La fin de cette boucle est atteinte, l'algorithme revient à l'étape 6.1.

6.1.(1) La boucle sur la permutation peut continuer car une permutation est réalisée. De la même façon que la boucle précédente, les calculs sont réalisés pour j = 3 et j' = 1. Le coût c_b pour ce placement est $+\infty$, il est inutile de continuer avec ce **j** et ce **j'**.

6.1.(2) Une barre fille de type j = 2 est à échanger avec une barre fille de type j' = 1.

6.1.2.3. Le coût de la découpe est $c_b = 0$. Le meilleur coût est remis à jour, c'est-à-dire $c_b^* = 0$.

Les variables sont maintenant :

E = (2,0,1) ; C' = (1,3,1).

A la boucle suivante, aucune permutation améliorant le coût c_b^* n'est possible aussi l'algorithme par permutations retient ce dernier placement E. Le placement final est donné dans la figure IV.5.

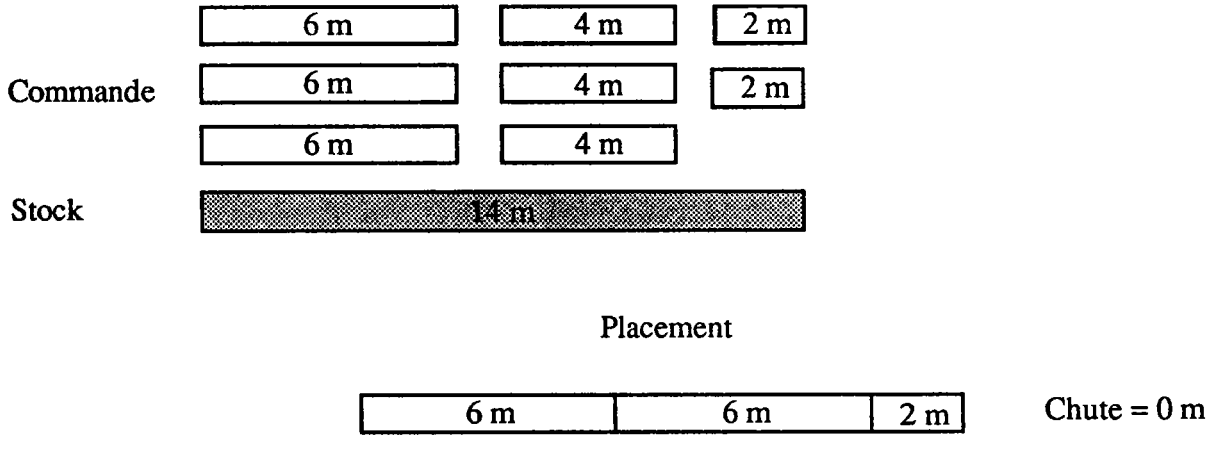

Figure IV.5 : Résulat du placement de l'algorithme par permutations

4.2.4. Conclusion concernant l'algorithme par permutations

La qualité des placements obtenus par cet algorithme est variable. Elle est liée au choix des barres filles à permuter. Le temps de traitement pour cet algorithme est très faible. Il peut donc être utilisé par d'autres algorithmes de placement sans contrainte de temps. Sa rapidité autorise une large exploration des solutions.

5. CONCLUSION

Dans ce quatrième chapitre, nous avons introduit un critère simple de comparaison entre placements locaux. Ce critère prend seulement en compte la génération des chutes et des tombants. Il est utilisé pour évaluer le coût des placements locaux fournis par la résolution des deux types de sous-problèmes de placement sur une barre mère.

En respectant les nouvelles notions telles que les affranchissements, les tombants et les traits de scie, chacun de ces sous-problèmes peut être résolu comme nous l'avons démontré de manière optimale par des algorithmes pseudo-polynomiaux. Dans nos problèmes, ces méthodes exactes nécessitent des temps de calcul trop importants pour être utilisées par la suite par les algorithmes de résolution du problème global.

Par conséquent, nous avons développé des heuristiques dont la qualité est très variable mais dont le temps d'exécution est très faible. Cette rapidité d'exécution est appréciable lorsque nous avons à résoudre un grand nombre de fois ces sous-problèmes. La méthode la plus rapide pour la résolution du premier sous-problème est l'algorithme FFD et pour le second sous-problème, la méthode par permutations.

Chapitre V :

RÉSOLUTION DES PROBLÈMES DE DÉCOUPE

CHAPITRE V

1. INTRODUCTION

Dans le chapitre précédent, nous avons montré les différentes méthodes de résolution des sous-problèmes de placements locaux. Il s'agit de placer sur une barre mère des barres filles choisies parmi un ensemble de barres filles candidates de manière à minimiser les coûts de chutes et de tombant. Nous allons maintenant utiliser ces méthodes de résolution des sous-problèmes pour résoudre le problème global. Le problème est de réaliser toutes les barres filles commandées dans l'ensemble des barres en stock de façon à minimiser une fonction objectif.

Dans certains cas, le critère n'est plus uniquement de minimiser la chute mais également le nombre de paquets. La découpe en paquets consiste à découper plusieurs barres de même longueur en même temps. Ceci a pour objectif de réduire les temps de réglage et de coupe. Cependant, pour une des branches de notre partenaire industriel, nous ne disposons pas de données exploitables en ce qui concerne les coûts de réglage de la machine de coupe, ainsi que le coût d'utilisation de la machine pour évaluer le coût des coupes. Ce dernier coût varie avec la qualité des matériaux, le profil et le nombre de barres découpées en même temps.

Nous optons, dans la section 2, pour l'utilisation de méthodes par construction utilisant des règles de priorité du chapitre II, section 3.1. Ces méthodes ne considèrent que les critères chutes et tombants. Elles optimisent localement les placements. Les solutions obtenues ne sont pas optimales globalement du point de vue des chutes, mais ces méthodes ont pour avantage d'approcher les critères qualitatifs exprimés par notre partenaire industriel.

En revanche lorsque tous les critères ont pu être exprimés en une seule unité de mesure, nous proposons, dans la section 3, des méthodes d'optimisation globale des découpes que nous a inspiré la programmation dynamique.

2. MÉTHODES PAR CONSTRUCTION

2.1. Introduction

Les méthodes par construction ajoutent, à la solution partielle existante, le placement d'une partie des commandes restantes. On obtient ainsi une nouvelle solution partielle. Ce processus se répète jusqu'à ce que toutes les commandes soient réalisées.

A chacune des étapes de la construction de la solution, le problème est de choisir le placement d'une partie des commandes restantes. Il s'agit de résoudre à chaque étape le premier sous-problème (placement de barres filles sur une barre mère).

Afin de comparer les placements locaux issus de la résolution du premier sous-problème, nous proposons, dans la section 2.2, une autre fonction objectif que le coût de génération de la chute et du tombant.

Dans les sections 2.3 et 2.4, nous décrivons deux méthodes ne faisant pas intervenir de paquets. Pour tenir compte du critère priviliégiant la génération de paquets, nous avons développé deux nouvelles méthodes, présentées dans les sections 2.5 et 2.6, qui utilisent en sous-routines les deux méthodes précédentes. Elles cherchent toutes deux à réaliser au mieux des paquets.

Ces méthodes utilisent des règles de priorité. Elles permettent d'obtenir des solutions de bonne qualité.

2.2. Critère de choix des placements locaux

Nous avons vu au chapitre IV, section 2, la définition des critères de choix entre les différents placements. Ce choix est basé principalement sur les chutes et les tombants générés. Cette comparaison directe, si elle est utilisée pour comparer des solutions partielles, amène à des situations paradoxales : pour une même chute, le placement sur une barre mère de petite longueur a le même coût que le placement sur une barre mère de grande longueur. Le pourcentage d'utilisation de la barre mère est très faible pour le premier placement alors qu'il est important dans le second placement.

Pour illustrer ce point de vue, nous prendrons un exemple simple où il faut choisir entre 2 placements possibles :

1. une barre mère de longueur 10 m est découpée en générant une chute totale de 1 m ;
2. une autre barre mère de longueur 20 m est découpée en générant une chute totale de 1,1 m.

L'algorithme doit choisir le placement local qui lui propose le coût le plus faible (qui se limite dans notre cas à la chute). Par conséquent, le premier placement va être préféré au second. Nous constatons que dans cette situation, le pourcentage de chute générée en choisissant le premier placement local (10 %) est très supérieur à celui du deuxième placement (5,5 %).

Nous allons introduire ici un autre critère de choix pour départager deux placements locaux sur deux barres mères données. Il s'agit de maximiser le rendement des barres utilisées.

Le rendement η' de la découpe des éléments du vecteur **E** des barres filles dans une barre mère est calculé comme suit :

$$\eta' = \frac{\sum_{1 \le j \le n} (e_j l_j)}{(L - t) + c(t)} \qquad (V.1)$$

où :

- **n** est le nombre de types différents de barres filles ;
- $\mathbf{l_j}$ est la longueur de la barre fille d'indice **j** ;

- **ej** est le nombre de barres filles d'indice **j** placées sur la barre mère ;
- **L** est la longueur de la barre mère découpée ;
- **t** est la longueur de la partie non utilisée de la barre mère ;
- **c(t)** est le coût (en longueur de chute) de la partie non utilisée de la barre mère. Ce coût correspond au coût d'une chute de longueur **t** s'il s'agit d'une chute, sinon il correspond au coût d'un tombant de longueur **t** (cf. relation (III.2)).

Cette nouvelle fonction coût permet de faire la distinction entre deux placements de chutes voisines. Pour une quantité donnée de chutes, le rendement est d'autant plus grand que la barre mère découpée est longue.

Comme il s'agit d'heuristiques appliquées à des problèmes NP-difficiles, nous ne pouvons pas garantir qu'une fonction coût soit meilleure qu'une autre. Une option du programme permet de choisir entre la minimisation de la chute et la maximisation du rendement. Le coût du placement trouvé est pondéré par le paramètre θ (≥ 1) qui favorise l'utilisation des tombants (cf. chapitre III, section 3.3).

$$\eta = \begin{cases} \theta \times \eta' & \text{si la barre mère est un tombant ;} \\ \eta' & \text{sinon} \end{cases} \qquad \text{(V.2)}$$

où **T** est égal à 1 si la barre mère est un tombant sinon **T** est nul.

2.3. Algorithme First Fit Decreasing (FFD) étendu

2.3.1. Description de l'algorithme FFD étendu

Rappelons que la méthode FFD (First Fit Decreasing) consiste à classer les barres filles commandées dans l'ordre décroissante de leur longueur. Ensuite, celles-ci sont placées si possible dans cet ordre sur une barre mère. Les barres mères de longueur identique sont découpées de cette manière, les unes après les autres, tant qu'il reste des barres filles à réaliser.

Au sens strict, la méthode FFD ne considère comme disponible en stock qu'une seule longueur de barres mères. Par conséquent, elle n'appréhende pas la diversité des stocks disponibles.

L'extension de cette méthode permet, à partir de la connaissance du stock réel et des commandes, de déterminer les coupes à effectuer afin de minimiser la chute. A chaque itération, l'algorithme considère les placements locaux pour chaque type de barre mère. Le meilleur placement local est retenu pour construire pas à pas le placement global.

2.3.2. Algorithme FFD étendu

Schématiquement, l'algorithme FFD étendu est le suivant :

ALGORITHME FFD ÉTENDU

1. Données
- *C_0, vecteur des barres filles candidates dont chaque composante correspond à la quantité de barres filles de chaque type commandée*
- *B_0, vecteur des barres mères disponibles dont les composantes correspondent aux quantités de barres mères de chaque type disponibles*

2. Initialiser
- *$C = C_0$, vecteur des barres filles candidates dont chaque élément j correspond à la quantité m_j de barres filles de type j restant à découper, avec $j = 1,...,n$*
- *$B = B_0$, vecteur des barres mères disponibles dont les composantes M_i correspondent aux quantités de barres mères de chaque type, avec $i = 1,...,N$*

3. Classer les types de barres filles par ordre décroissant de leurs longueurs l_j avec $j = 1, ..., n$

4. Tant que toutes les barres filles ne sont pas réalisées et qu'il reste des barres mères, faire

4.1. Initialiser $c_b^* = +\infty$ (c_b^* contiendra la meilleure valeur du critère pour une barre mère)

4.2. Pour tout type i de barres mères, i variant de 1 à N

4.2.1. Réaliser le placement sur la barre mère de type i dans l'ordre prédéfini (FFD). Le placement trouvé est stocké dans E

4.2.2. Calculer la valeur du critère c_b (chute ou rendement selon le cas)

4.2.3. Si $c_b < c_b^*$ (dans le cas de la minimisation de la chute) ou $c_b > c_b^*$ (dans le cas de la maximisation du rendement), alors

Retenir ce placement local dans E^* ainsi que le type de la barre mère dans i^*

4.3. Fin de boucle

4.4. Retenir le placement local donnant la meilleure valeur du critère c_b^* et retirer la barre mère et les barres filles retenues respectivement de B et de C

5. Fin tant que.

Cette méthode est très rapide, mais le principe peut être encore amélioré.

2.3.3. Amélioration de la méthode

2.3.3.1. Permutation des barres filles

La méthode précédente est fonction de l'ordre dans lequel les barres filles sont classées au début de l'algorithme. Le fait de permuter deux ou plusieurs barres filles peut diminuer la chute finale.(ou augmenter le rendement)

Exemple :

Nous disposons en stock de barres mères de longueur 10 m et 18 m. Nous désirons réaliser deux barres filles de longueurs respectives 10 m et 7 m. Comme les barres filles sont déjà classées par ordre des longueurs décroissantes, elles sont conservées dans cet ordre dans la liste des barres filles à réaliser.

Dans la première étape, l'heuristique FFD étendue va scruter un à un les types de barres mères disponibles :

1. Sur la barre mère de longueur 10 m, l'algorithme va placer en premier la barre fille de longueur 10 m. Comme ce qui reste de la barre mère est de longueur nulle, il arrête le placement sur cette barre mère. Le coût calculé pour ce placement local est 0 (**0 m** de chute).
2. Dans la barre de longueur 18 m, il va pouvoir découper les 2 barres filles (10 m et 7 m) et générer **1 m** de chute.

Dans la figure V.1, nous représentons les deux choix de découpe.

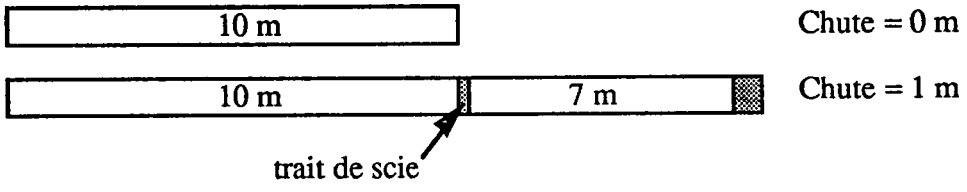

Figure V.1 : Choix pour le premier placement

La solution locale retenue correspond au premier placement où la chute générée est la plus faible.

Pour la deuxième étape, il reste à placer la dernière barre fille de 7 m, l'algorithme FFD étendu a deux choix, découper :

1. une autre barre mère de 10 m et réaliser une chute de **3 m** ;
2. une barre mère de 18 m et réaliser une chute de **11 m** si nous considérons que la génération de tombants n'est pas autorisée.

Les choix pour l'algorithme sont représentés dans la figure V.2.

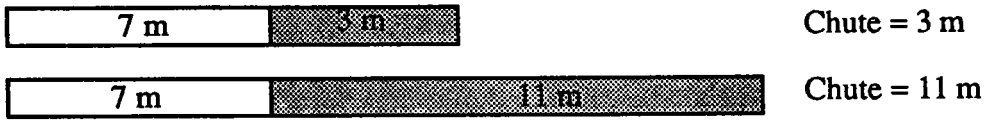

Figure V.2 : Choix pour le second placement

La solution retenue est la première. La chute totale de la découpe est de 0+3 = **3 m**.

En permutant les 2 types de barres filles, la liste des barres filles à réaliser devient {7 m,10 m}. L'algorithme procède de la manière suivante :

1. Sur la barre mère de 10 m, l'algorithme teste le placement de la barre fille de 7 m. La longueur non utilisée (**3 m**) de la barre mère est trop faible pour être réutilisée, elle est considérée comme une chute.
2. Sur la barre mère de 18 m, l'algorithme peut placer les 2 barres filles et générer **1 m** de chute.

Les deux choix de découpe sont représentés dans la figure V.3.

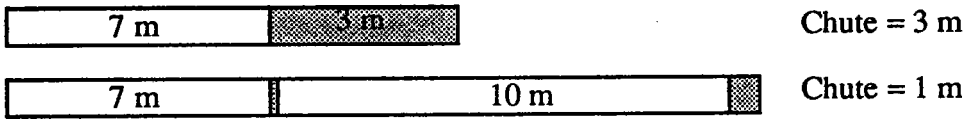

Figure V.3 : Choix pour le premier placement après permutation

Entre ces deux placements locaux, l'algorithme choisit la solution engendrant le moins de chute, à savoir la 2ème solution qui consiste à placer les 2 barres filles sur une barre mère de 18 m. La chute générée par ce placement local est de 1 m. Comme il n'y a plus de barres filles à découper, la chute finale de la découpe est la chute de l'unique placement local, i.e. **1 m**.

Nous voyons qu'après la permutation des types de barres filles, la chute finale est de **1 m** au lieu de **3 m** avant la permutation. Les deux placements sont illustrés dans la figure V.4.

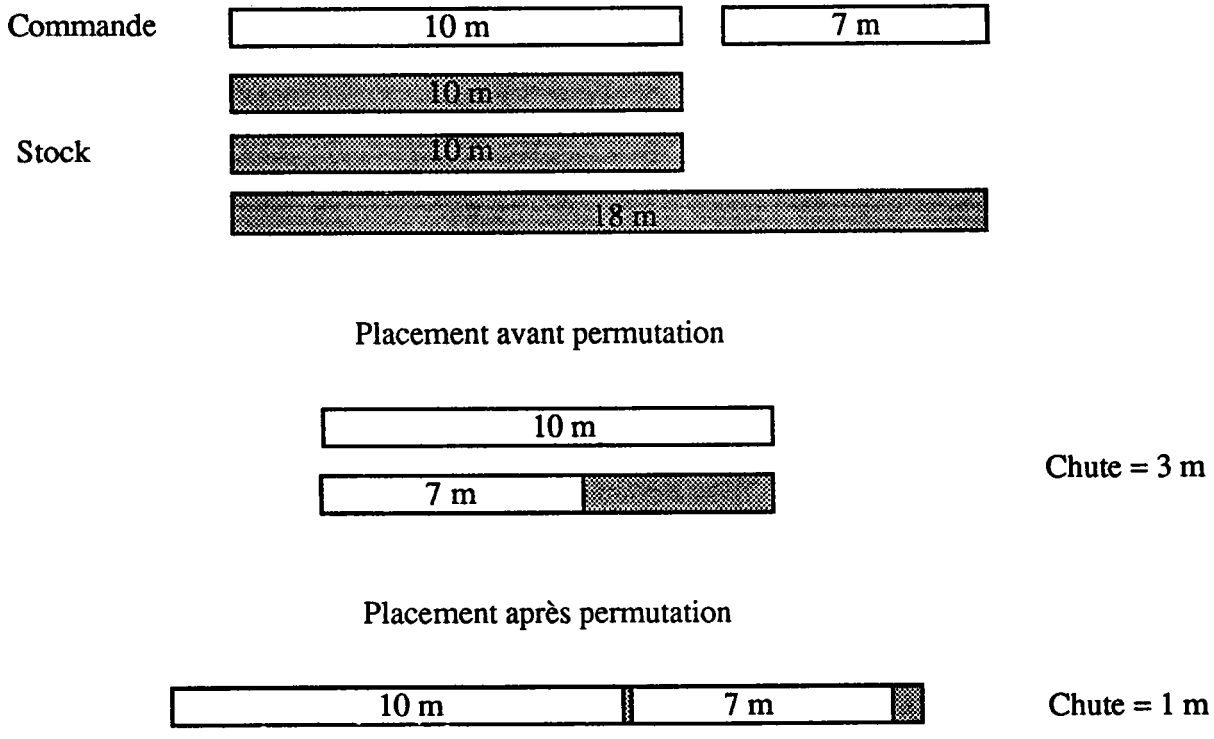

Figure V.4 : Résulat du placement de l'algorithme par permutations

Nous avons mis en place un module permettant de générer toutes les permutations des types de barres filles pour améliorer les résultats de l'heuristique du FFD étendue. Le nombre de

permutations possibles est très important. Lorsque le nombre de types de barres filles est faible (< 8), le programme énumère toutes les permutations à savoir 8! = 40320 permutations. En revanche, pour un nombre supérieur de types de barres filles, nous générons aléatoirement les permutations. Le programme arrêtera les recherches après un temps fixé au départ. Il restituera à ce moment là le meilleur placement trouvé.

2.3.3.2. Répétition du placement local

Pour réduire le nombre de boucles explorées, après avoir trouvé le meilleur coût c_b^* pour un placement local, nous cherchons à réaliser ce placement local autant de fois que possible .

Soit E^* le placement donnant le coût c_b^*. E^* est un vecteur dont les dimensions correspondent aux types de barres filles commandées et dont les valeurs correspondent au nombre de barres filles réalisées dans ce placement. L'algorithme recherche le nombre n_f de fois que le placement E^* peut être reproduit. Ce nombre est limité par le nombre de barres filles à réaliser et par le nombre de barres mères de type i^* disponibles.

$$n_f = \min\left(\min_{1\le j\le M / e_j^* \neq 0} \left\lfloor \frac{m_j}{e_j^*} \right\rfloor, M_{i^*} \right) \qquad (V.3)$$

où e_j^* est la $j^{ième}$ valeur du vecteur E^*.

Dès lors, il n'est plus nécessaire de parcourir à nouveau tous les types de barres du stock pour arriver au même placement local. Le même placement local peut se répéter un grand nombre de fois lorsque les nombres de barres mères et de barres filles de chaque type sont grands.

Le gain de temps est appréciable surtout lorsqu'on veut effectuer un grand nombre de permutations des types de barres filles dans la liste.

Pour simplifier les écritures, mettons le type i^* de barres mères sous forme vectorielle pour les calculs des vecteurs.

$$K(i^*) = (k_i)_{1\le i\le N} \qquad \text{avec} \quad k_i = \begin{cases} 1 & \text{si } i = i^* ; \\ 0 & \text{sinon.} \end{cases} \qquad (V.4)$$

où N est le nombre de types différents de barres mères.

Décrivons à présent l'algorithme FFD étendu intégrant ces deux améliorations.

2.3.4. Algorithme FFD étendu

Avec ces changements, l'algorithme FFD étendu devient :

ALGORITHME FFD ÉTENDU

1. Données

- *C_0, vecteur des barres filles candidates dont chaque élément j correspond à la quantité de barres filles de type j à découper, avec j = 1,...,n*
- *B_0, vecteur des barres mères disponibles dont la $i^{ème}$ composante correspond à la quantité de barres mères de type i disponible, avec i = 1,...,N*

2. Initialiser

- *$\chi^* = +\infty$, meilleure valeur du critère pour le placement global*
- *$n_p' = 0$, nombre de permutations réalisées*

3. Classer les types de barres filles par ordre décroissant de leurs longueurs l_j avec j = 1, ..., n. Les composantes du vecteur C_0 sont réarrangées dans cet ordre

4. Calculer le nombre maximal de permutations

$$n_p = n! \qquad (V.5)$$

5. Tant que toutes les permutations ne sont pas parcourues ou que le temps maximal de traitement n'est pas atteint faire

5.1. Initialiser

$C = C_0$; $B = B_0$ *et* $\chi = 0$

où C (resp. B) est le vecteur des barres à réaliser (resp. disponibles) dont les composantes m_j (resp. M_j) correspondent aux nombres de barres à réaliser (disponibles) de type j

5.2. Si n < 8, alors

Générer une permutation des types de barres filles de C en fonction de n_p' et incrémenter n_p'

5.3. Sinon

Générer aléatoirement une permutation des types de barres filles de C

5.4. Tant que toutes les barres filles ne sont pas réalisées et qu'il reste des barres mères, faire

5.4.1. Initialiser $c_b^* = +\infty$ (*c_b^* contiendra la meilleure valeur du critère pour une barre mère)*

5.4.2. Pour tout ***type i de barres mères****, i variant de 1 à N, avec $M_i > 0$*

5.4.2.1. Réaliser le placement sur la barre mère de type i dans l'ordre prédéfini (FFD). Le placement trouvé est stocké dans E

5.4.2.2. Calculer la valeur du critère c_b (chute ou rendement selon le cas)

5.4.2.3. Si $c_b < c_b^*$ (dans le cas de la minimisation de la chute) ou $c_b > c_b^*$ (dans le cas de la maximisation du rendement), alors

Retenir ce placement local dans E^* ainsi que le type de la barre mère dans i^*

5.4.3. Fin de boucle

5.4.4. Calculer le nombre n_f de fois que le placement local peut être réalisé

5.4.5. Mettre à jour les variables

- $\chi \Leftarrow \chi + n_f \times c_b^*$ *(V.6)*
- $C \Leftarrow C - n_f \times E^*$ *(V.7)*
- $B \Leftarrow B - n_f \times K(i^*)$ *(V.8)*

5.5 Fin tant que

5.6. Si $\chi < \chi^*$, alors

Retenir le placement

6. Fin tant que.

2.3.5. Exemple d'application de l'algorithme FFD étendu

Les données sont :

Stock :

- 3 barres mères de longueur 10 m ;
- 3 barres mères de longueur 15 m.

Commande :

- 3 barres filles de longueur 6 m à tolérance normale ;
- 6 barres filles de longueur 4 m à tolérance normale.

La largeur du trait de scie est 0,01 m.

Le nombre de traits de scie pouvant être négligés dans le cas d'une découpe de barres filles à tolérance normale est 2.

Pour une description simple du mécanisme de l'algorithme, nous choisissons de comparer les placements locaux avec les critères chutes et tombants décrits au chapitre IV, section 2.

1. Les données de départ sont :
   - $C_0 = (3,6)$, avec :
     $l_1 = 6$; $\tau_1 = 0$; $m_1 = 3$;
     $l_2 = 4$; $\tau_2 = 0$; $m_2 = 6$;
   - $B_0 = (3,3)$, avec :
     $L_1 = 10$; $T_1 = 0$; $M_1 = 3$;
     $L_2 = 15$; $T_2 = 0$; $M_2 = 3$.
2. Les variables sont initialisées comme suit :
   $C = C_0 = (3,6)$; $B = B_0 = (3,3)$; $\chi^* = +\infty$; $n_p' = 0$.
3. L'algorithme classe tout d'abord les types de barres filles par ordre décroissant des longueurs de barres correspondantes, avec $n = 2$. Ceci est déjà le cas pour les composantes du vecteur $C_0 = (3,6)$.
4. Le nombre de permutations possibles pour $n = 2$ est $n_p = 2$.
5. L'algorithme entre dans la procédure de placement.

CHAPITRE V

5.1. Les variables de la procédure sont initialisées :
$C = (3,6)$; $B = (3,3)$; $\chi = 0$.

5.2. Comme le nombre $n = 2$ de types de barres filles est inférieur à 8, l'algorithme va considérer toutes les permutations de l'ordre des types de barres filles. Rappelons que pour obtenir un placement de "bonne" qualité dès le départ, les types de barres filles sont classés par ordre décroissant des longueurs des barres filles correspondantes (étape 3).

5.4. L'algorithme commence ici le placement des barres filles du vecteur **C** sur les barres mères du vecteur **B**.

5.4.1. Le meilleur coût c_b^* est initialisé à $+\infty$.

5.4.2. L'algorithme parcourt chacun des types **i** de barres disponibles en stock. Il commence donc par le type de barres mères $i = 1$.

5.4.2.1. D'après la méthode FFD (cf. chapitre IV, section 3.2), le placement retenu est : $E = (1,1)$.
Les valeurs du vecteur **E** correspondent au nombre de barres filles de chacun des types placées sur la barre mère de type 1.

5.4.2.2. D'après le chapitre IV, section 2, le coût de la découpe de **E** dans la barre mère de type 1 est :
$c_b = \rho(L_i,E) = 0$.

5.4.2.3. Ce coût c_b est meilleur que c_b^*. La meilleure valeur du critère devient $c_b^* = 0$. Ce placement est retenu dans E^* ainsi que le type i^* de la barre mère choisie : $E^* = E$ et $i^* = 1$.
La boucle est terminée, l'algorithme revient au début de la boucle 5.4.2.

5.4.2.(1) Pour le type $i = 2$ de barres mères, d'après la méthode FFD (cf. chapitre IV, section 3.2), le placement retenu est : $E = (2,0)$

5.4.2.2. Le coût de la découpe de la barre mère de type 2 est :
$c_b = \rho(L_2,E) = L_2 - 2 \times l_1 = 3$.
La meilleure valeur du critère n'est pas meilleure que celle déjà retenue. Par conséquent, ce placement est rejeté.
Il ne reste plus de type de barres mères à parcourir. Le meilleur placement local est trouvé pour le type 1 de barres mères.

5.4.4. L'algorithme recherche à présent le nombre n_f de fois que ce placement peut être reproduit. D'après la relation (V.3), $n_f = 3$.

5.4.5. Les nouvelles données sont :

- pour le coût des placements locaux réalisés jusqu'à présent :
  $\chi = 0$;

- pour le vecteur des barres filles restant à réaliser :

  C = (0,3) ;

- pour le vecteur des barres mères restant en stock :

  B = (0,3) avec K(1) = (1,0).

Les barres filles ne sont pas toutes réalisées. L'algorithme continue à parcourir la boucle 5.4.

5.4.(1) Le meilleur coût c_b^* pour une barre mère est initialisé à $+\infty$. Les barres mères de type 1 ne sont plus disponibles, l'algorithme continue le placement avec le type i = 2 de barres mères.

5.4.2.1. D'après la méthode FFD, le placement est : E = (0,3).

5.4.2.2. Le coût de la découpe de la barre mère de type 2 est :

$c_b = \rho(L_i,E) = L_2 - 3\times l_1 = 3.$

Ce placement améliore la fonction économique. Le meilleur coût est maintenant $c_b^* = 3$ et ce placement est retenu :

$E^* = E$ et $i^* = 2$.

5.4.4. Le nombre n_f de fois que ce placement peut être reproduit est 1. Les données deviennent :

$\chi = 3$; C = (0,0) ; B = (0,2).

Il n'y a plus de barres filles à réaliser. L'algorithme recommence toute cette démarche en permutant les types de barres filles à réaliser. Dans ces autres cas, les solutions ne sont pas meilleures. Le placement final est donné dans la figure V.5.

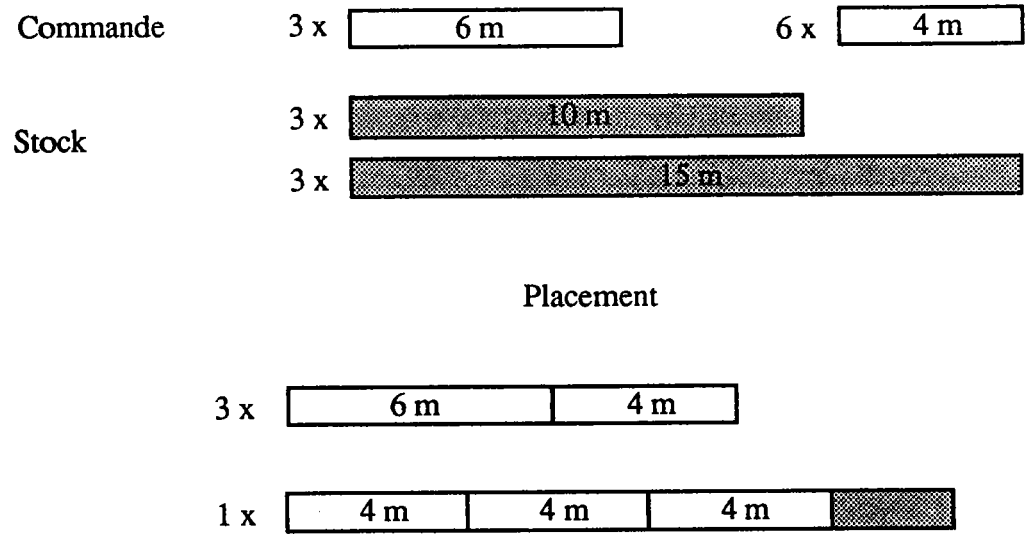

Figure V.5 : Résulat du placement de l'algorithme FFD étendu

2.3.6. Conclusion relative à l'algorithme FFD étendu

La méthode est très rapide même pour des problèmes mettant en jeu un nombre important de barres filles et de barres mères. Par conséquent, elle permet de parcourir un nombre élevé de permutations. Au final, la qualité des solutions proposées est très satisfaisante. Il faut remarquer que les permutations sont réalisées pour les types de barres filles. Pour améliorer encore la qualité des placements trouvés par cet algorithme, il faudrait non plus considérer toutes les permutations des types de barres filles mais considérer les barres filles comme toutes différentes, et parcourir toutes les permutations de ces barres filles. Cela n'est pas envisageable dans la pratique car le temps de traitement augmenterait exponentiellement avec le nombre de barres filles commandées. En effet, le nombre de permutations à considérer serait n! (où **n** est le nombre total de barres filles commandées).

Cet algorithme destiné à l'aide à la décision est particulièrement apprécié par notre partenaire industriel pour :

- sa **rapidité** (de l'ordre de quelques secondes[1]) à résoudre les problèmes de découpe de grandes tailles (plus de 1000 barres filles de 15 types différents à découper dans 300 barres mères de 15 types différents) ;
- la **qualité** des placements trouvés (cf. tableau V.2).

Ces deux qualités sont importantes lors du passage de commandes par un client et qu'il faut réaliser le devis de la découpe.

2.4. Algorithme de l'arborescence limitée étendu

2.4.1. Description de l'algorithme de l'arborescence limitée étendu

De même que dans la méthode précédente, nous étendons la méthode de recherche de l'optimum local sur un seul type de barres du stock à tous les types disponibles. Nous intégrons dans la construction du placement final la simplification décrite à la section 2.3, concernant la répétition du placement trouvé.

2.4.2. Algorithme de l'arborescence limitée étendu

Schématiquement, l'algorithme de l'arborescence limitée étendu est le suivant :

ALGORITHME DE L'ARBORESCENCE LIMITÉE ÉTENDU
1. Données • *C_0, vecteur des barres filles candidates dont chaque élément j correspond à la quantité de barres filles de type j à découper, avec j = 1,...,n* • *B_0, vecteur des barres mères disponibles dont les composantes correspondent aux quantités de barres mères de chaque type*

[1] Temps d'exécution sur un Sun Sparcstation 10

2. *Initialiser*
   - $C = C_0$, *vecteur des barres filles candidates dont chaque élément* m_j *est la quantité de barres filles de même indice restant à découper, avec* $j = 1,...,n$
   - $B = B_0$, *vecteur correspondant aux quantités* M_i *de barres mères de chaque type encore disponibles, avec* $i = 1,...,N$
   - $\chi = 0$, *meilleure valeur du critère pour le placement global*
3. *Tant que toutes les barres filles ne sont pas réalisées et qu'il reste des barres mères, faire*
   - 3.1. *Initialiser* $c_b^* = +\infty$ (c_b^* *contiendra la meilleure valeur du critère pour une barre mère)*
   - 3.2. *Pour tout* ***type i de barres mères,*** *avec i variant de 1 à N et* $M_i > 0$
     - 3.2.1. *Réaliser le placement par la méthode de l'arborescence limitée (cf. chapitre IV, section 3.3). Le placement local est E*
     - 3.2.2. *Calculer la valeur du critère* c_b *(chute ou rendement selon le cas)*
     - 3.2.3. *Si* $c_b < c_b^*$ *(dans le cas de la minimisation de la chute) ou* $c_b > c_b^*$ *(dans le cas de la maximisation du rendement), alors*

       Retenir ce placement local dans E^* *et faire* $i^* = i$
   - 3.3. *Fin de boucle*
   - 3.4. *Chercher le nombre* n_f *de fois que le placement local donnant la meilleure valeur du critère* c_b^* *peut être réalisé*
   - 3.5. *Mettre à jour les variables*
     - $\chi \Leftarrow \chi + n_f \times c_b^*$
     - $C \Leftarrow C - n_f \times E^*$
     - $B \Leftarrow B - n_f \times K(i^*)$
4. *Fin tant que.*

2.4.3. *Exemple d'application de l'algorithme de l'arborescence limitée étendu*

Les données sont :

Stock :
- 3 barres mères de longueur 14 m ;
- 3 barres mères de longueur 9 m.

Commande :
- 3 barres filles de longueur 6 m à tolérance normale ;
- 8 barres filles de longueur 4 m à tolérance normale.

La largeur du trait de scie est 0,01 m.

Le nombre de traits de scie pouvant être négligés dans le cas d'une découpe de barres filles à tolérance normale est 2.

CHAPITRE V

1. Les données de départ sont :
   - $n_S = 2$; $\gamma = 0{,}01$;
   - $C_0 = (3,8)$, avec :
     $l_1 = 6$; $\tau_1 = 0$;
     $l_2 = 4$; $\tau_2 = 0$;
   - $B_0 = (3,3)$, avec :
     $L_1 = 14$; $T_1 = 0$;
     $L_2 = 9$; $T_2 = 0$.
2. Les variables sont initialisées comme suit :
   $C = C_0 = (3,8)$ avec $m_1 = 3$ et $m_2 = 8$;
   $B = B_0 = (3,3)$ avec $M_1 = 3$ et $M_2 = 3$;
   $\chi = 0$.
3. L'algorithme commence le placement des barres filles du vecteur **C** sur des barres mères du vecteur **B**.
   3.1. Le meilleur coût c_b^* est initialisé à $+\infty$.
   3.2. L'algorithme parcourt chacun des types **i** de barres disponibles en stock en commençant par i = 1.
      3.2.1. D'après l'algorithme de l'arborescence limitée (cf. chapitre IV, section 3.3), le placement retenu est : E = (1,2).
      3.2.2. D'après le chapitre IV, section 2, le coût de la découpe de la barre mère de type 1 est :
      $c_b = \rho(L_i,E) = 0$.
      3.2.3. Ce coût c_b est meilleur que c_b^* et il est affecté à la meilleure valeur du critère, par conséquent $c_b^* = 0$. Ce placement local est retenu :
      $E^* = (1,2)$ et $i^* = 1$.
      La fin de la boucle est atteinte, l'algorithme revient à l'étape 3.2.
   3.2.(1) Pour le type i = 2 de barres mères, d'après l'algorithme de l'arborescence limitée, le placement retenu est : E = (1,0).
   Le coût de la découpe de la barre mère de type 2 est $c_b = L_2 - 1 \times l_1 = 3$. La meilleure valeur du critère n'est pas améliorée, par conséquent ce placement n'est pas retenu.
   Il ne reste plus de type de barres mères à parcourir. Le meilleur placement local est trouvé pour le type 1 de barres mères.
   3.4. L'algorithme recherche à présent le nombre n_f de fois que ce placement peut être reproduit. D'après la relation (V.3) : $n_f = 3$.
   3.5. Les nouvelles données sont :
   d'après la relation (V.6) $\chi = 0$;
   d'après la relation (V.7) $C = (0,2)$;

d'après la relation (V.8) B = (0,3).

Les barres filles ne sont pas toutes réalisées. L'algorithme continue dans la boucle 3.

3.(1) Le meilleur coût c_b^* pour une barre mère est initialisé à $+\infty$. Les barres mères de type 1 ne sont plus disponibles, l'algorithme continue le placement avec le type i = 2 de barres mères.

3.2.1. D'après l'algorithme de l'arborescence limitée : E = (0,2).

Le coût de la découpe de la barre mère de type 2 est $c_b = L_2 - 2 \times l_1 = 1$. Ce placement améliore la fonction économique. Le meilleur coût est maintenant $c_b^* = 1$ et ce placement est retenu : $E^* = (0,2)$ et $i^* = 2$.

Le nombre de fois que ce placement peut être reproduit est $n_f = 1$. Les données deviennent :

$\chi = 1$; C = (0,0) ; B = (0,2).

L'algorithme s'arrête car il n'y a plus de barres filles à réaliser. Le coût final est $\chi = 1$. Le placement final est donné dans la figure V.6.

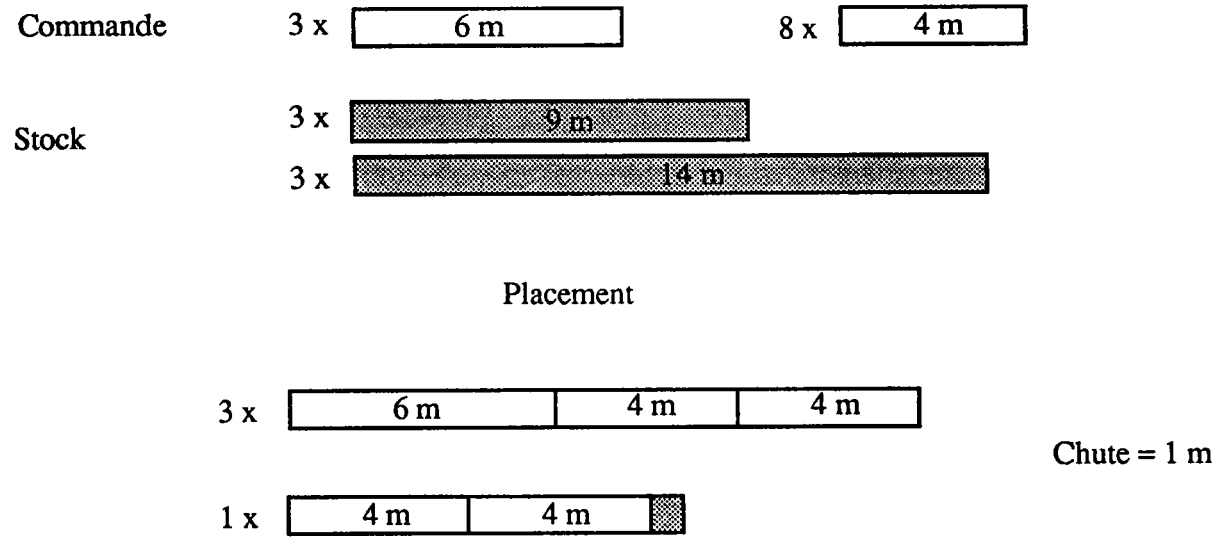

Figure V.6 : Résulat du placement de l'algorithme de l'arborescence limitée étendu

2.4.4. Conclusion concernant l'algorithme de l'arborescence limitée étendu

Cette méthode peut demander un temps de calcul important lorsque le nombre de barres en stock et le nombre de barres commandées sont grands. Ceci est aussi directement lié au nombre de types de barres filles et au nombre de types de barres mères. Dans le programme implanté, nous basculons automatiquement cette méthode sur la méthode FFD étendue lorsque les nombres de types de barres sont trop importants ($N \times M > 5 \times 7 = 35$).

2.5. Méthode locale par paquets

2.5.1. Description de la méthode locale par paquets

La première approche, développée pour minimiser le nombre de paquets réalisés, consiste à appliquer différentes tailles de paquets aux différents types de barres mères. Pour chaque type de barres mères, on cherche la taille du paquet qui minimise la fonction coût. Le coût de coupe d'un paquet de taille **s** n'est pas connu. Par conséquent, nous pondérons le coût total du paquet par une fonction décroissante de la taille du paquet et dont la pente est variable suivant le choix de l'utilisateur. Nous pouvons prendre une fonction simple telle que :

$$f(s) = 1 - \frac{s}{\sum_{1\le j\le n}(m_j l_j) + \varphi} \qquad (V.9)$$

où :

- **s** est le nombre de barres mères dans ce paquet ;
- **n** est le nombre de types différents de barres filles ;
- m_j est le nombre de barres filles de type j placées sur une barre mère du paquet ;
- φ est un paramètre de réglage de la fonction. Elle a été fixée à 10 après de nombreux essais.

Pour reduire le nombre d'itérations, l'algorithme ne parcourt que les tailles de paquets appartenant à l'ensemble des nombres premiers. Les paquets ayant les mêmes découpes sont regroupés en fin de placement.

On a la possibilité d'utiliser, pour l'optimisation des placements, un des deux algorithmes décrits précédemment, c'est-à-dire l'algorithme FFD ou l'algorithme de l'arborescence limitée. L'algorithme choisit, à chaque étape, le type de barres mères qui donne le meilleur résultat.

2.5.2. Algorithme local par paquets

Cet algorithme local par paquets s'écrit :

ALGORITHME LOCAL PAR PAQUETS
1. Données • *C_0, vecteur des barres filles candidates dont chaque élément j correspond à la quantité de barres filles de type j à découper, avec j = 1,...,n* • *B_0, vecteur des barres mères disponibles dont la $i^{ème}$ composante correspond à la quantité de barres mères de type i, avec i = 1,...,N*
2. Initialiser • *$C = C_0$, vecteur des barres filles candidates dont chaque élément m_j est la quantité de barres filles de même indice restant à découper, avec j = 1,...,n* • *$B = B_0$, vecteur correspondant aux quantités M_i de barres mères de chaque type, encore disponibles avec i = 1,...,N*

- $\chi = 0$, *meilleure valeur du critère pour le placement global*

3. Tant que toutes les barres filles ne sont pas réalisées et qu'il reste des barres mères, faire

3.1. Initialiser c_p^* *à* $+\infty$ (c_p^* *correspond à la meilleure valeur du critère local pour une barre mère du paquet)*

3.2. Pour tout ***type i de barres mères****, i variant de 1 à N*

3.2.1. Pour toute ***taille s de paquets*** *de barres de type i, s variant de 1 à* M_i *et s appartient à l'ensemble des nombres premiers*

3.2.1.1. Construire un problème avec une commande fictive $C' = \left\lfloor \frac{C}{s} \right\rfloor$

3.2.1.2. Résoudre le problème de placement des barres filles de C' sur une barre mère de type i par l'algorithme FFD ou l'algorithme de l'arborescence limitée. Le placement local est stocké dans E

=> le placement obtenu correspond à la découpe d'un paquet de s barres mères de type i

3.2.1.3. Calculer le coût du placement correspondant à un paquet de s barres mères de type i :

$$c_p = \rho(L_i,E)\times f(s) \qquad (V.10)$$

(f(s) est une fonction décroissante de la taille de paquets)

3.2.1.4. Si $c_p < c_p^*$ *alors*

$$c_p^* = c_p \; ; \; E^* = E \; ; \; s^* = s \; ; \; i^* = i$$

3.2.2. Fin boucle pour

3.3. Fin boucle pour

3.4. Mettre à jour les variables

$$\chi \Leftarrow \chi + s^* \times c_p^*$$

$$B \Leftarrow B - s^* \times K(i^*)$$

$$C \Leftarrow C - s^* \times E^*$$

4. Fin tant que.

2.5.3. Exemple d'application de la méthode locale par paquets

Les données sont :

Stock :

- 10 barres mères de longueur 15 m ;
- 10 barres mères de longueur 23 m.

Commande :

- 19 barres filles de longueur 6 m à tolérance normale ;
- 23 barres filles de longueur 4 m à tolérance normale.

La largeur du trait de scie est 0,01 m.

Le nombre de traits de scie pouvant être négligés dans le cas d'une découpe de barres filles à tolérance normale est 2.

Utilisons par exemple, l'algorithme de l'arborescence limitée pour définir les placements locaux (cf. chapitre IV, section 3.3).

1. Les données de départ sont :
   - $n_S = 2$; $\gamma = 0,01$;
   - $C_0 = (19,23)$, avec :
     $l_1 = 6$; $\tau_1 = 0$; $m_1 = 19$;
     $l_2 = 4$; $\tau_2 = 0$; $m_2 = 23$;
   - $B_0 = (10,10)$, avec :
     $L_1 = 15$; $T_1 = 0$; $M_1 = 10$;
     $L_2 = 23$; $T_2 = 0$; $M_2 = 10$.
2. Les variables sont initialisées comme suit :
   $C = C_0 = (19,23)$; $B = B_0 = (10,10)$; $\chi = 0$.
3. L'algorithme commence le placement des barres filles du vecteur **C** sur des barres mères du vecteur **B**.
   3.1. Le meilleur coût c_p^* est initialisé à $+\infty$.
   3.2. L'algorithme parcourt chacun des types **i** de barres disponibles en stock en commençant par i = 1.
      3.2.1. L'algorithme parcourt toutes les tailles **s** de paquets en commençant par s = 1.
         3.2.1.1. Comme s = 1, il n'y a pas de division du nombre de barres mères et du nombre de barres filles : C' = C.
         3.2.1.2. Le placement effectué par l'algorithme de l'arborescence limitée donne :
         E = (1,2)
         où **E** est le vecteur des quantités de chaque type de barres filles placées sur la barre mère de type i = 1.
         3.2.1.3. Si, pour simplifier, nous n'autorisons pas la génération de tombants, le coût de la découpe d'une barre mère se réduit au coût des chutes (c'est le cas pour notre exemple). Le coût c_b pour ce placement local est donné par :

$$c_b = \rho(L_i,E) = L_i - \sum_{j \in E} l_j \qquad (V.11)$$

         d'où : $c_b = 1$.

Le coût c_p du paquet est fonction du coût pour une barre mère du paquet pondéré par la fonction f(s) décroissante de la taille s du paquet :

$c_p = c_b \times f(s)$

d'après la relation (V.9) $f(1) = 1 - \frac{1}{206+10} = 0,995$

d'où : $c_p = 0,995.$

3.2.1.4. Ce coût c_p est meilleur que c_p^*. Il est retenu comme le meilleur coût pour un paquet : $c_p^* = 0,995.$

Les autres variables sont remises à jour :

$E^* = (1,2)$; $s^* = 1$; $i^* = 1$.

L'algorithme continue de la même manière jusqu'à la taille de paquets s = 10.

...

3.2.1.(4) Pour une taille s = 7 de paquets, en divisant le nombre de barres filles de chaque type par 7, le nouveau vecteur des barres filles est : C' = (2,3).

3.2.1.2. Le placement effectué par l'algorithme de l'arborescence limitée donne :

E = (1,2).

3.2.1.3. Il n'y a que la fonction f(s), d'après la relation (V.9), qui change :

$f(7) = 1 - \frac{7}{206+10} = 0,968$; $c_b = 1$

d'où : $c_p = 0,968.$

3.2.1.4. c_p est le meilleur coût pour un paquet de barres mères de type i = 1 :

$c_p^* = 0,968$; $i^* = 1$; $E^* = (1,2)$; $s^* = 7$.

La boucle est terminée pour le type i = 1 de barres mères. L'algorithme recommence la boucle 3.2 pour le type de barres mères suivant.

3.2.(1) L'algorithme commence le placement pour le type i = 2 de barres mères.

3.2.1. L'algorithme parcourt toutes les tailles s de paquets en commençant par s = 1. Le placement effectué par l'algorithme de l'arborescence limitée donne : E = (3,1).

3.2.1.3. Le coût du placement des barres filles du vecteur E sur une barre mère de type i = 2 est :

d'après la relation (V.11) $c_b = 1$;

d'après la relation (V.9) $f(1) = 0,995$;

d'après la relation (V.10) $c_p = 0,995.$

3.2.1.4. Ce coût n'améliore pas le meilleur coût pour un paquet c_p^*. Ce placement n'est pas retenu.

...

3.2.1.(1) Les placements locaux ne sont pas retenus jusqu'à une taille s = 6 de paquets.

3.2.1.1. Les nombres recalculés de barres filles à réaliser de chaque type sont : C' = (3,3).

3.2.1.2. Le placement effectué par l'algorithme de l'arborescence limitée donne : E = (3,1).

3.2.1.3. Le coût du placement des barres filles du vecteur E sur une barre mère de type i = 2 est :
d'après la relation (V.11) $c_b = 1$;
d'après la relation (V.9) $f(6) = 0,972$;
d'après la relation (V.10) $c_p = 0,972$.

3.2.1.4. Ce coût n'améliore pas le meilleur coût pour un paquet c_p^*. Ce placement n'est pas retenu.
Pour les autres tailles de paquets, le coût pour un placement local c_b est très supérieur au coût $c_b = 1$. Par conséquent, le coût d'un paquet n'est pas amélioré.
Il n'y a plus de type de barres mères à considérer. L'algorithme a trouvé le meilleur placement local E^* pour le type i^* de barre mère et une taille s^* de paquets.

3.4. L'algorithme remet à jour les variables.
d'après la relation (V.6) $\chi \Leftarrow \chi + s^* \times c_p^*$
d'où $\chi = 0 + 7 \times 0,968 = 6,776$;
d'après la relation (V.7) $C \Leftarrow C - s^* \times E^*$
d'où C = (12,9) ;
d'après la relation (V.8) $B \Leftarrow B - s^* \times K(i^*)$ avec $K(i^*) = (1,0)$
d'où B = (3,10).

On continue le placement de la même façon pour le reste des barres mères et des barres filles. Le paquet retenu est :
$c_p^* = 0,986$; $i^* = 1$; $E^* = (1,2)$; $s^* = 3$; C = (9,3) ; B = (0,10) ; $\chi = 9,734$.

Enfin le dernier paquet est :
$c_p^* = 0,986$; $i^* = 2$; $E^* = (3,1)$; $s^* = 3$.

L'algorithme se termine là car toutes les barres filles commandées ont été réalisées. Le coût final est la somme de tous les coûts des paquets réalisés, c'est-à-dire :

$$\chi = \sum \left(s^* \times c_p^* \right) \qquad (V.12)$$

χ = 12,692.

Le placement final est illustré dans la figure V.7.

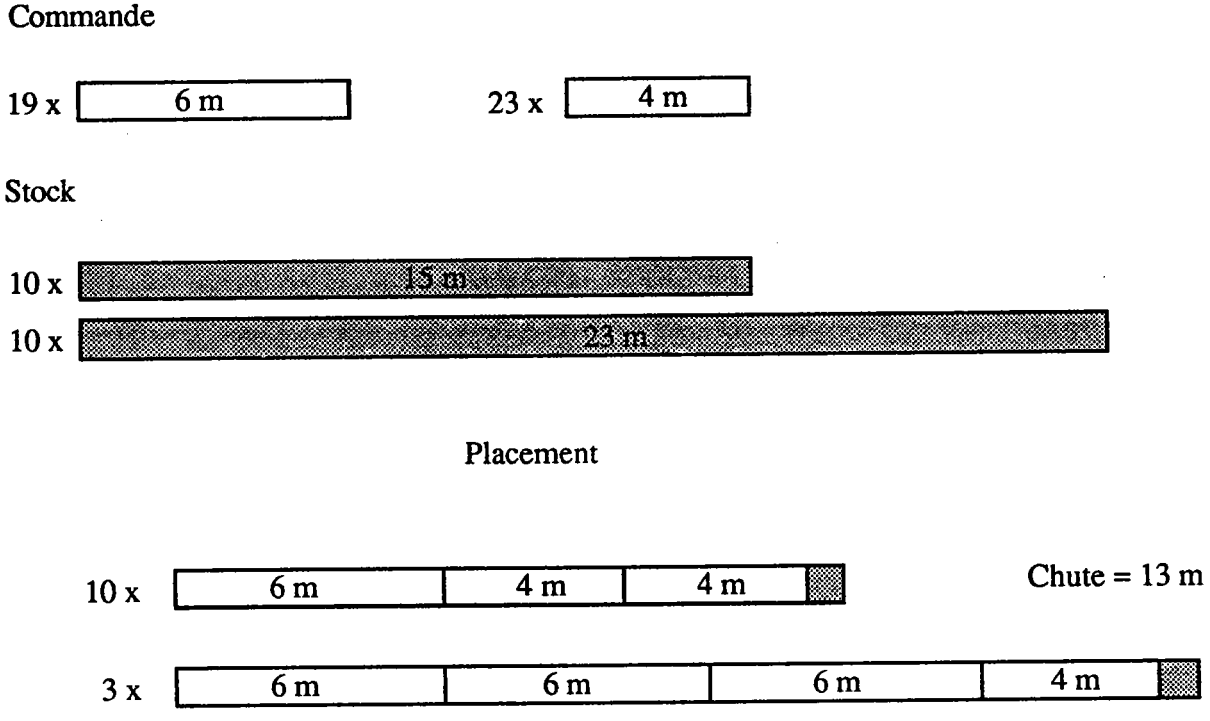

Figure V.7 : Résulat du placement de la méthode locale par paquets

2.5.4. Conclusion relative à la méthode locale par paquets

Cette méthode travaille par type de barres mères contrairement à la méthode suivante qui considère des groupes de paquets. Cette méthode est donc précise dans l'exploration des solutions, mais elle est onéreuse en temps de calcul. Pour diminuer le temps d'exécution, les tailles s de paquets doivent être décomposées en nombres premiers. Par conséquent, un paquet de taille 10 sera réalisé en 2 paquets de 5. Ces paquets seront rassemblés en fin de traitement.

2.6. Méthode globale par paquets

2.6.1. Description de la méthode globale par paquets

La différence avec l'approche précédente est qu'on applique une même taille de paquets pour des barres mères de types différents. On réalise les placements en imposant la taille de paquets. On fait varier cette taille de 1 au maximum possible T_{max}.

La taille maximale d'un paquet est fonction du nombre maximal de barres mères d'un même type ainsi que du nombre maximal de barres filles d'un même type, c'est-à-dire :

$$T_{max} = \min\left(\max_{1 \le i \le N}(M_i), \max_{1 \le j \le n}(m_j) \right) \qquad (V.13)$$

La taille retenue est celle qui minimise un critère fonction de la chute par barre du paquet et du nombre de tombants pondéré par une fonction f(s) décroissante par rapport à la taille du paquet. Cette fonction favorise les gros paquets. Elle est calculée suivant la formule (V.3).

Pour réduire le temps de traitement, dans cette méthode aussi nous nous limitons aux tailles de paquets appartenant à l'ensemble des nombres premiers.

2.6.2. Algorithme global par paquets

La démarche de l'algorithme global par paquets est la suivante :

ALGORITHME GLOBAL PAR PAQUETS

1. Données

- *C_0, vecteur des barres filles candidates dont chaque élément j correspond à la quantité de barres filles de type j à découper, avec j = 1,...,n*
- *B_0, vecteur des barres mères disponibles dont la $i^{ème}$ composante correspond à la quantité de barres mères du type i, avec i = 1,...,N*

2. Initialiser

- *$C = C_0$, vecteur des barres filles candidates dont chaque élément m_j est la quantité de barres filles de même indice restant à découper, avec j = 1,...,n*
- *$B = B_0$, vecteur correspondant aux quantités M_i de barres mères de chaque type encore disponible, avec i = 1,...,N*
- *$\chi = 0$, meilleure valeur du critère pour le placement global*

3. Tant que toutes les barres filles ne sont pas réalisées et qu'il reste des barres mères, faire

3.1. $c_p^* = +\infty$ *(c_p^* contiendra la meilleure valeur du critère pour un groupe de paquets)*

3.2. Calculer T_{max}

*3.3. Pour toute **taille s de paquets** de barres de stock, s variant de 1 à T_{max} et s appartient à l'ensemble des nombres premiers*

3.3.1. Construire un problème avec une commande fictive $C_1 = \left\lfloor \frac{C}{s} \right\rfloor$

et un stock fictif $B_1 = \left\lfloor \frac{B}{s} \right\rfloor$

3.3.2. Résoudre ce problème par l'algorithme FFD étendu ou l'algorithme de l'arborescence limitée étendu. Les barres mères utilisées sont stockées dans le vecteur B' et les barres filles réalisées sont stockées dans le vecteur C'. Le placement de chaque type de barres mères découpées de chacun des k paquets est stocké dans E. $E_{k'}$ est le vecteur des barres filles réalisées dans une barre mère du $k^{ième}$ paquet ($E = \{E_1, ..., E_k\}$)

3.3.3. Calculer le coût de ce placement correspondant à une taille s de paquets :

$$c_b' = \sum_{k'=1}^{k} (c_b(k') \times s_{k'}) \qquad (V.14)$$

($c_b(k')$ est le coût pour la découpe d'une barre mère du $k'^{ième}$ paquet et $s_{k'}$ est le nombre de fois que le placement $E_{k'}$ est reproduit)

$$c_p = c_b' \times f(s) \qquad (V.15)$$

($f(s)$ est une fonction décroissante de la taille s de paquets)

3.3.4. Si $c_p < c_p^*$ alors

$$c_p^* = c_p \; ; \; B^* = B' \; ; \; C^* = C' \; ; \; E^* = E \; ; \; s^* = s$$

3.4. Fin boucle pour

3.5. Mettre à jour les variables

$$\chi \Leftarrow \chi + c_p^* \times s^*$$

$$B \Leftarrow B - B^* \times s^*$$

$$C \Leftarrow C - C^* \times s^*$$

4. Fin tant que.

2.6.3. Exemple d'application de la méthode globale par paquets

Par comparaison avec l'algorithme local par paquets, nous reprenons les mêmes données.

L'algorithme de l'arborescence limitée **étendu** est choisi pour résoudre les placements locaux (cf. section 2.4).

1. Les données de départ sont :
   - $n_S = 2$; $\gamma = 0{,}01$;
   - $C_0 = (19,23)$, avec :
     $l_1 = 6$; $\tau_1 = 0$; $m_1 = 19$;
     $l_2 = 4$; $\tau_2 = 0$; $m_2 = 23$;
   - $B_0 = (10,10)$, avec :
     $L_1 = 15$; $T_1 = 0$; $M_1 = 10$;
     $L_2 = 23$; $T_2 = 0$; $M_2 = 10$.
2. Les variables sont initialisées comme suit :
   $C = C_0 = (19,23)$; $B = B_0 = (10,10)$; $\chi = 0$.
3. L'algorithme commence le placement des barres filles du vecteur C sur des barres mères du vecteur **B**.
   3.1. Le meilleur coût c_p^* est initialisé à $+\infty$.
   3.2. La taille maximale d'un paquet est : $T_{max} = 10$.
   3.3. L'algorithme parcourt toutes les tailles s de paquets en commençant par s = 1.

3.3.1. Comme la taille de paquets est s = 1, il suffit de considérer le stock initial et la commande initiale : $C_1 = C$ et $B_1 = B$.

3.3.2. Soient k le nombre de paquets et k' l'indice de chacun des paquets. Le placement par l'algorithme de l'arborescence limitée étendu est le suivant :

$k = 4$;

$k' = 1$; $E_1 = (1,2)$; $i_1 = 1$; $s'_1 = 5$; $\lambda_1 = 1$;

$k' = 2$; $E_2 = (3,1)$; $i_2 = 2$; $s'_2 = 3$; $\lambda_2 = 1$;

$k' = 3$; $E_3 = (2,0)$; $i_3 = 1$; $s'_3 = 2$; $\lambda_3 = 3$;

$k' = 4$; $E_4 = (1,0)$; $i_4 = 1$; $s'_4 = 1$; $\lambda_4 = 9$;

où les indices correspondent à chacun des paquets réalisés :

- $\mathbf{E_{k'}}$ est le vecteur des barres filles réalisées dans une barre mère du k'ième paquet dont les composantes sont les nombres de barres de chaque type ;
- $\mathbf{i_{k'}}$ est l'indice du type de barres mères découpées ;
- $\mathbf{s'_{k'}}$ est le nombre de répétitions du placement $\mathbf{E_{k'}}$. Il est obtenu par regroupement de toutes les barres mères ayant la même découpe ;
- $\mathbf{\lambda_{k'}}$ est la longueur de la chute dans une barre mère découpée du paquet **k'**. Pour simplifier l'exemple, nous n'autorisons pas la génération de tombants.

Ces placements locaux $\mathbf{E_{k'}}$ de barres filles sur un type de barres mères sont regroupés dans **E**.

Le vecteur des barres filles réalisées et le vecteur des barres mères utilisées sont :

$$C' = \sum_{k'=1}^{k} \left(E_{k'} \times s_{k'}\right) = (19,23) ;$$

$$B' = \sum_{k'=1}^{k} \left(K(i_{k'}) \times s_{k'}\right) = (8,3).$$

3.3.3. Pour ce placement, le coût $\mathbf{c_p}$ pour cette taille de paquets est la somme des coûts des paquets pondérée par la fonction f(s) favorisant la génération de paquets :

d'après la relation (V.14) $c'_b = \sum_{k'=1}^{k} \left(c_b(k') \times s_{k'}\right)$

$= 5\times1 + 3\times1 + 2\times3 + 1\times9 = 23$;

d'après la relation (V.9) $f(1) = 0,995$;

d'après la relation (V.15) $c_p = 22,885$.

3.3.4. Ce coût est retenu comme le meilleur coût pour cette taille s de paquets :

$c_p^* = 22,885$; $B^* = (8,3)$; $C^* = (19,23)$; $E^* = E$; $s^* = 1$.

3.3.(4) Passons directement au nombre premier s = 5 (les autres étant calculés de la même manière et n'améliorent pas c_p^*).

3.3.1. L'algorithme divise le nombre de barres mères de chaque type et le nombre de barres filles de chaque type par s = 5 :

$B_1 = (2,2)$; $C_1 = (3,4)$.

3.3.2. L'algorithme de l'arborescence limitée étendu donne le placement suivant :

$E_1 = (1,2)$; $i_1 = 2$; $s_1' = 1$; $\lambda_1 = 1$;

$E_2 = (2,0)$; $i_2 = 1$; $s_2' = 1$; $\lambda_2 = 3$.

3.3.3. Le coût c_p pour ce placement est par conséquent :

d'après la relation (V.14) $c_b' = 1\times1 + 1\times3 = 4$;

d'après la relation (V.9) $f(5) = 0,991$;

d'après la relation (V.15) $c_p = 3,964$.

3.3.4. Le coût de ce placement est inférieur au meilleur coût trouvé. Par conséquent, il est retenu comme le meilleur coût pour cette taille de paquets :

$c_p^* = 3,964$; $B^* = (1,1)$; $C^* = (3,4)$; $E^* = E$; $s^* = 5$.

Pour la taille de paquets s=7, le placement local trouvé est :

$E_1 = (1,2)$; $i_1 = 2$; $s_1' = 1$; $\lambda_1 = 1$;

$E_2 = (1,1)$; $i_2 = 1$; $s_2' = 1$; $\lambda_2 = 13$.

Le coût du placement est très supérieur au coût $c_p^* = 3,964$. Par conséquent, le coût d'un paquet n'est pas amélioré.

Toutes les tailles de paquets sont explorées. L'algorithme retient le meilleur placement trouvé E^*, le vecteur B^* des barres mères utilisées et le vecteur C^* des barres filles réalisées.

3.5. L'algorithme remet à jour les variables :

$\chi = 3,964$; $B = (5,5)$; $C = (4,3)$.

3.(1) L'algorithme continue le placement de la même façon pour le reste des barres filles à réaliser. Le paquet retenu est :

$E_1 = (3,1)$; $i_1 = 2$; $s_1' = 1$; $\lambda_1 = 1$;

$c_p^* = 0,997$; $E^* = (3,1)$; $s^* = 1$;

et $B = (4,5)$; $C = (1,2)$.

Le dernier placement retenu est :

$E_1 = (1,2)$; $i_1 = 1$; $s_1' = 1$; $\lambda_1 = 1$;

$c_p^* = 0,997$; $E^* = (1,2)$; $s^* = 1$;

et $B = (4,4)$; $C = (0,0)$.

L'algorithme se termine là car toutes les barres commandées ont été réalisées. Le coût final est la somme de tous les coûts des paquets réalisés, c'est-à-dire :

$$\chi = \sum \left(s^* \times c_p^* \right) ;$$

$\chi = 5 \times 3{,}964 + 0{,}997 + 0{,}997 = 21{,}814.$

Cette solution est différente de celle de la méthode précédente. Pour ce cas-ci, elle est presque deux fois moins bonne d'après la fonction coût considérée (les coûts de coupe n'étant pas connus). Le placement final est illustré dans la figure V.8.

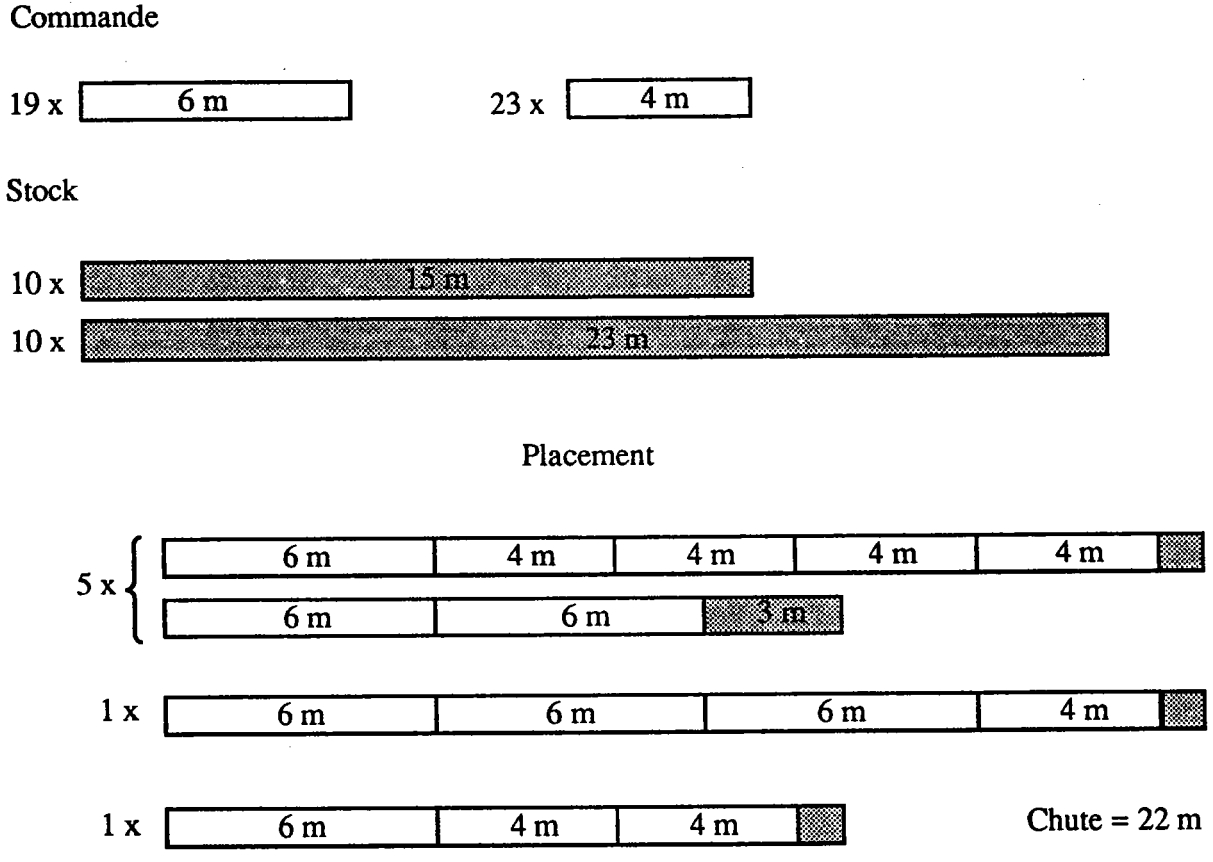

Figure V.8 : Résulat du placement de la méthode globale par paquets

2.6.4. Conclusion relative à la méthode globale par paquets

Comme pour la méthode locale par paquets, l'utilité de cette méthode globale par paquets est sensible lorsque les commandes et les stocks sont de tailles importantes. Les paquets ainsi réalisés évitent aux scieurs les erreurs dues aux réglages trop fréquents des longueurs à réaliser.

En plaçant les barres filles à découper paquet par parquet, les algorithmes par parquets permet d'éviter certains optima locaux dus aux choix des placements locaux initiaux.

Considérons sur un exemple ce cas de figure :

Commandes :

- 2 barres de longueur 6 m à tolérance normale ;

- 2 barres de longueur 5 m à tolérance normale.

Stocks :

- 3 barres de longueur 12,1 m ;
- 2 barres de longueur 11,2 m.

Le cheminement que nous décrivons est vrai pour les méthodes ne considérant pas le problème des paquets. Le programme recherche à chaque étape la barre mère dans laquelle il va générer le moins de chute possible :

- dans la barre mère de 12,1 m, il peut découper deux barres filles de 6 m en réalisant 2 traits de scie (0,01 m par coupe). La chute alors générée lors de cette découpe est de **0,1 m**.
- dans la barre mère de 11,2 m, il va découper une barre fille de 6 m et une de 5 m et réaliser 2 traits de scie ; ce qui nous donne pour ce placement local, une chute de **0,2 m**.

Les deux choix de découpe sont représentés dans la figure V.9.

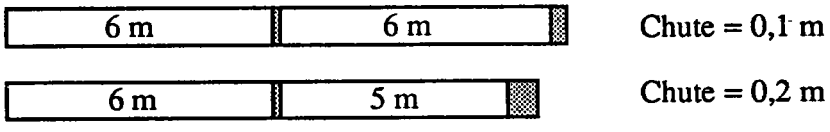

Figure V.9 : Choix pour le premier placement

Le programme compare les chutes générées de ces deux solutions partielles. La chute générée correspondant au premier placement est inférieure à celle générée par le second, aussi la première solution partielle est retenue.

De même à l'étape suivante, le programme devra choisir entre deux découpes pour obtenir deux barres filles de 5 m :

- découper la barre mère de 12,1 m où la chute générée est de **2,1 m**.
- découper la barre mère de 11,2 m et générer une chute de **1,2 m**.

Les deux choix de découpe sont représentés dans la figure V.10.

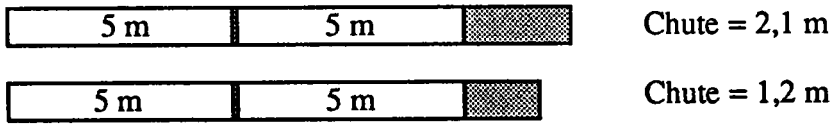

Figure V.10 : Choix pour le second placement

Il va choisir le placement sur la barre de 11,2 m où la chute générée par ce nouveau placement est la plus faible. En conséquence, la chute finale pour le placement proposé par le programme est la somme des chutes générées dans les deux barres mères découpées, c'est-à-dire **1,3 m**. Ce placement est représenté dans la figure V.11.

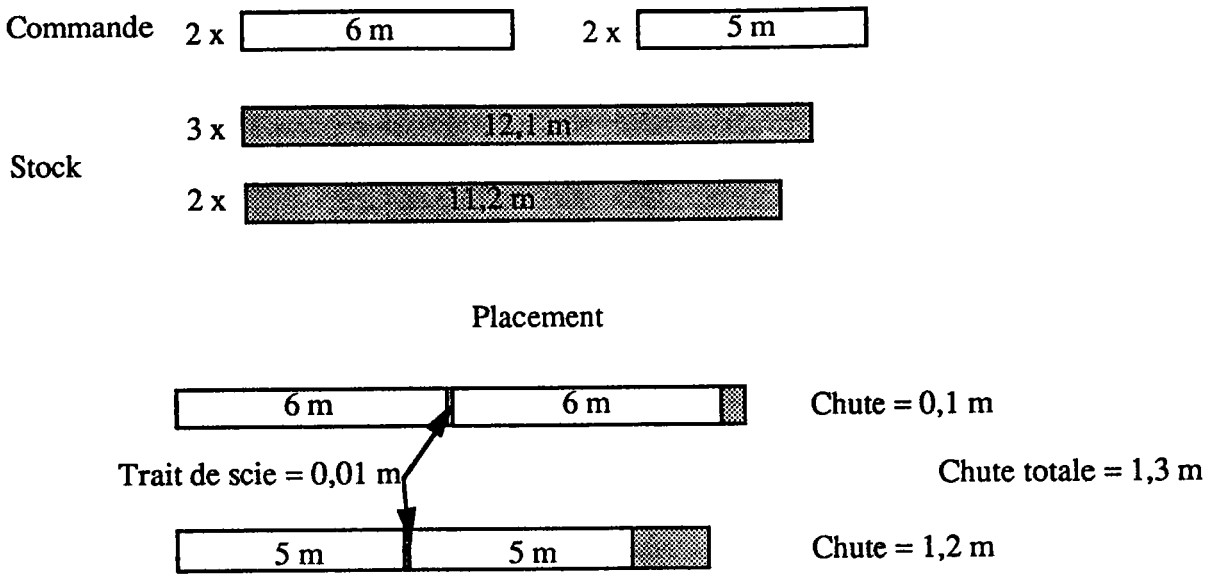

Figure V.11 : Résulat du placement pour une méthode sans paquets

Regardons maintenant la découpe pour une des méthodes par paquets. La démarche est identique lorsque le paquet est constitué d'une seule barre mère. Pour la deuxième itération, le nombre de barres mères dans le paquet est de deux. Les données initiales sont divisées par deux, il ne reste plus qu'une barre de chaque type (commande ou stock). L'algorithme a deux possibilités de découper les barres filles de 5 et 6 m :

- dans la barre mère de 12,1 m où la chute générée est de **1,1 m**.
- dans la barre mère de 11,2 m et générer une chute de **0,2 m**.

Les deux choix de découpe sont représentés dans la figure V.12.

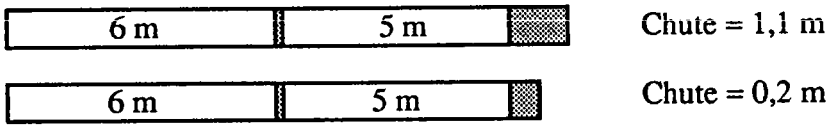

Figure V.12 : Choix pour le placement en tenant compte des paquets

En pondérant le coût des placements locaux par la fonction f(s), l'algorithme choisira la seconde option qui consiste à réaliser un paquet de deux barres mères de longueur 11,2 et à y découper les quatre barres filles (5 et 6 m dans chaque barre mère). Il n'y a plus de barres filles à réaliser, la chute finale est donc **0,4 m**. Par conséquent, ce placement est meilleur que le précédent aussi bien au niveau chute qu'au niveau paquet. Le placement est illustré par la figure V.13.

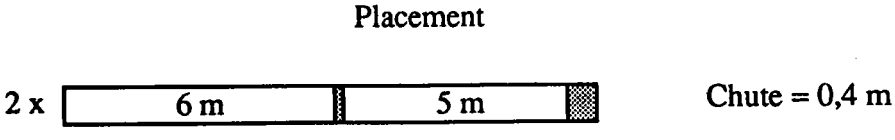

Figure V.13 : Résulat du placement pour une méthode avec paquets

Cependant, les résultats obtenus par les deux algorithmes privilégiant la réalisation des paquets sont variables suivant les données du problème.

2.7. Expériences numériques

A la demande des utilisateurs du logiciel et afin d'éviter de produire des tombants de trop petites longueurs, les barres mères donnant un tombant à la fin de la découpe sont échangées avec des barres plus longues restant en stock. Ceci a pour objectif d'augmenter les possibilités de réutiliser ces tombants dans le futur.

2.7.1. Comparaison des résultats du programme avec la mise en barres réalisée par notre partenaire

Les placements sont réalisés à partir de l'état du stock et des commandes au moment où l'agent commercial décide de réaliser un placement. Les données utilisées par le logociel sont identiques à ce que connaît l'agent commercial au moment de sa préparation de la fiche de travail. Les solutions obtenues sont des ***solutions globales***, c'est-à-dire que toutes les commandes réalisées dans la même journée sont lancées avec les mêmes paramètres. Les comparaisons des résultats du logiciel avec ceux réalisés manuellement par notre partenaire industriel sont récapitulées dans le tableau V.1.

Fichiers	Meilleurs	Égaux	Moins bons	Gain de chutes (%)	Gain de tombants (%)	Tps moy / affaire (sec.)
1	35,80 %	61,73 %	2,47 %	54,27	9,43	3
2	7,74 %	87,45 %	4,81 %	2,11	5,28	2
3	11,03 %	87,50 %	1,47 %	0,01	16,67	1
4	2,06 %	97,94 %	0 %	0,08	21,67	1
Moyenne	10,56 %	86,31 %	3,13 %	6,43	10,72	2

Tableau V.1 : Tableau de comparaisons entre les placements de notre partenaire et ceux des algorithmes développés

La colonne ***"Fichiers"*** correspond aux noms des placements exécutés dans la journée. Dans une journée en moyenne, 100 affaires sont traitées correspondant à 650 barres filles réalisées et 350 barres mères découpées.

Les colonnes ***"Meilleurs"***, ***"Égaux"*** et ***"Moins bons"*** donnent la proportion des affaires dont le placement préconisé par le logiciel est respectivement meilleur, égal ou moins bon du point de vue des critères retenus que celui de notre partenaire industriel par rapport à l'ensemble des affaires traitées.

D'une référence (qualité, profil et dimensions) à l'autre, les coûts des chutes ne sont pas comparables. Par conséquent, notre partenaire industriel a choisi de comparer le poids des chutes induites par les placements. Comme le poids des chutes générées est étroitement lié à la génération de tombants, nous y avons intégré les coûts de ces derniers (les coûts sont ramenés en coûts unitaires de chutes). La colonne ***"Gain de chute"*** donne la proportion de réduction des chutes (en poids) des solutions données par le logiciel par rapport au poids des chutes des solutions de notre partenaire. La colonne ***"Gain de tombants"*** correspond à la proportion du nombre de tombants générés en moins par les placements du logiciel par rapport aux placements de notre partenaire.

Enfin, la colonne ***"Tps moy / affaire"*** donne le temps moyen d'exécution par le logiciel pour une affaire.

Nous voyons que, dans seulement 3,13 % des cas, les placements réalisés par le logiciel sont moins bons que ceux réalisés manuellement par notre partenaire. Cela est dû au fait que l'agent commercial a une vue globale du problème à contrario du programme qui "optimise" localement.

Les gains substantiels, sur la chute, sur la génération et l'utilisation des tombants, apportés aux placements du fichier 1 méritent une analyse succincte. Nous pouvons distinguer trois causes principales à ces gains :

1. Depuis peu, notre partenaire industriel a pu ***estimer le coût de la génération d'un tombant*** par rapport au coût de la matière. Le paramètre α utilisé dans le coût de la génération de tombant (cf. chapitre III, section 3.3) est fixé. Ainsi, il nous est facile de choisir pour chacun des placements locaux (sur une barre) la génération soit d'un tombant, soit d'une chute. Par contre, l'agent commercial ne calcule pas exactement la longueur du bout, dont la génération d'un tombant est plus intéressante que celle d'une chute; il estime approximativement cette limite suivant son expérience.
2. Lorsque la ***taille du problème*** est importante (en nombre de barres en stock et en nombre de barres commandées), il est difficile d'optimiser la solution manuellement surtout si, en plus, il y a une grande diversité de types de barres mères et de types de barres filles. En effet, l'agent commercial ne peut avoir la même vue globale que celle qu'il a pour les problèmes de petites tailles. Par conséquent, il ne parcourt qu'un nombre limité de solutions contrairement au programme de découpe qui lui n'est limité que par le temps.
3. Les agents commerciaux connaissent parfaitement les longueurs standards disponibles en stock. Ils ont "l'***habitude***" de travailler avec ce disponible sûr. Le programme de découpe n'a pas d'a priori. Il parcourt absolument tout le stock disponible et bien souvent, la découpe d'un tombant apporte de sérieuses améliorations au placement.

Dans le placement du fichier 1, le cumul de tous les détails cités, auxquels il faut ajouter les nombreuses utilisations de tombants (le coût d'un tombant est connu), apporte de très importantes améliorations. Nous donnons maintenant le détail des résultats pour le placement du fichier 1.

2.7.2. Résultats détaillés pour le fichier 1

nombre d'affaires : 92

nombre total de barres commandées : 624

nombre total de barres en stock : 327

	% de chute (poids)	Nombre de tombants	Gain en tonnes	Temps total
Solution partenaire	2,53 %	58		
Solution globale heuristique FFD méthode globale par paquets comparaison sur les chutes	1,64 %	55	3,072	3 mn 48 s
Aide à la décision	1,56 %	58	3,348	
Planification 1 heuristique FFD méthode locale par paquets comparaison sur le rendement pénalisation tombant = 0,077	1,58 %	38	3,279	3 mn 47s
Planification 2 heuristique FFD méthode locale par paquets comparaison sur le rendement pénalisation tombant = 0,001	1,47 %	56	3,659	3 mn 36 s

Tableau V.2 : Tableau récapitulatif des placements de notre partenaire et ceux des algorithmes développés

Dans le tableau V.2, nous donnons en première ligne, ***solution partenaire***, les solutions choisies par notre partenaire. Le temps total passé par les agents commerciaux pour réaliser l'ensemble de ces placements n'est pas connu. Cependant, il faut savoir que, pour certaines de ces affaires, un agent commercial averti peut passer jusqu'à 2 heures pour réaliser une affaire.

La deuxième ligne, ***solution globale***, est obtenue en lançant simplement le programme de découpe pour toutes les affaires. Il n'y a pas d'intervention locale pour changer les paramètres ou les algorithmes et ainsi améliorer les solutions. Rappelons qu'un algorithme ne donne pas la meilleure solution dans tous les cas. L'algorithme retenu utilise la méthode globale par paquets à l'intérieur de laquelle l'heuristique FFD est utilisée avec une comparaison des coûts des placements en terme de coûts de chutes.

L'***aide à la décision*** consiste à lancer le programme pour chaque affaire avec des paramètres et des algorithmes différents, et à garder la meilleure solution. Par conséquent, cette méthode nécessite l'intervention d'un agent commercial. Du point de vue de la qualité des

placements trouvés, elle est surclassée par la qualité des placements, ci-après, regroupant toutes les commandes de la journée.

Pour la ***planification***, toutes les commandes d'une journée sont regroupées. Par conséquent, des commandes correspondant à des affaires différentes, mais de même référence, peuvent être réalisées sur la même barre de stock. Les solutions que nous proposons sont obtenues en utilisant des paramètres globaux, ce qui signifie que nous ne faisons varier ni les paramètres, ni les algorithmes pour améliorer les placements de certaines références. Afin de mieux percevoir l'impact de la pénalisation de la génération de tombants nous considérons deux valeurs du paramètre β dans le coût de la génération de tombants du chapitre III, section 3.3.

Dans le premier cas, ***planification 1***, la génération de tombants est fortement pénalisée. En effet, en plus des coûts de transport et de stockage (α), nous ajoutons, pour 1 m de tombant, le coût de 77 mm de chute. Le nombre de tombants générés a ainsi chuté de 58 à 38 (35 %) sans trop de répercussion sur le pourcentage de chute global.

Dans le second cas, ***planification 2***, la génération de tombants est très légèrement pénalisée. Nous ajoutons, pour 1 m de tombant, le coût de 1 mm de chute afin de faire la distinction entre la génération d'un tombant et d'une chute. Le nombre de tombants générés n'est pas très important par rapport au nombre de références traitées, mais le pourcentage de chute global est plus intéressant par rapport au poids total de commandes à réaliser. Le gain en poids de chutes est de 380 kg par rapport à la planification 1 et de 587 kg par rapport à la solution de notre partenaire industriel.

2.8. Conclusion relative aux méthodes par construction

Le problème étant combinatoire, les solutions trouvées par les heuristiques de placement ne sont pas optimales. Chacune des heuristiques présentées explore à sa façon le domaine des solutions réalisables. Elles sont complémentaires au point de vue des solutions proposées. Aussi, une heuristique ne donnera pas systématiquement une meilleure solution que les autres. Seule une exploitation approfondie de tous les paramètres disponibles peut procurer la meilleure des algorithmes de découpe implémentés. Ceci n'est généralement pas nécessaire car les solutions réalisables obtenues par les heuristiques de découpe sont de bonne qualité.

Cependant, il faut remarquer qu'il existe des cas où ces heuristiques présentent des solutions moins bonnes que celles réalisées manuellement. En effet, ces heuristiques cherchent à optimiser, à chaque étape, le placement pour chaque barre présente en stock. Ce placement ne prend pas en compte l'ensemble des barres du stock.

Voyons sur un exemple simple que le meilleur placement local ne donne pas toujours le meilleur placement global :

commande : 2 barres de longueur 5 m à tolérance réduite ;

stock : 3 barres respectivement de longueur 5,1, 6,1 et 10,2 m.

Le cheminement que nous décrivons est vrai pour toutes les méthodes par construction. Le programme recherche à chaque étape la barre mère dans laquelle il va générer le moins de chute possible :

- dans la barre mère de 5,1 m, il peut découper la barre fille de 5 m en réalisant 2 affranchissements (0,04 m par affranchissement) et 2 traits de scie (0,01 m par coupe). La chute générée lors de cette découpe est de **0,1 m** ;
- dans la barre mère de 10,2 m, il va découper les 2 barres filles de 5 m et réaliser 2 affranchissements et 3 traits de scie ; ce qui nous donne pour ce placement local, une chute de **0,2 m**.

Remarque : le rendement est le même pour les deux placements.

Les deux choix de découpe sont représentés dans la figure V.14.

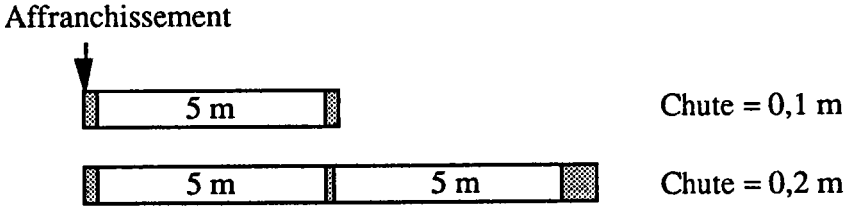

Figure V.14 : Choix pour le premier placement

Le programme compare les chutes générées de ces deux solutions partielles. La chute générée par le premier placement est inférieure au second, aussi la première solution partielle est retenue.

A l'étape suivante, puisqu'il n'y a plus de barres de 5,1 m en stock, le programme devra choisir entre :

- découper la barre mère de 6,1 m où la chute générée est de **1,1 m** ;
- découper la barre mère de 10,2 m et générer une chute de **5,2 m**.

Les choix pour le second placement sont représentés dans la figure V.15.

Figure V.15 : Choix pour le second placement

L'algorithme va choisir le placement sur la barre de 6,1 m, placement pour lequel la chute générée est la plus faible. En conséquence, la chute finale pour le placement proposé par le programme est la somme des chutes générées dans les deux barres mères découpées, c'est-à-dire **1,2 m**. Ce placement est représenté dans la figure V.16.

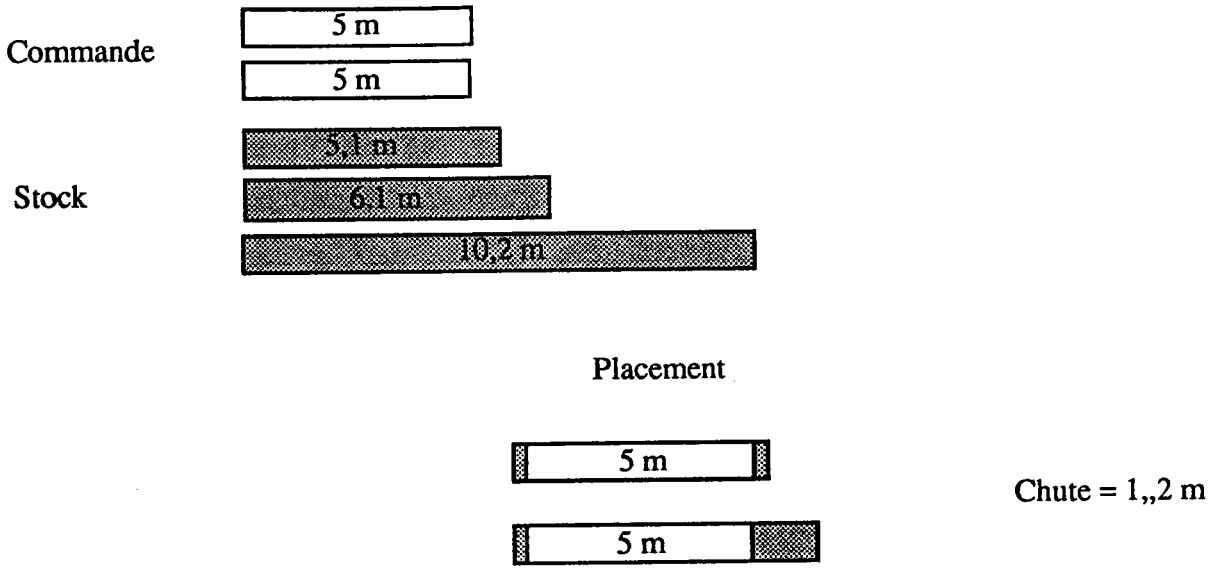

Figure V.16 : Résulat du placement pour une méthode de placement par construction

Manuellement, en considérant toutes les barres mères, la solution optimale serait de placer, dès le départ, les 2 barres commandées sur la barre mère de 10,2 m pour avoir une chute finale de **0,2 m**. Ce placement est représenté dans la figure V.17.

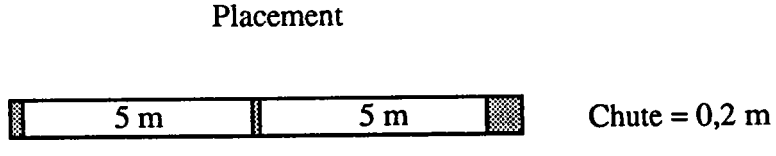

Figure V.17 : Placement optimal

Nous voyons dans cet exemple que pour être dans un tel cas, il faut réunir 3 conditions :

- les longueurs des barres filles sont proches,
- les longueurs des barres mères sont proches des multiples des longueurs des barres filles,
- et la quantité de stock disponible est très faible.

Ce cas de figure se présente évidemment très rarement dans la réalité du simple fait que les commandes et les stocks sont importants en quantité et en diversité. Il fallait cependant le signaler et en tenir compte. Une des conséquences de cette remarque est que le programme sera nécessairement interactif pour permettre aux agents commerciaux de modifier et/ou de relancer le programme sur d'autres bases.

Dans cette section, nous avons vu deux méthodes de comparaison entre placements locaux, deux méthodes de placements locaux et trois méthodes de résolution du problème global (la méthode sans paquet, la méthode locale par paquet, la méthode globale par paquet). En conséquence, le nombre de combinaisons des méthodes est de 2×2×3 = 12.

CHAPITRE V

3. MÉTHODES INSPIRÉES DE LA PROGRAMMATION DYNAMIQUE

3.1. Introduction

Les algorithmes par construction développés dans la section 2 optimisent une fonction intégrant plusieurs paramètres variables. Les critères n'étant pas comparables entre-eux, la solution choisie consiste à hiérarchiser ces critères qualitatifs et à utiliser des algorithmes qui utilisent des règles de priorité pour en tenir compte.

Un deuxième problème de découpe est posé par une autre branche de notre partenaire industriel. Il s'agit là aussi de découper des produits longs. Ce problème diffère du précédent par le fait que tous les critères utilisés (cf. chapitre III, section 3.3) ont pu être exprimés à l'aide du même paramètre. Nous avons donc à résoudre ici un problème **mono-critère.**

Dans cette section, nous introduisons une nouvelle notion : les tailles fictives de paquets. Nous montrons ensuite que le problème peut être résolu de manière optimale par la méthode de programmation dynamique. Malheureusement, cette méthode est de complexité exponentielle, comme nous le montrons plus tard. Nous décrivons ensuite cinq méthodes [ANT 2] que la programmation dynamique nous a inspirées. Enfin, nous comparons les résultats obtenus par ces nouveaux algorithmes aux placements réels de notre partenaire.

3.2. Tailles fictives de paquets

3.2.1. Description de la méthode de calcul des coûts des paquets fictifs

A cause des tailles de paquets non réalisables, il est approprié de définir des "paquets fictifs". Un paquet fictif est conceptuellement considéré comme un seul paquet mais physiquement il est composé de différents paquets ayant des tailles réalisables. Soit $\mu(s,\sigma)$ la somme des coûts de coupe et des coûts de préparation pour un paquet (réel ou fictif) ayant une taille s nécessitant un nombre σ de coupes. Par définition, pour chaque taille réalisable de paquets, nous avons :

$$\mu(s,\sigma) = \pi + \sigma \times p(s) \qquad (V.16)$$

où :

- π est le coût du placement d'un paquet sur la scieuse (appelé aussi coût de préparation) ;
- **p(s)** est le coût d'une coupe dans un paquet de s barres mères.

Pour chaque taille s de paquets non-réalisables, nous considérons un paquet fictif. Il y a différentes façons de décomposer ce paquet fictif en paquets de tailles réalisables. Naturellement, nous nous intéressons uniquement à la décomposition donnant le coût le plus faible. Par conséquent, pour un paquet de taille s non-réalisable, nous avons :

$$\mu(s,\sigma) = \min_{1 \le s' \le \lfloor s/2 \rfloor} \{\mu(s-s',\sigma) + \mu(s',\sigma)\} \qquad (V.17)$$

où $\mu(s\text{-}s',\sigma)$ et $\mu(s',\sigma)$ sont les coûts d'installation et de coupe, réels ou fictifs, déjà calculés (V.16).

Ainsi, les coûts $\mu(s,\sigma)$ des paquets fictifs peuvent être calculés en utilisant une approche de type "programmation dynamique" où les relations (V.16) et (V.17) sont utilisées respectivement comme conditions limites et formules de récurrence.

3.2.2. Algorithme du calcul des coûts des paquets fictifs

L'algorithme s'écrit de la manière suivante :

ALGORITHME DU CALCUL DES COÛTS DES PAQUETS FICTIFS

1. Données
- *π, coût de préparation d'un paquet*
- *S, taille maximale d'un paquet (où S est le nombre le plus grand de barres mères identiques)*
- *p(s), coût d'une coupe dans un paquet de taille S*
- *$\overline{\sigma}$, nombre maximal de coupes dans une barre mère*

2. Pour un nombre σ de coupes, σ variant de 1 à $\overline{\sigma}$

2.1. Pour toute taille s de paquets, s variant de 1 à S

2.1.1. Si s est une taille de paquets réalisable

Calculer le coût : $\mu(s,\sigma) = \pi + \sigma \times p(s)$

2.1.2. Sinon

2.1.2.1. Initialiser le coût $\mu(s,\sigma)$ à $+\infty$

2.1.2.2. Décomposer le paquet en deux paquets de taille s' et (s-s') avec s' variant de 1 à $\lfloor s/2 \rfloor$

2.1.2.2.1. Si $(\mu(s',\sigma) + \mu(s\text{-}s',\sigma)) < \mu(s,\sigma)$ alors

$\mu(s,\sigma) = (\mu(s',\sigma) + \mu(s\text{-}s',\sigma))$

2.1.2.3. Fin décomposer

2.1.3. Fin si

2.2. Fin pour

3. Fin pour .

3.2.3. Exemple d'application de l'algorithme du calcul des coûts des paquets fictifs

Les données sont :

Coût de préparation (m) : 0,01 (en mètres de chute)

Les coûts d'une coupe suivant la taille du paquet sont donnés dans le tableau V.3.

tailles de paquets réalisables :	1	2	4	6
coûts pour une coupe p(s) :	0,005	0,006	0,008	0,01

Tableau V.3 : Coûts de coupe en fonction du nombre s de barres mères découpées

La taille de paquets la plus grande est 7 (barres mères).

Le nombre le plus grand de coupes dans une barre mère est 3.

1. Les données initiales sont :

   $\pi = 0,01$;

   $S = 7$;

   les coûts p(s) sont donnés dans le tableau V.3, les coûts des tailles de paquets non réalisables sont mis à $+\infty$;

   $\bar{\sigma} = 3$.

2. L'algorithme balaie tous les nombres σ de 1 à $\bar{\sigma}$ en commençant par $\sigma = 1$.

   2.1. L'algorithme parcourt les tailles s de paquets de 1 à S en commençant par s = 1.

      2.1.1. Il s'agit d'une taille réalisable. L'algorithme calcule le coût $\mu(s,\sigma)$ d'après (V.16) $\mu(1,1) = 0,01 + 1\times0,005 = 0,015$.

   2.1.(1) La taille de paquets suivante est s = 2.

      2.1.1. C'est une taille réalisable. L'algorithme calcule le coût $\mu(s,\sigma)$.

      D'après (V.16), $\mu(2,1) = 0,01 + 1\times0,006 = 0,016$.

   2.1.(2) L'algorithme continue avec la taille de paquets suivante : s = 3. Cette taille n'est pas réalisable. L'algorithme passe à l'étape 2.1.2.

      2.1.2. Cette taille n'est pas réalisable. L'algorithme continue aux niveaux inférieurs de l'étape 2.1.2.

         2.1.2.1. L'algorithme initialise $\mu(3,1)$ à $+\infty$.

         2.1.2.2. Pour un nombre $\sigma = 1$ de coupes, l'algorithme décompose le paquet en deux paquets de taille s' et s-s', avec s' variant de 1 à $\lfloor s/2 \rfloor = 1$.

         Pour une décomposition en taille de paquets s' = 1 et s-s' = 2, le coût est :

         $(\mu(1,1) + \mu(2,1)) = 0,015 + 0,016 = 0,031$.

      2.1.2.2.1. Comme $(\mu(1,1) + \mu(2,1)) < \mu(3,1)$, alors

      $\mu(3,1) = 0,031$.

      C'est la seule décomposition possible et elle est donc la meilleure.

2.1.(3) La taille de paquets suivante est s = 4.

2.1.1. La taille 4 est réalisable, l'algorithme calcule le coût $\mu(s,\sigma)$.

D'après (V.16), $\mu(4,1) = 0,01 + 1\times0,008 = 0,018$.

2.1.(4) L'algorithme passe à la taille de paquets s = 5.

2.1.2. L'algorithme continue aux niveaux inférieurs de l'étape car cette taille n'est pas réalisable.

2.1.2.1. L'algorithme initialise $\mu(5,1)$ à $+\infty$.

2.1.2.2. Pour un nombre $\sigma = 1$ de coupes, l'algorithme décompose le paquet en deux paquets de taille s' et s-s', avec s' variant de 1 à $\lfloor s/2 \rfloor = 2$.

Pour une décomposition en taille de paquets s' = 1 et s-s' = 4, le coût est :

$(\mu(1,1) + \mu(4,1)) = 0,015 + 0,018 = 0,033$.

2.1.2.2.1. Comme $(\mu(1,1) + \mu(4,1)) < \mu(5,1)$, alors $\mu(5,1) = 0,033$.

2.1.2.2(1) Pour une décomposition en taille de paquets s' = 2 et s-s' = 3, le coût est :

$(\mu(2,1) + \mu(3,1)) = 0,016 + 0,031 = 0,047$.

2.1.2.2.1. La relation $(\mu(1,1) + \mu(4,1)) < \mu(5,1)$ n'est pas vérifiée, aussi cette décomposition n'est pas retenue.

Il n'y a pas d'autre décomposition. La première est donc la meilleure.

Le reste des valeurs de $\mu(s,\sigma)$ est obtenu de la même façon. Les valeurs sont récapitulées dans le tableau V.4 :

		s						
		1	2	3	4	5	6	7
σ	1	0,015	0,016	0,031	0,018	0,033	0,02	0,053
	2	0,020	0,022	0,042	0,026	0,046	0,03	0,076
	3	0,025	0,028	0,053	0,034	0,059	0,04	0,099

Tableau V.4 : Valeurs de μ suivant s et σ

3.2.4. Conclusion concernant l'algorithme du calcul des coûts des paquets fictifs

Cet algorithme est très rapide même lorsque les valeurs de S et $\overline{\sigma}$ sont grandes. En effet, il s'agit d'un algorithme polynomial. Il y a au plus S valeurs de s et $\overline{\sigma}$ valeurs de σ à considérer. Autrement dit, il faut calculer $S\times\overline{\sigma}$ valeurs de $\mu(s,\sigma)$. Pour chaque combinaison de s et de σ,

afin de calculer $\mu(s,\sigma)$, il faut considérer au plus S/2 valeurs de s', ce qui nous amène à une complexité de $O(S^2\sigma)$ pour cet algorithme.

3.3. Algorithme optimal basé sur la programmation dynamique

Dans cette section, nous décrivons les algorithmes globaux utilisant les algorithmes que nous avons développés pour résoudre les sous-problèmes de découpe à une dimension défini dans le chapitre IV. Nous montrons d'abord que le problème peut-être résolu de manière exacte par l'approche de la programmation dynamique.

Soient $\mathbf{B_0}$ l'ensemble des barres mères disponibles, et $\mathbf{C_0}$ l'ensemble des barres filles commandées. Rappelons que ces ensembles peuvent être représentés sous forme vectorielles.

Soit $\phi(\mathbf{p},\mathbf{B},\mathbf{C})$ le coût minimal du placement d'un sous-ensemble de barres filles $C \leq C_0$ sur un sous-ensemble de barres mères $B \leq B_0$ regroupé en **p** paquets.

Si les barres filles du sous-ensemble **C** et les barres mères du sous-ensemble **B** peuvent constituer un seul paquet réel ou fictif, $\phi(1,B,C)$ est le coût issu des chutes et des temps de coupe de ce paquet (cf. relation (III.9), chapitre III). Dans le cas contraire, $\phi(1,B,C)$ est mis à $+\infty$.

Ceci donne la condition limite de la procédure de la programmation dynamique.

Pour d'autres **p**, **B** et **C**, le coût $\phi(p, B,C)$ peut-être calculé en utilisant la formule de récurrence suivante :

$$\phi(p,B,C) = \min_{\substack{B'<B \\ C'<C}} \{\phi(p-1,B',C') + \phi(1,B-B',C-C')\} \qquad (V.18)$$

avec : $B \leq B_0$; $C \leq C_0$; $p \leq 1^T B$; $P = \min(1^T B_0, 1^T C_0)$

Par définition, la valeur optimale du critère est alors $\min_{p<P} \{\phi(p,B_0,C_0)\}$. La solution optimale correspondante peut-être trouvée par rétrogradation.

On remarque qu'il faut considérer pour tous les **p**, toutes les combinaisons de **B** et de **C**. Le nombre de combinaisons possibles de **B** (resp. C) est $\prod_{i=1}^{N} M_i$ (resp. $\prod_{j=1}^{n} m_j$). Par conséquent, le nombre de combinaisons de **p**, **B** et de **C** est $p \times \left(\prod_{i=1}^{N} M_i\right) \times \left(\prod_{j=1}^{n} m_j\right)$. De plus, pour chaque combinaison de **B** et de **C**, on considère toutes les combinaisons des sous-ensembles de **B** et des sous-ensembles de **C**. Le nombre de ces sous-ensembles augmente exponentiellement avec le nombre de barres mères disponibles et le nombre de barres filles commandées. Il n'est donc pas possible d'utiliser cet algorithme pour résoudre des problèmes réels de grande taille.

Dans la suite, pour réduire l'ensemble des solutions examinées, nous ne considérons pas toutes les combinaisons de sous-ensembles comme dans la programmation dynamique, mais nous faisons varier uniquement les **valeurs des variables** à prendre en compte. Les variables considérées sont :

- **x**, nombre de barres mères disponibles,
- **y**, nombre de barres filles à réaliser,
- **p**, nombre de paquets.

Nous ne modifions les variables que deux à deux. Nous cherchons à chaque étape la meilleure combinaison. Le temps d'exécution, sur un Sun SparcStation 10, devient alors admissible, mais le nombre de solutions examinées en est considérablement réduit.

Cette profusion d'algorithmes (5 au total) a pour objet de diminuer la fonction coût tout en ayant un temps de réponse raisonnable.

3.4. **Algorithme 1** (par modification des nombres de paquets et de barres mères)

3.4.1. Description de l'algorithme 1 : paquets/barres mères

Le premier algorithme mis en œuvre fait varier le nombre de paquets et le nombre de barres mères. Nous cherchons le meilleur placement pour **p** paquets constitués de **x** barres mères.

Soient :

- $\xi(p,x)$, le plus petit coût calculé d'après la relation (III.9) pour **p** paquets et **x** barres mères utilisées,
- B_0 est le vecteur des barres mères disponibles en stock,
- C_0 est le vecteur des barres filles commandées,
- **U(p,x)**, le vecteur correspondant des barres mères encore disponibles,
- **V(p,x)**, le vecteur correspondant des barres filles restantes à réaliser,
- **h(x,B,C)**, le "meilleur" coût obtenu en réalisant une partie des barres filles du vecteur **C** dans un paquet constitué de **x** barres mères appartenant à **B**. Nous obtenons ce coût à partir d'une heuristique donnant le placement sur un paquet de **x** barres mères disponibles. Ce problème peut se réduire au ***sous-problème 1*** (cf. chapitre IV), il suffit de diviser le nombre de barres filles de chaque type par **x**, de parcourir tous les types de barres mères, et enfin de résoudre ce sous-problème pour chaque type de barres mères comme dans le chapitre IV, section 3. La recherche du placement optimal en utilisant l'algorithme pseudo-polynomial n'est pas envisageable car les longueurs des barres mères sont importantes par rapport à l'unité de mesure. Pour gagner du temps, l'heuristique que nous utilisons est du type FFD.

"La formule de récurrence" de l'algorithme est alors la suivante :

$$\xi(1, x) = h(x, B_0, C_0) \tag{V.19}$$

$$\xi(p,x) = \min_{p-1 \le x' < x} \{\xi(p-1, x') + h(x'-x, U(p-1, x'), V(p-1, x'))\} \tag{V.20}$$

x doit être inférieur au minimum du nombre de barres mères disponibles $|B_0|$ et du nombre de barres filles commandées $|C_0|$, c'est-à-dire :

$$X = \min\left(\sum 1^T B_0, \sum 1^T C_0\right) \tag{V.21}$$

x' doit être supérieur ou égal au nombre de paquets car il faut au moins une barre mère dans chaque paquet.

p doit être inférieur à un nombre maximal de paquets, noté P.

$$P = \min(X,30) \tag{V.22}$$

Nous l'avons tronqué à 30, i.e. 3 fois plus que le nombre moyen de paquets des exemples étudiés.

Nous mémorisons à chaque étape la découpe réalisée ainsi que les barres filles restant à réaliser et les barres mères encore disponibles.

Notons x^*, la valeur de x' donnant le meilleur coût $\xi(p,x)$ pour la relation (V.20). Les vecteurs des barres mères et des barres filles pour le doublet (p,x) sont :

$$U(p,x) = U(p-1,x^*) - u(p-1,x^*, U(p-1,x^*), V(p-1,x^*)) \tag{V.23}$$

où :

- **U(p-1,x*)** est le vecteur des barres mères restantes avant le placement du dernier paquet ;
- **u(„„)** est le vecteur des barres mères utilisées dans ce nouveau paquet.

$$V(p,x) = V(p-1,x^*) - v(p-1,x^*, U(p-1,x^*), V(p-1,x^*)) \tag{V.24}$$

où :

- **V(p-1,x*)** est le vecteur des barres filles restantes avant le placement du dernier paquet ;
- **v(„„)** est le vecteur des barres filles réalisées dans ce nouveau paquet.

Lorsque pour un couple (p,x), avec p = {1,..., P} et x = {1,..., X}, il ne reste plus de barres filles à réaliser, et si le coût est inférieure à celui déjà trouvé, nous retenons ce placement.

Le programme se termine une fois que tous les nombres possibles (P) de paquets sont examinés. La meilleure solution retenue est alors la meilleure pour cet algorithme.

3.4.2. Algorithme 1 : paquets/barres mères

Schématiquement, l'algorithme s'écrit comme suit :

ALGORITHME 1 : PAQUETS/BARRES MÈRES

1. Données

- *C_0, vecteur des barres filles candidates dont chaque élément j correspond à la quantité de barres filles de type j à découper, avec j = 1,...,n*
- *B_0, vecteur des barres mères disponibles dont la $i^{ème}$ composante correspond à la quantité de barres mères de type i, avec i = 1,...,N*

2. Initialiser

- *V(0,0) = C_0, (V(p,x) est le vecteur des barres filles restantes à réaliser dont le premier paramètre correspond au nombre p de paquets placés et le second paramètre correspond au nombre x de barres mères découpées)*
- *U(0,0) = B_0, (U(p,x) est le vecteur correspondant aux quantités de barres mères encore disponible de chaque type et dont le premier paramètre correspond au nombre p de paquets placés et le second paramètre correspond au nombre x de barres mères découpées)*
- *$\mu(s, \sigma)$, coûts des coupes et des installations des paquets de taille s sur la scieuse (calculés avec l'algorithme du calcul des coûts de paquets fictifs)*
- *les coûts $\xi(0,x) = 0$ avec x variant de 1 à X (nombre maximal de barres mères)*

*3. Pour tout **nombre p de paquets**, p variant de 1 à P (nombre maximal de paquets)*

*3.1. Pour tout **nombre x de barres mères**, x variant de p à X*

3.1.1. Initialiser le coût $\xi(p,x)$ à $+\infty$

*3.1.2. Pour tous les **placements** déjà étudiés **de p-1 paquets**, le nombre de barres mères déjà découpées x' variant de p-1 à x-1*

*3.1.2.1. Pour tout **type i de barres mères**, avec i variant de 1 à N*

3.1.2.1.1. Si ce qui reste de barres mères de type i ($i^{ème}$ élément de U(p-1,x')) est au moins aussi grand que (x-x')

3.1.2.1.1.1. Diviser ce qui reste de barres filles de chaque type par (x-x') et mettre la partie entière dans le vecteur C' afin de réaliser un seul paquet :

$$C' = \left\lfloor \frac{V(p-1,x')}{x-x'} \right\rfloor$$

3.1.2.1.1.2. Appliquer l'heuristique FFD pour découper dans une barre mère de type i une partie des éléments du vecteur C'. Le coût du placement est f(i,C'). Le placement est stocké dans E et σ est le nombre de coupes correspondantes

3.1.2.1.1.3. Si $(f(i,C')\times(x-x')+\mu(x-x',\sigma)+\xi(p-1,x'))<\xi(p,x)$, alors mettre à jour les variables

$\xi(p,x) = f(i,C')\times(x-x')+\mu(x-x',\sigma)+\xi(p-1,x')$;
$E^* = E$; $i^* = i$; $x^* = x'$

3.1.2.1.1.4. Fin si

3.1.2.1.2. Fin si

3.1.2.2. Fin pour

3.1.3. Fin pour

3.1.4. Soustraire les barres mères utilisées du stock disponible et les barres filles réalisées de la liste des commandes

$$U(p,x) = U(p\text{-}1,x^*) - s^* \times K(i^*) \qquad (V.25)$$

avec $s^* = x - x^*$ *et le vecteur* $K(i^*)$ *est calculé suivant la relation (V.4)*

$$V(p,x) = V(p\text{-}1,x^*) - s^* \times E^* \qquad (V.26)$$

3.2. Fin pour

4. Fin pour

3.4.3. Exemple d'application de l'algorithme 1 : paquets/barres mères

Les données sont :

Stock :

- 3 barres de longueur 14 m ;
- 3 barres de longueur 9 m.

Commande :

- 3 barres filles de longueur 6 m à tolérance normale ;
- 8 barres filles de longueur 4 m à tolérance normale.

La largeur du trait de scie γ est de 0,01 m. Les coûts de coupe sont donnés dans le tableau V.3.

Le nombre de traits de scie pouvant être négligés dans le cas d'une découpe de barres filles à tolérance normale est $n_S = 2$.

1. Les données de départ sont :
   - $n_S = 2$; $\gamma = 0{,}01$;
   - $C_0 = (3,8)$, avec :

     $l_1 = 6$; $\tau_1 = 0$; $m_1 = 3$;

     $l_2 = 4$; $\tau_2 = 0$; $m_2 = 8$;
   - $B_0 = (3,3)$, avec :

     $L_1 = 14$; $T_1 = 0$; $M_1 = 3$;

     $L_2 = 9$; $T_2 = 0$; $M_2 = 3$.
2. L'algorithme initialise les variables :
   - Les coûts $\xi(0,x)$ sont initialisés à 0, avec $x = 0, \ldots, X$;
   - D'après la relation (V.21), $X = \min(6,11) = 6$;
   - Le nombre maximal de paquets, d'après la relation (V.22), est $P = 3$;
   - Les valeurs des coûts $\mu(s, \sigma)$ sont calculées dans l'exemple de la section 3.2 ;
   - $U(0,0) = (3,3)$;
   - $V(0,0) = (3,8)$.
3. L'algorithme parcourt tous les nombres **p** de paquets en commençant par $p = 1$.

   3.1. L'algorithme parcourt tous les nombres **x** de barres mères en commençant par $x = 1$.

3.1.1. Le coût $\xi(p,1)$ est initialisé à $+\infty$.

3.1.2. L'algorithme parcourt tous les placements déjà étudiés en **p-1** paquets, c'est-à-dire dans notre cas le doublet (p-1,x') = (0,0).

3.1.2.1. L'algorithme parcourt tous les types **i** de barres mères en commençant par i = 1.

3.1.2.1.1. Le nombre de barres mères à découper est x = 1. L'algorithme 1 se poursuit car ce qui reste de barres mères de chaque type est au moins aussi grand que (**x-x'**).

3.1.2.1.1.1. Il n'y a pas de division du nombre de barres filles car (x-x') = 1.

3.1.2.1.1.2. Le FFD appliqué à la barre mère de type 1 donne le placement : E = (2,0) ; $\sigma = 2$.

D'après la section 2 du chapitre IV, le coût de la découpe des barres filles du vecteur C' = V(0,0) dans la barre mère de type 1, est $c_b = \rho(L_i,E)$. Aussi le coût du placement est :

$f(1,C') = c_b = 2$.

3.1.2.1.1.3. La condition $(f(1,C')\times(1)+\mu(1,2)+\xi(0,0))<\xi(1,1)$ est vérifiée. L'algorithme retient ce placement et son coût :

$E^* = (2,0)$; $i^* = 1$; $x^* = 0$;

$\xi(p,x) = f(1,C')\times(1)+\mu(1,2)+\xi(0,0)$

$= 2\times1 + 0,015$

$\xi(1,1) = 2,015$.

La boucle est terminée pour le type i = 1 de barres mères. L'algorithme passe au type suivant de barres mères.

3.1.2.1.(1) Pour le type i = 2 de barres mères, le nombre de barres mères à découper est x = 1.

3.1.2.1.1.2. Le FFD appliqué à la barre mère de type 2 donne le placement : E' = (1,0).

Le coût de la découpe f(2,C') est 3.

3.1.2.1.1.3. Cette découpe **E'** n'est pas meilleure que celle déjà retenue **E***. L'algorithme ne retient pas ce placement.

3.1.4. Il n'y a plus de placement à parcourir, il faut maintenant soustraire les barres mères retenues du stock disponible et les barres filles réalisées de la liste des commandes à réaliser.

D'après la relation (V.25) U(1,1) = (3,3) - 1×(1,0) = (2,3).

D'après la relation (V.26) V(1,1) = (3,8) - 1×(2,0) = (1,8).

La boucle est terminée pour un nombre x = 1 de barres mères.

L'algorithme continue pour les nombres suivants de barres mères et de paquets. Pour simplifier la suite de la présentation, nous utiliserons un tableau (cf. tableau V.5). Les notations sont les mêmes que pour p = 1 et x = 1.

"-" : Découpe impossible

Les valeurs barrées ne sont pas retenues dans la suite du calcul.

p	x	Init. ξ(p,x)	x'	i	x-x'	C'	E	σ	f(i,C')	Si plact retenu E*	i*	x*	ξ(p,x)	Qté restante U	V
1	1	+∞	0	1	1	3,8	2,0	2	2	2,0	1	0	2,015	2,3	1,8
				2			1,0	1	3				~~3,015~~		
	2	+∞	0	1	2	1,4	1,2	2	0	1,4	1	0	0,022	1,3	1,4
				2			1,0	1	3				~~6,016~~		
	3	+∞	0	1	3	1,2	1,2	2	0	1,2	1	0	0,042	0,3	0,2
				2			1,0	1	3				~~9,031~~		
2	2	+∞	1	1	1	1,8	1,2	2	0	1,2	1	1	2,017	1,3	0,4
				2			1,0	1	3				~~5,030~~		
	3	+∞	1	1	2	1,4	1,2	2	0				~~2,037~~		
				2			1,0	1	3				~~5,031~~		
			2	1	1	1,4	1,2	2	0	1,2	1	2	0,042	0,3	0,2
				2			1,0	1	3				~~3,037~~		
	4	+∞	1	1	3	0,2	0,2	2	6				~~20,05~~		
				2			0,2	2	1				~~5,057~~		
			2	1	2	0,2	0,2	2	6				~~12,04~~		
				2			0,2	2	1				~~2,044~~		
			3	1	1	0,2	0,2	2	6				~~6,062~~		
				2			0,2	2	1	0,2	2	3	1,062	0,2	0,0
3	...														

Tableau V.5 : Récapitulatif des étapes de l'algorithme 1 suivant p et x

L'algorithme continue pour d'autres combinaisons de nombre de paquets et de nombre de barres mères. Finalement, l'algorithme conclura que ce dernier placement, dont le coût est ξ(2,4) = 1,062, est le meilleur. Le placement final est illustré dans la figure V.18.

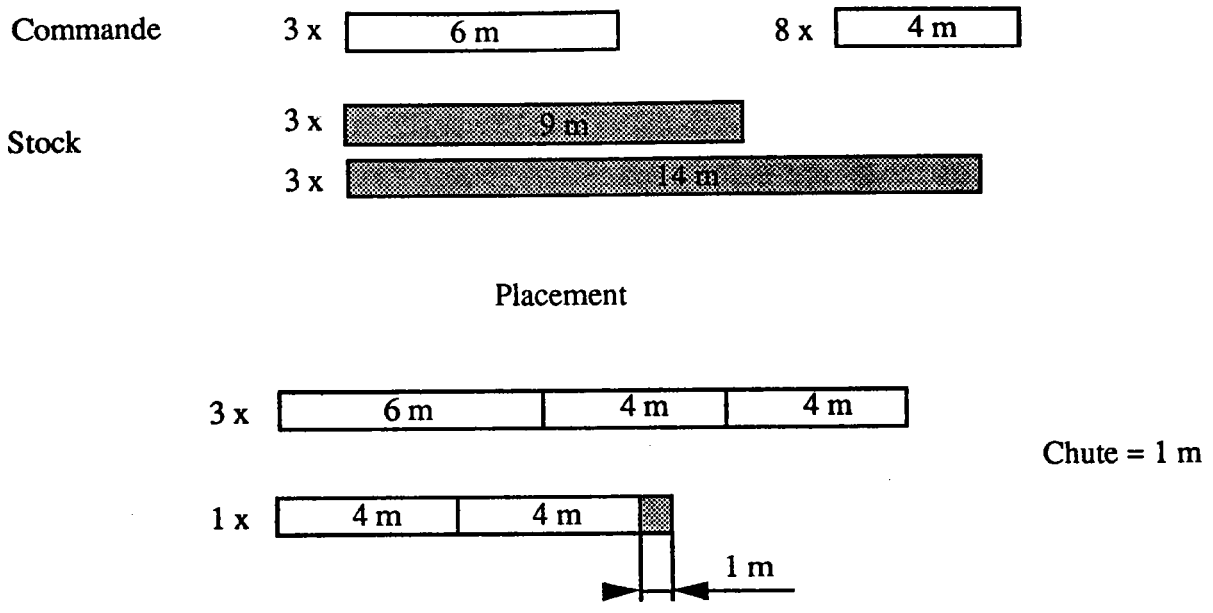

Figure V.18 : Résulat du placement avec l'algorithme 1

3.4.4. Conclusion relative à l'algorithme 1 : paquets/barres mères

Cet algorithme est très rapide (10 secondes en moyenne) pour des affaires d'environ 350 barres filles de 7 types différents à découper dans 80 barres mères. Ces dernières sont de même longueur pour pouvoir comparer les placements trouvés avec ceux de l'ancien programme de notre partenaire. Les résultats sont très satisfaisants (voir tableau V.9) : l'amélioration par rapport au programme existant est de 4,64 % en moyenne avec la fonction coût définie par la relation (III.9).

Du point de vue informatique, pour réduire fortement le besoin en mémoire vive, nous avons opté pour l'utilisation de listes chaînées (utilisation dynamique de la mémoire) qui sont plus complexes dans leur élaboration que l'utilisation statique de la mémoire (tableaux, structures, ...). Nous avons aussi, par ce biais, réduit le temps d'exécution du programme en ne considérant que les sous-ensembles de solutions retenus.

3.5. **Algorithme 2** (algorithme 1 utilisant un algorithme de placement optimal local)

3.5.1. Description de l'algorithme 2 -: optimal local

Le deuxième algorithme est une amélioration du premier. Nous cherchons le placement **optimal** sur une barre mère lorsque le cardinal de l'ensemble dans lequel on effectue le choix des barres filles à placer est faible (≤10). Le temps d'exécution devient trop important au-delà. Deux méthodes sont développées.

La première est décrite dans le premier sous-problème du chapitre IV, section 3. Il s'agit de la méthode de la programmation dynamique. Le temps d'exécution est long par rapport au nombre restreint de barres filles. Par conséquent, nous avons développé une autre méthode.

La seconde méthode est une méthode d'énumération explicite. Il s'agit de considérer une à une toutes les combinaisons $v = \sum_{j \in C} m_j$ de barres filles du vecteur **C** et de calculer le coût $\rho(i,E)$ de chacun de ses placements sur une barre mère de type i. Le meilleur coût est retenu dans f(i,C). Afin de définir la $c^{ième}$ combinaison avec c = 1 à 2^v, nous utilisons la relation du changement d'unité décimale en unité binaire $(c/2^{j-1})$ pour savoir si une barre fille **j** est placée ou non sur la barre mère. Lorsque la barre fille **j** est placée sur la barre mère, $(c/2^{j-1})$ est impaire et paire sinon.

3.5.2. Algorithme d'énumération explicite

L'algorithme s'énonce de la manière suivante :

ALGORITHME D'ÉNUMÉRATION EXPLICITE

1. Données
- *L_i, longueur de la barre mère de type i*
- *C, vecteur des barres filles non placées dont chaque élément j correspond à la quantité m_j de barres filles de type j restant à découper, avec j = 1,...,n*

2. Initialiser
- *E = ∅, vecteur des barres filles placées sur la barre mère (les composantes de E sont les nombres de barres filles de chaque type placées)*
- *$f^*(i,C) = +\infty$,meilleur coût f*

3. Calculer le nombre maximal de barres filles $v = \sum_{j \in C} m_j$ et considérer les barres filles comme toutes différentes

*4. Pour toutes les **combinaisons** c variant de 1 à 2^v*

*4.1. Pour toute **barre fille** d'indice **j**, j variant de 1 à v*

4.1.1. Si $(c/2^{j-1})$ est impair, cette barre fille est ajoutée à E

4.2. Fin pour

4.3. Calculer le coût $\rho(L_i,E)$ pour cette combinaison

4.4. Si le coût calculé est meilleur que $f^*(i,C)$, alors

$f^*(i,C) = \rho(L_i,E)$

5. Fin pour.

Pour un nombre de barres filles inférieur à 10, le temps de scrutation de toutes les combinaisons reste encore acceptable. Le temps d'exécution est, en moyenne, de 2 minutes, sur un Sun SparcStation 10, pour les 16 exemples traités.

Intégrons à présent cet algorithme dans l'algorithme 1.

3.5.3. Algorithme 2 : optimal local

Schématiquement, l'algorithme s'écrit comme suit :

ALGORITHME 2 : OPTIMAL LOCAL

1. Données

- *C_0, vecteur des barres filles candidates dont chaque élément j correspond à la quantité de barres filles de type j à découper, avec j = 1,...,n*
- *B_0, vecteur des barres mères disponibles dont la $i^{ème}$ composante correspond à la quantité de barres mères de type i, avec i = 1,...,N*

2. Initialiser

- *V(0,0) = C_0, (V(p,x) est le vecteur des barres filles candidates dont le premier paramètre correspond au nombre p de paquets placés et le second paramètre correspond au nombre x de barres mères découpées)*
- *U(0,0) = B_0, (U(p,x) est le vecteur correspondant aux quantités de barres mères de chaque type et dont le premier paramètre correspond au nombre p de paquets placés et le second paramètre correspond au nombre x de barres mères découpées)*
- *$\mu(s, \sigma)$, coûts de σ coupes et d'installation des paquets de taille s sur la scieuse*
- *les coûts $\xi(0,x)$ à 0 avec x variant de 1 à X (nombre maximal de barres mères)*

*3. Pour tout **nombre p de paquets**, p variant de 1 à P (nombre maximal de paquets)*

*3.1. Pour tout **nombre x de barres mères**, x variant de p à X*

3.1.1. Initialiser le coût $\xi(p,x)$ à $+\infty$

*3.1.2. Pour tous les **placements** déjà étudiés réalisant **p-1 paquets**, avec le nombre de barres mères déjà découpées x' variant de p-1 à x-1,*

*3.1.2.1. Pour tout **type i de barres mères**, avec i variant de 1 à N*

3.1.2.1.1. Si ce qui reste de barres mères de type i est au moins aussi grand que (x-x')

3.1.2.1.1.1. Diviser ce qui reste de barres filles de chaque type par (x-x') et mettre la partie entière dans le vecteur C' afin de réaliser un seul paquet :

$$C' = \left\lfloor \frac{V(p-1,x')}{x-x'} \right\rfloor$$

3.1.2.1.1.2. Appliquer au choix la programmation dynamique ou la méthode d'énumération explicite, pour découper dans une barre mère de type i une partie des éléments du vecteur C'. Le coût du placement est f(i,C'). Le placement est stocké dans E et σ est le nombre de coupes correspondant

3.1.2.1.1.3. Si $(f(i,C')\times(x-x')+\mu(x-x',\sigma)+\xi(p-1,x'))<\xi(p,x)$, alors Mettre à jour les variables

$\xi(p,x) = f(i,C')\times(x-x')+\mu(x-x',\sigma)+\xi(p-1,x')$;
$E^* = E$; $i^* = i$; $x^* = x'$

3.1.2.1.1.4. Fin si

3.1.2.1.2. Fin si

3.1.2.2. Fin pour

3.1.3. Fin pour

3.1.4. Soustraire les barres mères utilisées du stock disponible et les barres filles réalisées de la liste des commandes

$U(p,x) = U(p-1,x^*) - s^* \times K(i^*)$

avec $s^* = x - x^*$ *et le vecteur* $K(i^*)$ *est calculé suivant la relation (V.4)*

$V(p,x) = V(p-1,x^*) - s^* \times E^*$

3.2. Fin pour

4. Fin pour.

3.5.4. Conclusion relative à l'algorithme 2 : optimal local

Nous ne donnons pas d'exemple d'application pour l'algorithme 2 car la démarche est pratiquement identique à l'algorithme 1 à ceci près que : les placements locaux ne sont plus réalisés par l'algorithme FFD mais par l'algorithme d'énumération explicite. Il faut remarquer que dans l'exemple cité pour l'algorithme 1, la solution est atteinte plus rapidement avec cet algorithme. En effet, le choix au premier placement de réaliser 2 barres filles de type 1 de longueur 6 m sur une barre mère de type 1 de 14 m n'est pas retenu car la chute est trop importante (2 m).

Les solutions sont améliorées de 0,36 % en moyenne par rapport à celles trouvées avec l'algorithme 1 (voir tableau V.9).

3.6. **Algorithme 3** (par modification des nombres de paquets et de barres filles)

3.6.1. Description de l'algorithme 3 : paquets/barres filles

Le troisième algorithme consiste à faire évoluer le nombre de paquets et le nombre de barres filles à découper.

Nous cherchons $\zeta(\mathbf{p},\mathbf{y})$ le "meilleur" coût calculé pour **p** paquets et pour **y** barres filles à réaliser d'après la relation (III.9).

Soient :

- $\mathbf{B_0}$ est le vecteur des barres mères disponibles en stock,
- $\mathbf{C_0}$ est le vecteur des barres filles commandées,
- **U(p,x)**, le vecteur correspondant des barres mères encore disponibles,
- **V(p,x)**, le vecteur correspondant des barres filles restantes à réaliser,
- **h(y,B,C)** le "meilleur" coût obtenu en réalisant un paquet de y barres filles appartenant au vecteur **C**, et en utilisant les barres mères du vecteur **B**. Ce coût est obtenu à partir d'une heuristique donnant le placement sur un paquet de y barres filles appartenant à **C**. L'heuristique parcourt tous les types de barres mères, ce qui nous ramène au deuxième sous-problème de la section 4 du chapitre IV. Nous utilisons alors la méthode de

permutation des barres filles (cf. chapitre IV, section 4.2) pour placer les barres filles sur le paquet étudié. L'objectif de cette méthode est de placer un nombre **z** de barres filles du vecteur C sur une barre mère du stock choisie.

La formule de récurrence de l'algorithme 3 est la suivante :

$$\zeta(1, y) = h(y, B_0, C_0) \tag{V.27}$$

$$\zeta(p, y) = \min_{p-1 \le y' < y} \{\zeta(p-1, y') + h(y-y', U(p-1, y'), V(p-1, y'))\} \tag{V.28}$$

où :

- **y'** est le nombre de barres filles déjà réalisées et **y'** doit être supérieur ou égal au nombre de paquets -1 ;
- **(y'-y)** est le nombre de barres filles à réaliser dans le dernier paquet ;
- **p** doit être inférieur au nombre maximal de paquets, noté P (cf. relation (V.22)).

Nous mémorisons à chaque étape la découpe réalisée et ce qui reste de barres filles à réaliser et de barres en stocks utilisables.

Notons y^*, la valeur de **y'** donnant le meilleur coût $\zeta(p,y)$ pour la relation (V.28). Les vecteurs des barres mères et des barres filles pour le doublet (p,y) sont :

$$U(p,y) = U(p-1,y^*) - u(p-1,y^*, U(p-1,y^*), V(p-1,y^*)) \tag{V.29}$$

où :

- **$U(p-1,y^*)$** est le vecteur des barres mères restantes avant le placement des **(y'-y)** barres filles ;
- **u(,,,)** est le vecteur des barres mères utilisées pour le placement des **(y'-y)** barres filles.

$$V(p,x) = V(p-1,x^*) - v(p-1,x^*, U(p-1,x^*), V(p-1,x^*)) \tag{V.30}$$

où :

- **$V(p-1,x^*)$** est le vecteur des barres filles restantes avant le placement des **(y'-y)** barres filles ;
- **v(,,,)** est le vecteur des barres filles réalisées pour le placement des **(y'-y)** barres filles.

Lorsque pour un doublet (p, y), avec $p \in \{1,..., P\}$, il ne reste plus de barre fille à réaliser, et si la valeur du critère est meilleure (inférieure à) que celle déjà trouvée, nous retenons ce placement.

Le programme se termine une fois que tous les paquets ont été examinés. La meilleure solution retenue est alors la solution finale de cet algorithme.

3.6.2. Algorithme 3 : paquets/barres filles

Schématiquement, l'algorithme s'écrit comme suit :

ALGORITHME 3 : PAQUETS/BARRES FILLES

1. Données

- *C_0, vecteur des barres filles candidates dont chaque élément j correspond à la quantité de barres filles de type j à découper, avec j = 1,...,n*
- *B_0, vecteur des barres mères disponibles dont la $i^{ème}$ composante correspond à la quantité de barres mères de type i, avec i = 1,...,N*

2. Initialiser

- *$V(0,0) = C_0$, (V(p,y) est le vecteur des barres filles candidates dont le premier paramètre correspond au nombre p de paquets placés et le second paramètre correspond au nombre y de barres filles découpées. Les éléments m_j du vecteur (avec j = 1,...,n) correspondent aux quantités de barres filles de type j restant à découper)*
- *$U(0,0) = B_0$, (U(p,y) est le vecteur correspondant aux quantités de barres mères de chaque type et dont le premier paramètre correspond au nombre p de paquets placés et le second paramètre correspond au nombre y de barres filles découpées. Les éléments M_i du vecteur (avec i = 1,...,N) correspondent aux quantités de barres mères de type i encore disponible)*
- *$\mu(s, \sigma)$, coûts de σ coupes et d'installation des paquets de taille s sur la scieuse*
- $S = min\left(\max_{1\le i\le N}(M_i), \max_{1\le j\le n}(m_j) \right)$ *(V.31)*
- *les coûts $\zeta(0,y)$ à 0 avec y = 0, ...,* $\sum_{1\le j\le n} m_j$ *(nombre total de barres filles)*

3. Pour tout ***nombre p de paquets****, p variant de 1 à P (nombre maximal de paquets)*

3.1. Pour tout ***nombre y de barres filles****, y variant de p à* $\sum_{1\le j\le n} m_j$ *(nombre total de barres filles)*

3.1.1. Initialiser le coût $\zeta(p,y)$ à $+\infty$

3.1.2. Parcourir tous les ***placements*** *déjà étudiés réalisant* ***p-1 paquets*** *avec y' variant de p-1 à y-1*

3.1.2.1. Pour toute ***taille s de paquets****, s variant de 1 à S (taille maximale d'un paquet)*

3.1.2.1.1. Si (y-y') est un multiple de s

3.1.2.1.1.1. Diviser ce qui reste des nombres de barres filles de chaque type par s et les mettre dans le vecteur C' :

$$C' = \left\lfloor \frac{V(p-1,y')}{s} \right\rfloor$$

3.1.2.1.1.2. Diviser ce qui reste des nombres de barres mères de chaque type par s et les mettre dans le vecteur B' :

$$B' = \left\lfloor \frac{U(p-1,y')}{s} \right\rfloor$$

3.1.2.1.1.3. Calculer z, nombre de barres filles à réaliser dans une barre :

$$z = (y-y')/s \qquad (V.32)$$

3.1.2.1.1.4. Pour tout ***type i de barres mères*** *de B', i variant de 1 à N avec $M_i'>0$*

3.1.2.1.1.4.1. Appliquer l'algorithme de permutation pour découper dans une barre mère de type i, z barres filles du vecteur C'. Le coût du placement est g(z,i,C'). Le placement est stocké dans E et σ est le nombre de coupes correspondant

3.1.2.1.1.4.2. Si le coût de ce paquet (g(z,i,C')×s+μ(z,σ) ajouté au coût des placements précédents est inférieur au coût ζ(p,y), alors

Retenir ce coût calculé dans ζ(p,y) et mémoriser $E^* = E'$; $y^* = y'$; $i^* = i$ *et* $s^* = s$

3.1.2.1.1.5. Fin pour

3.1.2.1.2. Fin si

3.1.2.2. Fin pour

3.1.3. Fin parcourir

3.1.4. Soustraire les barres mères utilisées du stock disponible et les barres filles réalisées de la liste des commandes

$U(p,y) = U(p-1,y^*) - s^* \times K(i^*)$

(le vecteur $K(i^*)$ *est calculé suivant la relation (V.4))*

$V(p,y) = V(p-1,y^*) - s^* \times E^*$

3.2. Fin pour

4. Fin pour.

3.6.3. Exemple d'application de l'algorithme 3 : paquets/barres filles

Pour bien percevoir la différence avec les algorithmes précédents, nous appliquons l'algorithme 3 aux données de l'exemple de la section 3.3.

1. Les données de départ sont :
   - $n_S = 2$; $\gamma = 0,01$;
   - $C_0 = (3,8)$, avec :

     $l_1 = 6$; $\tau_1 = 0$; $m_1 = 3$;

     $l_2 = 4$; $\tau_2 = 0$; $m_2 = 8$;
   - $B_0 = (3,3)$, avec :

     $L_1 = 14$; $T_1 = 0$; $M_1 = 3$;

     $L_2 = 9$; $T_2 = 0$; $M_2 = 3$.
2. L'algorithme initialise les variables :
   - La taille maximale d'un paquet S = 3 ;
   - Le nombre maximal de paquets, d'après la relation (V.22), est P = 3 ;
   - Les valeurs des coûts μ(s, σ) sont calculées dans l'exemple de la section 3.2 ;
   - $U(0,0) = B_0 = (3,3)$; $V(0,0) = C_0 = (3,8)$; $\sum_{1 \le j \le n} m_j = 12$;
   - Les coûts ζ(0,y) sont initialisés à 0, avec x = 0, ...,12.

3. L'algorithme parcourt tous les nombres **p** de paquets en commençant par un nombre p = 1 de paquets.

3.1. L'algorithme parcourt tous les nombres **y** de barres filles en commençant par un nombre y = 1.

3.1.1. Le coût $\zeta(1,1)$ est initialisé à $+\infty$.

3.1.2. L'algorithme parcourt tous les placements déjà étudiés en p-1 paquets, c'est-à-dire dans notre cas le doublet (p-1,y') : (0,0).

3.1.2.1. L'algorithme parcourt toutes les tailles **s** de paquets en commençant par une taille s = 1.

3.1.2.1.1. Le nombre de barres filles à placer est (y-y') = 1. L'algorithme se poursuit car s est un multiple de (y'-y).

3.1.2.1.1.1. Il n'y a pas de division du nombre de barres filles/mères de chaque type car s = 1.

D'après la relation (V.32), le nombre de barres filles à placer sur une barre mère est z = 1.

3.1.2.1.1.4. L'algorithme parcourt tous les types **i** de barres mères en commençant par le type i = 1.

Le placement sera supplanté par la découpe de la barre mère de type i = 2, nous passons directement au type de barres mères suivant.

3.1.2.1.1.4.(1) Pour le type i = 2 de barres mères, l'algorithme de permutation donne :

E = (0,1) ; σ = 1 ; g(1,2,C') = 3

dans ce cas B' = U(0,0) = (3,3).

La condition $(g(1,2,C')\times s+\mu(1,1)+\zeta(0,0))<\zeta(p,y)$ est vérifiée.

L'algorithme retient ce placement :

E^* = (0,1) ; i^* = 2 ; y^* = 1 ; s^* = 1 ;

$\zeta(1,1) = 3\times1 + 0{,}015 + 0 = 3{,}015$

3.1.4. Il n'y a plus de doublet à parcourir, il faut maintenant soustraire les barres mères retenues du stock disponible et les barres filles réalisées de la liste des commandes à réaliser :

d'après la relation (V.25) U(1,1) = (3,3) - 1×(0,1) = (3,2) ;

d'après la relation (V.26) V(1,1) = (3,8) - 1×(1,0) = (2,8).

La présention des étapes de l'algorithme sous forme de tableau (cf. tableau V.6). L'étape p = 1 et y = 1 est donné comme exemple.

"-" : Découpe impossible

Les valeurs barrées ne sont pas retenues dans la suite du calcul.

p	y	Init. ζ(p,y)	y'	s	y-y'	B'	C'	z	i	E	σ	g(z, i,C')	Si plact retenu E*	i*	y*	s*	ζ(p,y)	Qté rest U	V
1	1	+∞	0	1	1	3,3	3,8	1	1	1,0	1	8					~~8,015~~		
									2	1,0	1	3	1,0	2	0	1	3,015	3,2	2,8
	2	+∞	0	1	2	3,3	3,8	2	1	2,0	2	2					~~2,020~~		
									2	0,2	2	1	0,2	2	0	1	1,020	3,2	3,6
				2	2	1,1	1,4	1	1	1,0	1	8					~~16,01~~		
									2	1,0	1	3					~~6,016~~		
	3	+∞	0	1	3	3,3	3,8	3	1	1,2	2	0	1,2	1	0	1	0,025	2,3	2,6
									2	-	-	-					+∞		
				2	3	1,1	1,4	-	-	-	-	-					+∞		
				3	3	1,1	1,2	1	1	1,0	1	8					~~24,03~~		
									2	1,0	1	3					~~9,031~~		
	4	+∞	0	1	4	3,3	3,8	4	1	-							+∞		
									2	-							+∞		
				2	4	1,1	1,4	2	1	1,1	2	4					~~8,022~~		
									2	0,2	2	1	0,2	2	0	2	2,022	3,1	3,4
				3	4	1,1	1,2	-									+∞		
				4	4	-											+∞		
	5	+∞	0	1.. ..5	5	-											+∞		
	6	+∞	0	1	6	-											+∞		
				2	6	1,1	1,4	3	1	1,2	2	0					0,022	1,3	1,2
									2	-			1,2	1	0	2	+∞		
				3	6	1,1	1,2	2	1	1,1	2	4					~~12,04~~		
									2	0,2	2	1					~~3,042~~		
				4.. ..6	6	-											+∞		
	7	+∞	0	1.. ..7	-												+∞	3,3	3,8
	8	+∞	0	1.. ..8	-												+∞	3,3	3,8
	9	+∞	0	1	9	3,3	3,8	9	1	-							+∞		
									2	-							+∞		
				2	9	1,1	1,4	-									+∞		

p	y	Init. ζ(p,y)	y'	s	y-y'	B'	C'	z	i	E	σ	g(z, i,C')	Si plact retenu E*	i*	y*	s*	ζ(p,y)	Qté rest U	V
				3	9	1,1	1,2	3	1	1,2	2	0	1,2	1	0	3	0,042	0,3	0,2
									2	-							+∞		
				4.. ..9													+∞		
2	2	+∞	1	1	1	3,2	2,8	1	1	1,0	1	8					~~11,03~~		
									2	1,0	1	3	1,0	2	1	1	6,030	3,1	1,8
	3	+∞	1	1	2	3,2	2,8	2	1	1,1	2	4					~~7,035~~		
									2	0,2	2	1	0,2	2	1	1	4,035	3,1	2,6
				2	2	1,1	1,4	1	1	1,0	1	8					~~17,03~~		
									2	1,0	1	3					~~7,016~~		
			2	1	1	3,2	3,6	1	1	1,0	1	8					~~9,035~~		
									2	1,0	1	3					~~4,035~~		
	4.. .10																+∞		
	11	+∞	1.. ..8														+∞		
			9	1	2	0,1	0,1	2	1	-							+∞		
									2	0,2	2	1	0,2	2	9	1	1,062	0,2	0,0
...	...	...	...																

Tableau V.6 : Récapitulatif des étapes de l'algorithme 3 suivant p et y

L'algorithme continue pour les autres combinaisons de nombre de paquets et de nombre de barres filles. Finalement, l'algorithme conclura que ce dernier placement dont le coût est $\zeta(2,11) = 1,062$ est le meilleur. Le placement dans ce cas-ci est identique au placement de l'algorithme 1 (voir figure V.18).

3.6.4. Conclusion relative à l'algorithme 3 : paquets/barres filles

Les résultats sont meilleurs que ceux obtenus avec les deux algorithmes précédents. Ils sont améliorés de 1,25 % par rapport au premier algorithme et de 5,84 % par rapport aux résultats obtenus par nos partenaires industriels.

Le nombre de placements parcourus est plus grand que celui des précédents algorithmes car le nombre de barres filles commandées est très supérieur au nombre de barres mères utilisées. Le temps d'exécution s'en trouve augmenté d'autant : 13 minutes en moyenne.

3.7. Algorithme 4 (par modification des nombres de barres mères et de barres filles)

3.7.1. Description de l'algorithme 4 : barres mères/barres filles

Le quatrième algorithme fait varier à la fois le nombre de barres mères utilisées et le nombre de barres filles placées. Nous définissons une fonction $\omega(x,y)$ qui est le coût minimal du placement de **y** barres filles sur **x** barres mères. Soient les vecteurs $\mathbf{U(x,y)}$ et $\mathbf{V(x,y)}$ correspondant respectivement à une partie des barres mères restantes et à une partie des barres filles restantes induisant le coût minimal $\omega(x,y)$.

Soit $\mathbf{h(x,y,B,C)}$ le coût "minimal" du placement de **y** barres filles du vecteur **C** sur **x** barres mères disponibles du vecteur **B**, sur un seul paquet réel ou fictif.

Soient $\mathbf{u(x,y,B,C)}$ et $\mathbf{v(x,y,B,C)}$ respectivement le vecteur des barres mères utilisées et le vecteur des barres filles placées correspondant à ce coût minimal. Il faut noter que comme le coût minimal est calculé pour un paquet de taille **x**, $u(x,y,B,C)$ est composé de **x** barres mères de même type. Alors l'algorithme de type "programmation dynamique" (puisque ce n'est pas une procédure de programmation dynamique dans le sens rigoureux) peut-être complété en utilisant les formulations récursives suivantes :

$$\omega(0,0) = 0$$

$$\omega(x,y) = \min_{\substack{1<x'<x \\ x'\le y'<y}} \left\{\omega(x',y') + h\left(x-x',y-y',U(x',y'),V(x',y')\right)\right\} \qquad (V.33)$$

où :

- **x'** et **y'** sont respectivement le nombre de barres mères déjà utilisées et le nombre de barres filles déjà réalisées ;
- $\omega(x',y')$ est le meilleur coût pour placer **y'** barres filles sur **x'** barres mères ;
- $\mathbf{U(x',y')}$ et $\mathbf{V(x',y')}$ sont respectivement le vecteur des barres mères et le vecteur des barres filles restantes après le placement donnant le coût $\omega(x',y')$;
- **(x-x')** et **(y-y')** correspondent au nombre (x-x') de barres mères identiques à découper pour réaliser (y-y') barres filles ;
- $\mathbf{h(,,,)}$ est le coût du nouveau paquet à rajouter au placement précédent de coût $\omega(x',y')$.

Soient $\mathbf{x^*}$ et $\mathbf{y^*}$, les **x'** et **y'** dans la relation (V.33) qui donnent la valeur minimale pour $\omega(x,y)$. Les vecteurs des barres mères et des barres filles restantes après le placement de **y** barres filles sur **x** barres mères sont :

$$U(x,y) = U\left(x^*,y^*\right) - u\left(x-x^*,y-y^*,U\left(x^*,y^*\right),V\left(x^*,y^*\right)\right) \qquad (V.34)$$

où :

- $U(x^*,y^*)$ est le vecteur des barres mères restantes avant le placement du dernier paquet ;
- $u(„)$ est le vecteur des barres mères utilisées dans ce nouveau paquet.

$$V(x,y) = V(x^*,y^*) - v(x-x^*,y-y^*,U(x^*,y^*),V(x^*,y^*)) \qquad (V.35)$$

où :

- $V(x^*,y^*)$ est le vecteur des barres filles restantes avant le placement du dernier paquet ;
- $v(„)$ est le vecteur des barres filles réalisées dans ce nouveau paquet.

La solution finale est obtenue avec un coût de $\min_{1 \le x \le \min(|C_0|,|B_0|)} \omega(x,|C_0|)$ (V.36)

Pour initialiser cet algorithme, nous calculons $\omega(x,y)$ pour tous les x et les y qui peuvent constituer un seul paquet réel ou fictif. En d'autres termes, l'algorithme affecte au coût initial $\omega(x,y)$ le coût $h(x,y,B_0,C_0)$.

La question est maintenant de chercher comment calculer $h(x,y,B,C)$. Le coût $h(x,y,B,C)$ est composé de trois parties. Ce coût regroupe les chutes, les tombants et le coût de coupe. Il est très difficile de minimiser ces coûts. Ne sachant pas résoudre ce problème de manière globale, nous adoptons une démarche plus simple : nous essayons seulement de minimiser le coût entraîné par les ***chutes*** et les ***tombants***. Nous réintégrons le temps de coupe à la fin du placement. Comme nous allons le démontrer, la différence est minime quelle que soit la solution.

Pour placer y barres filles sur x barres mères, y doit être un multiple de x. Soit $z = y / x$. z est alors le nombre de barres filles à placer sur une barre mère. Quelle que soit la solution, c'est-à-dire quelles que soient les barres mères utilisées de même type et quel que soit le sous-ensemble des barres filles placées, ce nombre z de barres filles placées sur une barre mère est constant. Comme le montre la figure V.19, le nombre de coupes requises est d'au moins z-1 (sans affranchissements et sans partie restante) et d'au plus z+1 (avec un affranchissement et un bout restant).

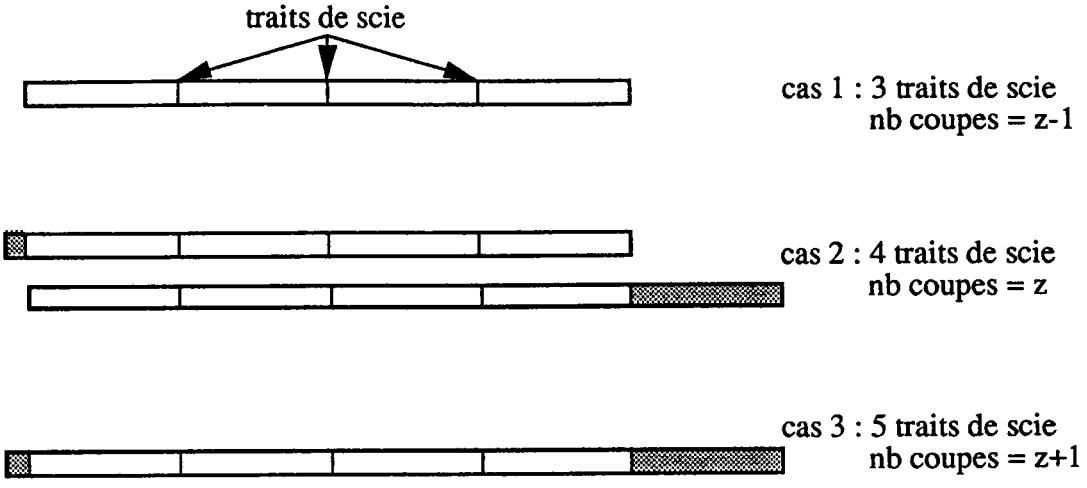

Figure V.19 : Nombre de coupes pour réaliser 4 barres filles

Dans la plupart des cas, le nombre de coupes est soit **z**, soit z+1, ce qui signifie que le coût de coupe ne dépend pas beaucoup des barres mères choisies ni de la manière dont les **z** barres filles sont placées sur chaque barre mère.

Afin d'obtenir h(x,y,B,C), nous devons considérer chaque type **i** de barres mères de **B**, tel que le nombre de barres mères restantes de ce type soit au moins égal au nombre de barres mères coupées ($b_i \geq x$). A chaque barre mère sont affectées **z** barres filles. Construisons d'autres vecteurs **B'** et **C'** tels que $B'=\lfloor B/x \rfloor$ et $C'=\lfloor C/x \rfloor$. Le problème revient à sélectionner une barre mère dans **B'** et **z** barres filles de **C'** afin que le coût total de chutes et de tombants soit minimisé. Pour une barre du stock donnée, nous savons comment sélectionner les barres filles du vecteur **C'** à réaliser de manière à minimiser le coût de chutes et de tombants. Il s'agit du second sous-problème décrit dans la section 4 du chapitre IV. Par conséquent, le calcul de h(x,y,B,C) consiste à considérer tous les types de barres mères de **B'** l'un après l'autre et à résoudre le second sous-problème pour chacun de ces types.

3.7.2. Algorithme 4 : barres mères/barres filles

Schématiquement, l'algorithme s'écrit comme suit :

ALGORITHME 4 : BARRES MÈRES/BARRES FILLES

1. Données

- *C_0, vecteur des barres filles candidates dont chaque élément j correspond à la quantité de barres filles de type j à découper, avec j = 1,...,n*
- *B_0, vecteur des barres mères disponibles dont la $i^{ème}$ composante correspond à la quantité de barres mères de type i, avec i = 1,...,N*

2. Initialiser

- *V(0,0) = C_0, (V(x,y) est le vecteur des barres filles candidates dont le premier paramètre correspond au nombre x de barres mères utilisées et le second paramètre correspond au nombre y de barres filles découpées)*

- *$U(0,0) = B_0$, ($U(x,y)$ est le vecteur correspondant aux quantités de barres mères de chaque type et dont le premier paramètre correspond au nombre x de barres mères utilisées et le second paramètre correspond au nombre y de barres filles découpées)*
- *$\mu(s, \sigma)$, coût des coupes et des installations des paquets sur la scieuse*
- *le nombre maximal X de barres mères à parcourir est le minimum entre le nombre de barres en stock et le nombre de barres filles :*

$$X = min\left(\max_{1\le i\le N}(M_i), \max_{1\le j\le n}(m_j) \right) \qquad (V.37)$$

- *le coût $\omega(0,0)$ à 0*

3. *Pour tout **nombre x de barres mères**, x variant de 1 à X*

3.1. *Pour tout **nombre y de barres filles**, y variant de x à $\sum_{1\le j\le n} m_j$ (nombre maximal de barres filles)*

3.1.1. *Initialiser le coût $\omega(x,y)$ à $+\infty$*

3.1.2. *Pour tous les **nombres x' de barres mères** déjà utilisées avec $1\le x'<x$ et y' le nombre correspondant de barres filles réalisées avec $x'\le y'<y$*

3.1.2.1. *Si (y-y') est un multiple de (x-x')*

3.1.2.1.1. *Calculer z, nombre de barres filles à réaliser dans une barre mère :*

$$z = (y-y')/(x-x') \qquad (V.38)$$

3.1.2.1.2. $B' = \left\lfloor \frac{U(x',y')}{x-x'} \right\rfloor$ *et* $C' = \left\lfloor \frac{V(x',y')}{y-y'} \right\rfloor$

3.1.2.1.3. *Pour tout **type i de barres mères** de B', i variant de 1 à N, dont la composante $M'_i>0$*

3.1.2.1.3.1. *Appliquer l'algorithme de permutation pour découper dans une barre mère de type i et une partie des éléments du vecteur C'. Le coût du placement est g(z,i,C'). Le placement est stocké dans E et σ est le nombre de coupes correspondant*

3.1.2.1.3.2. *Si le coût de ce paquet $(g(z,i,C')\times s+\mu(x-x',\sigma))$ ajouté au coût $\omega(x',y')$ des placements précédents est inférieur au coût $\omega(x,y)$, alors*

Retenir ce coût calculé dans $\omega(x,y)$ et mémoriser le placement : $E^* = E$, $x^* = x'$, $y^* = y'$ et $i^* = i$

3.1.2.1.3.3. *Fin si*

3.1.2.1.4. *Fin pour*

3.1.2.2. *Fin si*

3.1.3. *Fin pour*

3.1.4. *Soustraire les barres mères utilisées du stock disponible et les barres filles réalisées de la liste des commandes :*

$$U(x,y) = U(x^*,y^*) - s^*\times K(i^*)$$

avec $s^* = x - x^*$ et le vecteur $K(i^*)$ est calculé suivant la relation (V.4)

$V(x,y) = V(x^*,y^*) - s^* \times E^*$

3.2. Fin pour

4. Fin pour.

3.7.3. Améliorations

Les résultats sont meilleurs que ceux obtenus par les algorithmes précédents. Mais le temps nécessaire pour traiter un exemple de taille moyenne (350 barres filles à découper dans 80 barres en stock) est très long. Pour un "gros" exemple de 500 barres filles à placer sur un seul type de barres mères, il avait fallu plus de 6 heures de traitement sur un Sun SparcStation 10.

Pour diminuer le temps d'exécution, nous avons eu recours aux principes d'évaluation utilisés dans les Procédures par Séparation et Évaluation (branch and bound) pour éliminer les sous-ensembles vides de solutions réalisables ou qui ne contiennent pas de solutions améliorantes :

- les cas où la longueur totale maximale de **x** barres mères est inférieure à la longueur totale minimale de **y** barres filles. Ces nombres définissent la limite minimale du nombre de barres mères pour réaliser un nombre donné de barres filles.
- les cas où la "borne inférieure", $\mathbf{b_{inf}}$, du sous-ensemble de solutions considérées est supérieure à la borne supérieure, $\mathbf{b_{sup}}$, trouvé en utilisant l'algorithme 1. Ce dernier algorithme est moins rapide que l'algorithme FFD étendu mais il fournit une meilleure borne supérieure de la fonction économique. Pour gagner du temps, nous supposons que le coût est réparti uniformément dans toutes les barres filles réalisées. La "borne inférieure" est calculée comme suit :

$$b_{inf} = \chi(C') + \frac{b_{sup} \times \left(\sum_{j \in (C_0 - C')} l_j \right)}{\left(\sum_{j \in C_0} l_i \right) \times (1 + \varepsilon)} \qquad (V.39)$$

où :

- **C'** est le vecteur des barres filles déjà placées,
- $\chi(C')$ est le coût (chutes, tombants, temps de coupe) calculé pour le vecteur C' des barres filles déjà placées,
- $\mathbf{l_j}$ est la longueur d'une barre fille de type j,
- $(\mathbf{C_0} - \mathbf{C'})$ est le vecteur des barres filles restant à réaliser,
- ε est le paramètre correspondant à une estimation de la distance entre la solution obtenue par l'algorithme 1 et la solution optimale. Plus ε est petit, plus on est optimiste sur la performance de l'algorithme 1 et plus on a tendance à éliminer des sous-

ensembles de solutions. Sur un nombre important d'essais, les solutions restent inchangées si nous descendons ce paramètre jusqu'à 0,1. (c'est-à-dire que l'on suppose que la valeur du critère fournie par l'algorithme 1 est à 10 % de la solution optimale)

Pour réduire de moitié l'espace mémoire nécessaire, nous n'avons conservé pour chaque étape que le coût lié à un paquet, le nombre de barres filles placées, le nombre de barres mères utilisées, le stock restant et les barres filles restantes. Le placement pour chaque paquet sera alors recalculé pour la meilleure solution trouvée.

3.7.4. Exemple d'application de l'algorithme 4 : barres mères/barres filles

Nous appliquons l'algorithme 4 aux données de l'exemple de la section 3.3.

1. Les données de départ sont :
   - $n_S = 2$; $\gamma = 0,01$;
   - $C_0 = (3,8)$, avec :

     $l_1 = 6$; $\tau_1 = 0$; $m_1 = 3$;

     $l_2 = 4$; $\tau_2 = 0$; $m_2 = 8$;
   - $B_0 = (3,3)$, avec :

     $L_1 = 14$; $T_1 = 0$; $M_1 = 3$;

     $L_2 = 9$; $T_2 = 0$; $M_2 = 3$.
2. L'algorithme initialise les variables :

   Le coût $\omega(0,0)$ est initialisé à 0.

   D'après la relation (V.37), le nombre maximal X de barres mères est :

   $$X = \min(6,11) = 6.$$

   Les valeurs des coûts $\mu(s, \sigma)$ sont calculées dans l'exemple de la section 3.2.

   La borne supérieure a été calculée à l'aide de l'algorithme 1. Nous avons donc :

   $$b_{sup} = 1,062.$$

   La longueur totale des barres filles est :

   $$\left(\sum_{i \in C_0} l_i \right) = 3 \times 6 + 8 \times 4 = 50 ;$$

   $U(0,0) = B_0 = (3,3)$; $V(0,0) = C_0 = (3,8)$.
3. L'algorithme parcourt tous les nombres x de barres mères en commençant par un x = 1 de barre mère.

   3.1. L'algorithme parcourt tous les nombres y de barres filles en commençant par un nombre y = 1 de barres filles à placer.

   3.1.1. Le coût $\omega(1,1)$ est initialisé à $+\infty$.

3.1.2. L'algorithme parcourt tous les placements déjà étudiés en x-1 barres mères, c'est-à-dire dans notre cas le doublet (x-1,y') : (0,0).

3.1.2.1. (y-y') = 1 est un multiple de (x-x') = 1. L'algorithme peut continuer.

3.1.2.1.1. D'après la relation (V.38), le nombre de barres filles à placer sur une barre mère est z = 1.

3.1.2.1.2. Il n'y a pas de division des nombres de barres filles/mères de chaque type car le nombre s de barres mères dans le paquet est 1.

3.1.2.1.3. L'algorithme parcourt tous les types i de barres mères en commençant par le type i = 1.

Le placement sera supplanté par la découpe de la barre mère type i = 2, nous passons directement au type de barre mère suivant.

3.1.2.1.3.(1) Pour le type i = 2 de barres mères, l'algorithme de permutation donne :

E = (1,0) ; σ = 1 ; g(1,2,C') = 3

dans ce cas B' = U(0,0).

3.1.2.1.3.1. La condition $(g(1,2,C')\times 1+\mu(1,1)+\omega(0,0))<\omega(1,1)$ est vérifiée. L'algorithme retient ce placement :

E^* = (1,0) ; i^* = 2 ; y^* = 0 ; x^* = 0 ;

$\omega(1,1) = 3\times 1 + 0{,}015 + 0 = 3{,}015$.

3.1.4. Il n'y a plus de doublet à parcourir, il faut maintenant soustraire les barres mères retenues du stock disponible et les barres filles réalisées de la liste des barres filles à réaliser :

d'après la relation (V.25) U(1,1) = (3,3) - 1×(0,1) = (3,2) ;

d'après la relation (V.26) V(1,1) = (3,8) - 1×(1,0) = (2,8).

La boucle est terminée pour y = 1 barre fille à placer.

L'algorithme calcule la borne inférieure pour le doublet et le conserve si cette borne est inférieure au meilleur coût trouvé par l'algorithme 1 (1,062). A présent, montrons comment est calculée la borne inférieure du doublet (1,1).

Pour (1,1) et d'après la relation (V.39) :

$$l_{rest} = \left(\sum_{i\in(C_o-C')} l_i \right) = 2\times 6 + 8\times 4 = 44$$

$$b_{inf} = 3{,}015 + 1{,}062\times 44/(50\times 1{,}1) = 3{,}865.$$

Cette borne inférieure est trop grande par rapport au meilleur coût (1,062). Ce noeud est donc rejetée. "garder" est mis à "n", dans le cas contraire, on met dans "garder", "o"

L'algorithme continue de la même manière pour les autres doublets (x,y). Les étapes de l'algorithme 4 sont résumées dans le tableau V.7.

"-" : Découpe impossible

Les valeurs barrées ne sont pas retenues dans la suite du calcul à l'étape 3.1.2.1.3.2.

Les valeurs encadrées correspondent aux doublets retenus après calcul de b_{inf}.

x	y	Init. ω(x y)	x'	y'	y-y'	s=x-x'	z	B'	C'	i	E	σ	g(z,i,C')	Si placement retenu E*	x*	y*	i*	ω(x,y)	Qté rest (x,y) U	V	l_{rest}	b_{inf}	garder o/n
1	1	+∞	0	0	1	1	1	3,3	3,8	1	1,0	1	8					~~8,015~~					
										2	1,0	1	3	1,0	0	0	2	3,015	3,2	2,8	44	3,865	n
	2	+∞	0	0	2	1	2	3,3	3,8	1	2,0	2	2					~~2,020~~					
										2	0,2	2	1	0,2	0	0	2	1,020	3,2	3,6	42	1,831	n
	3	+∞	0	0	3	1	3	3,3	3,8	1	1,2	2	0	1,2	0	0	1	0,025	2,3	2,6	36	0,797	o
										2	-	-	-					+∞					
	4..11	+∞	0	0	-													+∞					n
2	2	+∞	0	0	2	2	1	1,1	1,4	1	1,0	1	8					~~16,016~~					
										2	1,0	1	3	1,0	0	0	2	6,016	3,1	1,8	38	6,750	n
	3	+∞	0	0	3	2	-											+∞					n
	4	+∞	0	0	4	2	2	1,1	1,4	1	2,0	2	2					~~4,022~~					
										2	0,2	2	1	0,2	0	0	2	2,022	3,1	3,4	34	2,679	n
			1	3	1	1	1	2,3	2,6	1	1,0	1	8					~~8,040~~					
										2	1,0	1	3					~~3,040~~					
	5	+∞	0	0	5	2	-											+∞					
			1	3	2	1	2	2,3	2,6	1	2,0	2	2					~~2,045~~					
										2	0,2	2	1	0,2	1	3	2	1,045	2,2	2,4	28	1,586	n
	6	+∞	0	0	6	2	3	1,1	1,4	1	1,2	2	0	1,2	0	0	1	0,022	1,3	1,4	22	0,447	o
										2	-							+∞					
			1	3	3	1	3	2,3	2,6	1	1,2	2	0					~~0,045~~					
										2	-							+∞					
	7..11	+∞																+∞				+∞	n
3	3	+∞	0	0	3	3	1	1,1	1,2	1	1,0	1	8					~~24,031~~					
										2	1,0	1	3	1,0	0	0	2	9,031	2,3	0,8	32	9,649	n
	4	+∞	0	0	4	3	-																
			1	3	1	2	-											+∞				+∞	n
	5	+∞	0	0	5	3	-											+∞					
			1	3	2	2	1	1,1	1,3	1	1,0	1	8					~~16,041~~					
										2	1,0	1	3	1,0	1	3	2	6,041	2,1	0,6	24	6,504	n
	6	+∞	0	0	6	3	2	1,1	1,2	1	1,1	2	4					~~12,042~~					
					-					2	0,2	2	1	0,2	0	0	2	3,042	3,0	3,2	26	3,544	n
			1	3	3	2	-											+∞					
	7	+∞	0	0	7	3	-											+∞					
			1	3	4	2	2	1,1	1,3	1	1,1	2	4					~~8,047~~					
										2	0,2	2	1	0,2	1	3	2	2,047	2,1	2,2	20	2,433	n
			2	6	1	1	1	1,3	1,4	1	1,0	1	8					~~8,037~~					
										2	1,0	1	3					~~3,037~~					
	8	+∞	0	0	8	3	-											+∞					
			1	3	5	2	-											+∞					
			2	6	2	1	2	1,3	1,4	1	1,1	2	4					~~4,042~~					
										2	0,2	2	1	0,2	2	6	2	1,042	1,2	1,2	14	1,312	n

x	y	Init. ω(x y)	x'	y'	y-y'	s=x-x'	z	B'	C'	i	E	σ	g(z, i, C')	Si placement retenu E*	x*	y*	i*	ω(x,y)	Qté rest (x,y) U	V	l_{rest}	b_{inf}	garder o/n
	9	+∞	0	0	9	3	3	1,1	1,2	1 2	1,2 -	2	0	1,2	0	0	1	0,042 +∞	0,3	0,2	8	0,196	o
			1	3	6	2	3	1,1	1,3	1 2	1,2 -	2	0					~~0,047~~ +∞					
			2	6	3	1	3	1,3	1,4	1 2	1,2 -	2	0					~~0,042~~ +∞					
	10	+∞	0	0	10	3	-											+∞					
			1	3	7	2	-											+∞					
			2	6	4	1	4	1,3	1,4	1 2	- -							+∞ +∞				+∞	n
	11	+∞	0	0	11	3	-											+∞					
			1	3	8	2	4	2,3	2,6	1 2	- -							+∞ +∞					
			2	6	5	1	5	1,3	1,4	1 2	- -							+∞ +∞					n
4	4	+∞	0	0	4	4	4	3,3	3,8	1 2	- -							+∞ +∞				+∞	n
	5	+∞	0	0	5	4	-											+∞					
			1	3	2	3	-											+∞				+∞	n
	6	+∞	0	0	6	4	-											+∞					
			1	3	3	3	1	0,1	0,2	1 2	0,1 0,1	1 1	10 5	0,1	1	3	2	~~30,040~~ 15,040	2,0	2,3	24	15,50	n
	7	+∞	0	0	7	4	-											+∞					
			1	3	4	3	-											+∞					
			2	6	1	2	-											+∞				+∞	n
	8	+∞	0	0	8	4	2	-										+∞					
			1	3	5	3	-											+∞					n
			2	6	2	2	1	0,1	0,2	1 2	0,1 0,1	1 1	10 5	0,1	2	6	2	~~20,037~~ 10,037	1,1	1,2	14	10,30	n
	9	+∞	0	0	9	4	-											+∞					
			1	3	6	3	2	0,1	0,2	1 2	0,2 0,2	2 2	6 1	0,1	1	3	2	~~18,067~~ 3,067	2,0	2,0	12	3,299	n
			2	6	3	2	-											+∞					
	10	+∞	0	0	10	4	-											+∞					
			1	3	7	3	-											+∞					
			2	6	4	2	2	0,1	0,2	1 2	- 0,2	2	1					+∞ ~~2,044~~					n
			3	9	1	1	1	0,3	0,2	1 2	0,1 0,1	1 1	10 5	0,1	3	9	2	~~10,057~~ 5,057	0,2	0,1	4	5,134	n

x	y	Init. ω(x y)	x'	y'	y-y'	s=x-x'	z	B'	C'	i	E	σ	g(z,i,C')	Si placement retenu E*	x*	y*	i*	ω(x,y)	Qté rest (x,y) U	V	l_{rest}	b_{inf}	garder o/n
	11	+∞	0	0	11	4	-											+∞					
			1	3	8	3	-											+∞					
			2	6	5	2	-											+∞					
			3	9	2	1	2	0,3	0,2	1	-							+∞					
										2	0,2	2	1	0,2	3	9	2	1,062	0,2	0,0	0	1,062	0

Tableau V.7 : Récapitulatif des étapes de l'algorithme 4 suivant x et y

L'algorithme continue pour d'autres combinaisons de nombre de barres mères et de nombre de barres filles. Finalement, l'algorithme conclura que ce dernier placement (doublet (4,11)) dont le coût est ω(4,11) = 1,062 est le meilleur. Pour cet exemple simple, le placement trouvé par cet algorithme est identique au placement de l'algorithme 1. Il est représenté dans la figure V.18.

3.7.5. Conclusion relative à l'algorithme 4 : barres mères/barres filles

L'algorithme décrit ci-dessus donne des solutions très satisfaisantes. Comparé aux performances moyennes de la compagnie, cet algorithme réduit le coût de plus de 8 % en moyenne comme le montre la section 3.9. Cependant, cette performance nécessite un temps de traitement relativement long. Pour un problème industriel de taille moyenne (350 barres filles à découper dans 80 barres en stock), le temps d'exécution est de quelques minutes sur un Sun SparcStation 10. Pour de vraiment gros problèmes contenant près de 1000 barres filles commandées de 15 types, le traitement nécessite 1 heure. **Au final, cet algorithme convient très bien aux plannings journaliers puisque le calcul peut-être exécuté pendant la nuit.**

Par conséquent, nous utiliserons les autres algorithmes de placement pour réaliser des devis pour les clients.

3.8. Algorithme 5 (par modification du nombre de barres filles)

3.8.1. Description de l'algorithme 5 : barres filles

Dans cette variante, nous définissons une fonction ς(y) qui est le coût le plus faible pour réaliser y barres filles. Soit θ(y',B,C) le plus faible coût pour réaliser y' barres filles tirées du vecteur C dans des barres mères définies par le vecteur B. Nous avons :

$$\varsigma(1) = \theta(1, B_0, C_0) \qquad (V.40)$$

où :

- B_0 est le vecteur des barres mères disponibles en stock ;
- C_0 est le vecteur des barres filles commandées.

Pour les autres y (y>1), nous considérons tous les **y'** tels que y'<y. Pour un **y'** donné, nous pouvons calculer $\theta(y', B, C)$ où **B** et **C** sont respectivement les barres mères restantes et les barres filles restantes après avoir découpé **y-y'** barres filles. La meilleure solution correspond alors à **y'** tel que $\varsigma(y-y') + \theta(y', B, C)$ est minimal.

$$\varsigma(y) = \min_{1 \le y' < |C_0|} \{\varsigma(y') + \theta(y - y', U(y'), V(y'))\} \qquad (V.41)$$

où :

- **y'** est le nombre de barres filles déjà réalisées ;
- **U(y')** et **V(y')** sont respectivement le vecteur des barres mères et le vecteur des barres filles restantes après le placement donnant le coût $\varsigma(y')$;
- **y-y'** est le nombre de barres filles à réaliser ;
- **θ(y-y', U(y'), V(y'))** est le coût du nouveau placement de **y-y'** barres filles du vecteur V(y') sur des barres mères du vecteur U(y').

La solution finale est alors obtenue pour $y = |C_0|$ (toutes les barres filles à réaliser sont placées). Le choix est fait de telle sorte que le coût soit le plus faible.

Soit y^*, la valeur de **y'** donnant la meilleur valeur du critère $\varsigma(y)$ dans la relation (V.41). Les vecteurs des barres mères disponibles et des barres filles à réaliser sont :

$$U(y) = U(y^*) - u(y^*, U(y^*), V(y^*)) \qquad (V.42)$$

où :

- **U(y^*)** est le vecteur des barres mères restantes avant le placement des (y-y') barres filles ;
- **u(,,)** est le vecteur des barres mères utilisées pour le placement des (y-y') barres filles.

$$V(y) = V(y^*) - v(y^*, U(y^*), V(y^*)) \qquad (V.43)$$

où :

- **V(y^*)** est le vecteur des barres filles restantes avant le placement des (y-y') barres filles ;
- **v(,,)** est le vecteur des barres filles réalisées pour le placement des (y-y') barres filles.

Le reste du problème est maintenant le calcul de $\theta(y, B, C)$. Utilisant les mêmes arguments que pour le calcul de $h(x, y, B, C)$ dans l'algorithme 4, nous pouvons montrer que ce calcul

consiste à résoudre différents sous-problèmes du second type, décrits dans la section 4 du chapitre IV. Ces sous-problèmes peuvent être résolus en utilisant l'algorithme de permutation de la section 4.2 du chapitre IV.

3.8.2. Algorithme 5 : barres filles

Schématiquement, l'algorithme s'écrit de la manière suivante :

ALGORITHME 5 : barres filles

1. Données

- *C_0, vecteur des barres filles candidates dont chaque élément j correspond à la quantité de barres filles de type j à découper, avec j = 1,...,n*
- *B_0, vecteur des barres mères disponibles dont la $i^{ème}$ composante correspond à la quantité de barres mères de type i, avec i = 1,...,N*

2. Initialiser

- *$V(0) = C_0$, vecteur des barres filles candidates dont le paramètre correspond au nombre y de barres filles découpées*
- *$U(0) = B_0$, vecteur correspondant aux quantités de barres mères de chaque type et dont le paramètre correspond au nombre y de barres filles découpées*
- *$\mu(s, \sigma)$, coûts de σ coupes et d'installation des paquets de taille s sur la scieuse*
- *la taille S maximale d'un paquet (réel ou fictif) :*

$$S = \min\left(\max_{1\le i\le N} M_i, \max_{1\le j\le n} m_j\right) \qquad (V.44)$$

- *le coût $\varphi(0)$ à 0*

*3. Pour tout **nombre y de barres filles**, y variant de 1 à $\sum_{j=1}^{n} m_j$ (nombre maximal de barres filles)*

3.1. Initialiser le coût $\varphi(y)$ à $+\infty$

*3.2. Pour tous les **placements déjà réalisés**, y' étant le nombre correspondant de barres filles réalisées (y' < y)*

*3.2.1. Pour toute **taille s de paquets**, s variant de 1 à $\min(S, y-y')$*

3.2.1.1. Si (y-y') est un multiple de s

3.2.1.1.1. Calculer z, nombre de barres filles à réaliser dans une barre

$z = (y-y')/s$

3.2.1.1.2. Diviser le nombre restant de barres filles de chaque type par s et le mettre dans le vecteur $C' = \lfloor V(y')/s \rfloor$

3.2.1.1.3. Diviser le nombre de barres du stock restantes par s et le mettre dans le vecteur $B' = \lfloor U(y')/s \rfloor$

*3.2.1.1.4. Pour tout **type i de barres mères** de B', i variant de 1 à N et dont la composante $M'_i > 0$*

3.2.1.1.4.1 Appliquer l'algorithme de permutation pour découper dans une barre mère de type i une partie des éléments du vecteur C'. Le coût du placement est g(z,i,C') et le

placement est stocké dans E et σ est le nombre de coupes correspondant

3.2.1.1.4.2. Si le coût de ce paquet ($g(z,i,C')\times s+\mu(s,\sigma)$) ajouté au coût des placements précédents est inférieur au coût $\varsigma(y)$, alors

Retenir ce coût calculé dans $\varsigma(y)$ et le placement :
$E^* = E$; $y^* = y'$; $i^* = i$ et $s^* = s$

3.2.1.1. Fin pour

3.2.1.1. Fin si

3.2.2. Fin pour

3.2.3. Soustraire les barres mères utilisées du stock disponible et les barres filles réalisées de la liste des commandes

$U(y) = U(y^*) - s^* \times K(i^*)$

(le vecteur $K(i^*)$ est calculé suivant la relation (V.4))

$V(y) = V(y^*) - s^* \times E^*$

3.3. Fin pour

4. Fin pour.

3.8.3. Exemple d'application de l'algorithme 5 : barres filles

Nous donnons la démarche de l'algorithme appliqué aux données de l'exemple de la section 3.3. afin de distinguer les différences avec les algorithmes précédents.

1. Les données de départ sont :
   - $n_S = 2$; $\gamma = 0{,}01$;
   - $C_0 = (3,8)$, avec :
     $l_1 = 6$; $\tau_1 = 0$; $m_1 = 3$;
     $l_2 = 4$; $\tau_2 = 0$; $m_2 = 8$;
   - $B_0 = (3,3)$, avec :
     $L_1 = 14$; $T_1 = 0$; $M_1 = 3$;
     $L_2 = 9$; $T_2 = 0$; $M_2 = 3$.
2. L'algorithme initialise les variables :
   - La taille maximale d'un paquet $S = \min\left(\max_{1\le i\le N}(M_i), \max_{1\le j\le n}(m_j) \right) = 3$;
   - Les valeurs des coûts $\mu(s, \sigma)$ sont calculées dans l'exemple de la section 3.2 ;
   - $U(0) = (3,3)$; $V(0) = (3,8)$;
   - Le coût $\varsigma(0)$ est initialisé à 0.
3. L'algorithme parcourt tous les nombres y de barres filles de 1 à 11 en commençant par un nombre y = 1 de barres filles à placer.
   3.1. Le coût $\varsigma(1)$ est initialisé à $+\infty$.

3.2. L'algorithme parcourt tous les placements déjà réalisés, c'est-à-dire dans notre cas le placement (y') : (0).

3.2.1. L'algorithme parcourt toutes les tailles s de paquets en commençant par une taille s = 1 de paquets.

3.2.1.1. Le nombre de barres filles à placer est (y-y') = 1. L'algorithme se poursuit car (y-y') est un multiple de s.

3.2.1.1.1. D'après la relation (V.32), le nombre de barres filles à placer sur une barre mère est z = 1.

3.2.1.1.2. Il n'y a pas de division des nombres de barres filles/mères de chaque type car le paquet n'est constitué que d'une barre mère (s = 1). Les vecteurs du sous-problème sont donc :

$C' = C_0$ et $B' = B_0$.

3.2.1.1.4. L'algorithme parcourt tous les types i de barres mères en commençant par le type i = 1.

Le placement sera supplanté par la découpe de la barre mère de type i = 2, nous passons directement au type de barres mères suivant.

3.2.1.1.4.1. Pour le type i = 2 de barres mères, l'algorithme de permutation donne le :

$E = (1,0)$; $\sigma = 1$; $g(1,2,C') = 3$;

dans ce cas $B' = B_0$.

3.2.1.1.4.2. La condition $(g(1,2,C')\times s+\mu(1,1)+\varsigma(0))<\varsigma(1)$ est vérifiée. L'algorithme retient ce placement :

$E^* = (1,0)$; $i^* = 2$; $y^* = 0$; $s^* = 1$;

$\varsigma(1) = 3\times1 + 0{,}015 + 0 = 3{,}015$.

3.2.3. Il n'y a plus de placements locaux à parcourir. Il faut maintenant soustraire les barres mères du stock disponible et les barres filles de la liste des commandes à réaliser :

d'après la relation (V.25) $U(1) = (3,3) - 1\times(0,1) = (3,2)$;

d'après la relation (V.26) $V(1) = (3,8) - 1\times(1,0) = (2,8)$.

3.(1) L'algorithme continue pour les autres nombres y de barres filles à placer.

Les étapes de l'algorithme sont résumées dans le tableau V.8. Les notations sont ceux de l'étape y =1.

CHAPITRE V

"-" : Découpe impossible

"..." : L'algorithme continue de la même manière.

Les valeurs barrées ne sont pas retenues dans la suite du calcul.

y	Init. ς(y)	y'	y-y'	s	z	B'	C'	i	E	σ	g(z, i,C')	Si plact retenu E*	y*	i*	s*	ς(y)	Qté rest (y) U	V
1	+∞	0	1	1	1	3,3	3,8	1	1,0	1	8					~~8,015~~		
								2	1,0	1	3	1,0	0	2	1	3,015	3,2	2,8
2	+∞	0	2	1	2	3,3	3,8	1	2,0	2	2					~~2,020~~		
								2	0,2	2	1	0,2	0	2	2	1,020	3,2	3,6
				2	1	1,1	1,4	1	1,0	1	8					~~16,01~~		
								2	1,0	1	3					~~6,016~~		
		1	1	1	1	3,2	2,8	1	1,0	1	8					~~11,03~~		
								2	1,0	1	3					~~6,030~~		
3	+∞	0	3	1	3	3,3	3,8	1	1,2	2	0	1,2	0	1	1	0,020	2,3	2,6
								2	-							+∞		
				2	-											+∞		
				3	1	1,1	1,2	1	1,0	1	8					~~24,03~~		
								2	1,0	1	3					~~9,031~~		
		1	2	1	2	3,2	2,8	1	2,0	2	2					~~5,035~~		
								2	0,2	2	2					~~5,035~~		
				2	1	1,1	1,4	1	1,0	1	8					~~19,03~~		
								2	1,0	1	3					~~9,031~~		
		2	1	1	1	3,2	3,6	1	1,0	1	8					~~9,036~~		
								2	1,0	1	3					~~4,036~~		
4	+∞	0	4	1	4	3,3	3,8	1	-							+∞		
								2	-							+∞		
				2	2	1,1	1,4	1	2,0	2	2					~~7,037~~		
								2	0,2	2	1					~~5,037~~		
				3	-											+∞		
				4	1	-										+∞		
		1	3	1	2	3,2	2,8	1	2,0	2	2					~~5,035~~		
								2	0,2	2	1					~~4,035~~		
				2	-											+∞		
				3	1	1,0	0,2	1	1,0	1	8					~~27,04~~		
								2	1,0	1	3					~~12,04~~		
		2	2	1	2	3,2	3,6	1	2,0	2	2					~~3,040~~		
								2	0,2	2	1	0,2	2	2	1	2,040	3,1	3,4
				2	1	1,1	1,3	1	1,0	1	8					~~17,03~~		
								2	-1,0	1	3					~~7,036~~		
		3	1	1	1	2,3	2,6	1	1,0	1	8					~~8,035~~		
								2	1,0	1	3					~~3,035~~		
5	+∞	0	5	1	5	3,3	3,8	1	-							+∞		
								2	-							+∞		
				2.. .5	- 1	-										+∞		
		1	4	1	4	3,2	2,8	1	-							+∞		
								2	-							+∞		
				2	2	1,1	1,4	1	1,1	2	4					~~11,03~~		
								2	0,2	2	1					~~5,031~~		

y	Init. ç(y)	y'	y-y'	s	z	B'	C'	i	E	σ	g(z, i,C')	Si plact retenu E*	y*	i*	s*	ç(y)	Qté rest (y) U	V
				3	-											+∞		
				4	1	-										+∞		
		2	3	1	3	3,2	3,6	1 2	1,2 -	2	0					~~1,040~~ +∞		
				2	-											+∞		
				3	1	1,0	1,2	1 2	1,0 -	1	8					~~25,05~~ +∞		
		3	2	1	2	2,3	2,6	1 2	2,0 0,2	2 2	2 1	0,2	3	2	1	~~2,040~~ 1,040	2,2	2,4
				2	1	1,1	1,3	1 2	1,0 1,0	1 1	4 1					~~16,03~~ ~~6,036~~		
		4	1	1	1	3,1	3,4	1 2	1,0 1,0	1 1	8 3					~~10,05~~ ~~5,055~~		
6	+∞	0.. .2																
		3	3	1	3	2,3	2,6	1 2	1,2 -	2	0	1,2	3	1	1	0,040 +∞	1,3	1,4
				2	-											+∞		
				3	1	0,1	0,2	1 2	- 0,1	1	5					+∞ ~~15,05~~		
		4.. .5																
7	+∞	0.. .5																
		6	1	1	1	1,3	1,4	1 2	1,0 1,0	1 1	8 3	1,0	6	2	1	~~8,190~~ 3,055	1,2	0,4
8	+∞	0.. .5																
		6	2	1	2	1,3	1,4	1 2	1,1 0,2	2 2	4 1	0,2	6	2	1	~~4,240~~ 1,060	1,2	1,2
				2	1	0,1	0,2	1 2	0,1 0,1	1 1	10 5					~~20,05~~ ~~10,05~~		
		7	...															
9	+∞	0	9	1.. .2		3,3	3,8											
		-		3	3	1,1	1,2	1 2	1,2 -	2	0	1,2	0	1	3	0,042 +∞	0,3	0,2
				4.. .9														
		1.. .8																
10	+∞	0.. .5																
		6	4	1	4	1,3	1,4	1 2	- -							+∞ +∞		
				2	2	0,1	0,2	1 2	- 0,2	2	1	0,2	6	2	2	+∞ 2,062	1,1	1,0

y	Init. ς(y)	y'	y-y'	s	z	B'	C'	i	E	σ	g(z, i,C')	Si plact retenu E*	y*	i*	s*	ς(y)	Qté rest (y) U	V
		7.. .9																
11	+∞	0.. .8																
		9	2	1	2	0,3	0,2	1 2	- 0,2	2	1	0,2	9	2	1	+∞ 1,062	0,2	0,0
				...														

Tableau V.8 : Récapitulatif des étapes de l'algorithme 5 suivant y

L'algorithme a parcouru tous les nombres de barres filles. Finalement, la solution obtenue correspond à un coût $\varsigma\left(\sum_{1\le j\le n} m_j\right) = 1{,}062$. Comme nous avons pris un exemple simple, afin de percevoir les singularités de chacun des algorithmes, il en résulte que le placement trouvé est le même que celui de l'algorithme 1 (voir figure V.18).

3.8.4. Conclusion relative à l'algorithme 5 : barres filles

En moyenne les résultats sont légèrement meilleurs que ceux obtenus avec l'algorithme 1 avec un temps d'exécution important lorsque le nombre de barres filles est grand. Il est possible de le réduire en utilisant le calcul de la borne inférieure (cf. section 3.7).

3.9. Évaluation

Nous avons testé nos algorithmes sur 16 exemples typiques (avec 370 barres filles en moyenne) provenant de notre partenaire industriel. Les placements trouvés sont résumés dans le tableau V.9. Dans ce tableau, pour chaque exemple, nous donnons le coût de la solution obtenue en utilisant le programme déjà en place (colonne ***"part"*** qui signifie "partenaire"), le résultat donné par les algorithmes dérivés de la programmation dynamique (**DPO** : Dynamic Programming Oriented). La dernière colone donne les performances moyennes.

Notez que ces exemples sont considérés comme étant ***difficiles*** par notre partenaire. Nous voulons souligner que cette comparaison est ***défavorable*** pour nos algorithmes. Ces exemples sont reconstitués à partir des solutions réalisées par notre partenaire, lorsque les commandes sont passées en utilisant un programme déjà existant.

Par conséquent, l'ensemble des barres mères disponibles $\mathbf{B_0}$ est seulement composé de barres utilisées dans ces solutions. Ce qui signifie que nous ne connaissons pas toutes les barres disponibles au moment où la commande du client est arrivée, ce qui conduit à réduire les choix de nos algorithmes.

Exemples	Nouveau logiciel										Part.	gain avec
	Algorithme 1		Algorithme 2		Algorithme 3		Algorithme 4		Algorithme 5		Optimax	Algo 4
N°	coût (mn)	temps	coût (mn)	temps	coût (mn)	temps	coût (mn)	temps	coût (mn)	temps	coût (mn)	(%)
1	852,43	1"	***795,60***	***0"***	807,72	1"	795,60	1"	***795,60***	***0"***	827,03	3,80
2	***4069,47***	***3"***	4069,47	26"	4069,47	22"	4069,47	6"	4069,47	5"	4071,97	0,06
3	***301,39***	***0"***	301,39	1"	***301,39***	***0"***	***301,39***	***0"***	***301,39***	***0"***	301,39	0
4	529,00	12"	535,00	3'42"	487,15	11'55"	***455,03***	***11'35"***	529,00	59"	531,09	14,32
5	2809,81	42"	2809,81	4'10"	2809,81	14'26"	***2765,25***	***6'41"***	2809,81	53"	2850,81	3,00
6	1405,42	0"	1405,42	2"	1339,60	35"	***1318,95***	***9"***	1405,42	1"	1545,95	14,68
7	***246,80***	***1"***	***246,80***	***1"***	273,95	2"	***246,80***	***1"***	***246,80***	***1"***	307,95	19,85
8	***392,38***	***0"***	392,38	1"	***392,38***	***0"***	***392,38***	***0***	392,38	1"	398,44	1,52
9	***2360,49***	***2"***	2360,49	38"	2402,35	2'04"	2360,49	12"	2360,49	7"	2479,58	4,80
10	748,57	1"	721,54	17"	725,75	25"	***693,54***	***8"***	748,57	4"	988,52	29,84
11	430,54	7"	430,54	2'04"	430,54	24"	***430,54***	***10"***	430,54	16"	430,54	0
12	***807,48***	***4"***	807,48	37"	807,48	6'35"	807,48	1'41"	807,48	16"	807,48	0
13	687,25	11"	687,25	3'03"	786,67	26'49"	***656,22***	***13'37"***	687,25	38"	684,75	4,17
14	3047,29	22"	3047,29	6'42"	3022,79	43'07"	***3005,26***	***5'19"***	3047,29	1'07"	3667,76	18,06
15	1647,84	23"	1647,84	4'52"	1502,37	55'36"	***1370,84***	***23'11"***	1647,84	1'24"	1516,99	9,63
16	1370,16	34"	1370,16	4'39"	1274,36	54'25"	***1273,11***	***1°19'17"***	1370,16	1'46"	1353,39	5,93
moyenne	1356,65	10"	1351,78	1'57"	1339,61	13'33"	***1308,90***	***8'53"***	1353,09	29'	1422,73	8,00

Tableau V.9: Récapitulatif des placements obtenus à partir des algorithmes inspirés de la programmation dynamique pour les 16 exemples fournis par notre partenaire industriel

En dépit de ces points, nos algorithmes donnent des résultats satisfaisants. En moyenne, la DPO améliore de **8 %** les placements. Pour certains de nos exemples, l'amélioration est même proche de 30 %. Le temps de traitement n'est pas excessif, excepté pour l'exemple 16. Malgré des résultats moins bons que ceux fournis par l'algorithme 4, les autres algorithmes (1,2,3 et 5) améliorent de plus de 4,5 % les performances avec, en moyenne, des temps de calculs très faibles.

Ces algorithmes améliorent considérablement les performances de notre partenaire. Ils peuvent être utilisés dans différentes circonstances. Les ***algorithmes les plus rapides*** (1,2 et 5) sont plutôt destinés à être utilisés comme un outil d'***aide à la décision*** des agents commerciaux lorsqu'ils reçoivent une commande. Ils ont besoin pour cela d'une solution en temps réel et de bonne qualité. Ces algorithmes peuvent alors être mis en concurrence si le temps de traitement total n'handicape pas l'agent commercial.

L'algorithme 4, donnant le ***meilleur placement*** dans la plupart des cas, sera utilisé pour la mise en barres du lendemain. Il sera lancé de préférence le soir, lorsque le système informatique, dédié à la mise en barres, sera moins chargé.

3.10. Conclusion relative aux méthodes inspirées de la programmation dynamique

Les algorithmes présentés dans cette section sont implantés chez notre partenaire industriel depuis 1 an. Celui-ci est spécialisé dans la découpe de barres d'acier de formes variées (carrées, rondes, triangles, ...).

Les algorithmes sont utilisés de la manière suivante. Lorsque l'agent commercial reçoit une commande d'un client, il doit réaliser un plan de coupe pour établir un devis pour le client. Durant la journée, il peut y avoir plus d'un client demandant la même référence de produit. Toutes les commandes d'une même référence sont regroupées. Un plan de coupe est alors réalisé pour le lendemain par l'algorithme le plus performant durant la nuit.

La découpe se pose donc à deux niveaux : **réalisation de devis** pour les clients et **établissement d'une fiche de travail**. Pour le premier type de problèmes, le vendeur essaie d'utiliser l'algorithme 4. Si le temps de traitement excède 5 minutes pour un problème de grande taille, une variante (l'algorithme 1) de l'algorithme 4 est automatiquement lancée. La fiche de travail est réalisée durant la nuit en utilisant l'algorithme 4.

Cependant, il n'y a pas de suivi de comparaison entre les anciennes méthodes et nos algorithmes. La raison en est la suivante. Avant l'installation de nos algorithmes, un algorithme de type énumératif était utilisé pour lequel nous ne disposons pas d'autres détails. Cet algorithme est très long et donne parfois de mauvais résultats. Dans la plupart des cas, les devis de découpe pour les clients étaient réalisés manuellement par les agents commerciaux. Après

l'installation de nos algorithmes, les solutions proposées sont si satisfaisantes que les agents commerciaux les acceptent toujours. Par ailleurs, ils n'ont pas le temps d'essayer de faire eux-mêmes le placement pour chaque client afin de comparer. Il en résulte que les solutions proposées par nos algorithmes sont très satisfaisantes et qu'il est très difficile de trouver manuellement ou avec l'ancien algorithme une meilleure solution.

4. CONCLUSION

Dans cette étude, nous avons développé des algorithmes approchés pour résoudre des problèmes réels de découpe. Ces algorithmes satisfont divers problèmes réels et améliorent les performances habituelles de notre partenaire industriel.

Les méthodes par construction mises au point pour répondre à des critères ***"qualitatifs"*** (temps de coupe et temps de réglage) donnent dans la plupart des problèmes rencontrés des chutes et des nombres de tombants générés faibles. La fonction objectif n'est donc pas réellement mesurable. Lorsque tous les critères entrant dans la fonction à minimiser seront définis et ***comparables entre-eux***, notre partenaire industriel pourra utiliser les méthodes inspirées de la programmation dynamique pour améliorer encore les résultats. En effet, ces dernières autorisent une exploration plus importante de l'ensemble des solutions en ayant une vue plus "globale" que les méthodes par construction.

Chapitre VI :

CONCLUSION GÉNÉRALE

1. Description du travail
2. Directions de recherche
   2.1. Amélioration des heuristiques
   2.2. Extention des domaines d'utilisation des méthodes

CHAPITRE VI

1. DESCRIPTION DU TRAVAIL

Comme nous l'avons vu, de nombreux problèmes (non exclusivement industriels) peuvent être formulés comme des "problèmes de découpe". Nous avons présenté les principaux types d'applications.

La diversité des problèmes de découpe et les difficultés pour les résoudre ont conduit à l'élaboration de multiple heuristiques destinées chacune à la résolution de ces problèmes spécifiques. Les résultats sont alors difficilement comparables car ils sont fonction des domaines d'utilisation pour lesquels ces heuristiques sont développées.

Le développement des méthodes exactes n'est pas envisageable pour les problèmes de découpe, mais elles peuvent constituer des outils pour les heuristiques résolvant ces problèmes. Nous avons présenté les deux méthodes les plus fréquemment utilisées :

- la procédure par séparation et évaluation ;
- la programmation dynamique ou récursive.

Nous avons ensuite présenté un grand nombre d'heuristiques, développées pour résoudre les problèmes de découpe allant des méthodes utilisant des règles de priorité, le recuit simulé, les systèmes experts, les plans de coupe, aux découpes de formes complexes.

Après avoir fait l'inventaire des méthodes généralement utilisées, nous avons montré les limites de ces méthodes tant au point de vue des données :

- **barres en stock de longueurs quelconques** ;

des contraintes :

- largeur **des traits de scie** ;
- **affranchissement** de l'extrémité brute de la barre mère dans le cas d'une découpe de barres filles à tolérance réduite. Cette action n'est pas nécessaire lorsque la barre fille découpée à cette extrémité est à tolérance normale ;
- **surproduction non autorisée** ;

que du point de vue des critères :

- **préparation d'un paquet** ;
- **temps de coupe** ;
- **génération de tombants** ;
- **utilisation de tombants.**

Nous avons décomposé la résolution du problème de découpe en dimension 1 en deux étapes. La première consiste à résoudre localement le problème de placement (sous-problèmes) et la seconde utilise ces méthodes pour résoudre globalement le problème de placement. Avant d'aborder la résolution des sous-problèmes, nous avons établi la fonction coût à minimiser pour

le placement local. Elle prend uniquement en compte les coûts des chutes et des tombants, les autres coûts n'interviendront que dans la résolution du problème global. Les sous-problèmes sont de deux types :

1. placement sur une barre mère sans spécification du nombre de barres filles ;
2. placement d'un nombre précis de barres filles sur une barre mère.

Pour chacun de ces sous-problèmes, nous avons montré qu'il est possible, en tenant compte des critères énoncés, de le résoudre par une méthode exacte. Malheureusement cette méthode, bien que pseudo-polynomiale, est trop onéreuse en temps de calcul pour être utilisée un grand nombre de fois par des heuristiques de résolution du problème global.

Nous avons par conséquent développé des heuristiques :

- **Algorithme First fit decreasing** pour le premier sous-problème ;
- **Algorithme de l'arborescence limité** pour le premier sous-problème ;
- **Algorithme par permutations** pour le second sous-problème.

La résolution du problème général dépend de la connaissance des coûts entrant dans la fonction économique. Nous avons considéré deux cas :

1. les coûts ne sont pas tous évaluables ;
2. les coûts sont quantifiables et comparables entre eux.

Dans le premier cas, nous avons mis au point des méthodes par construction pour optimiser la fonction coût, basée sur les chutes et les tombants, et pour approcher des critères qualitatifs des industriels (coût de réglage, coût d'une chute suivant les tailles de paquets, coût des tombants...).

Avant de présenter les méthodes par construction résolvant le problème général, nous avons montré l'importance d'un autre critère de choix que les chutes et les tombants pour le choix des placements locaux : le rendement d'une barre.

Deux groupes de méthodes par construction ont été étudiés:

- les premières tiennent compte seulement des chutes et des tombants ;
- les autres tiennent compte en plus de la génération des paquets.

Les méthodes du premier groupe sont des extensions des méthodes développées pour résoudre le premier sous-problème :

- **Méthode FFD étendue ;**
- **Méthode de l'arborescence limitée étendue.**

Les méthodes du second groupe utilisent les méthodes de résolution du premier groupe pour construire les placements locaux :

- **Méthode locale par paquet ;**
- **Méthode globale par paquet.**

Nous avons introduit la notion de **taille fictive de paquets** afin de réduire le temps d'exécution. Elle consiste à décomposer un paquet de taille non réalisable en petits paquets de taille réalisable tout en minimisant les coûts associés : coût de préparation et coût de coupe suivant la taille du paquet. L'approche de la programmation dynamique a permis de résoudre en un temps polynomial ce problème.

Lorsque les coûts peuvent être exprimés dans une même unité de mesure, nous avons élaboré des méthodes inspirées de la programmation dynamique. La programmation dynamique n'est pas directement applicable à la résolution du problème général. Nous avons eu l'idée de réduire le nombre de combinaisons explorées en ne considérant plus toutes les combinaisons des sous-ensembles mais les valeurs des variables à considérer.

Cinq algorithmes ont été implémentés, ils procèdent par :

1. **modification des nombres de paquets et de barres mères ;**
2. **modification des nombres de paquets et de barres mères *en utilisant un algorithme de placement optimal local* ;**
3. **modification des nombres de paquets et de barres filles ;**
4. **modification des nombres de barres mères et barres filles ;**
5. **modification du nombre de barres filles.**

Les algorithmes développés, inspirés de la programmation dynamique, sont très **flexibles** en ce qui concerne les critères à prendre en compte. La plupart des variables sont déjà calculées (nombre de réglages, nombre et longueur totale des tombants, chutes, ...). Chaque nouveau critère sera appliqué au calcul de la fonction coût de chaque paquet et favorisera plus ou moins le choix des placements locaux.

Nous pensons que les améliorations significatives viennent des placements obtenus par les méthodes inspirées de la programmation dynamique. Ces méthodes nous permettent de préserver une vue globale du problème par exploration de plusieurs solutions, en contraste avec les méthodes myopes, où la solution est construite pas à pas très localement.

2. DIRECTIONS DE RECHERCHE

Les deux directions majeures de recherche que nous proposons sont d'une part d'améliorer les performances des algorithmes présentés tout au long de cette thèse et d'autre part d'étendre l'utilisation des algorithmes.

2.1. Amélioration des heuristiques

Nous avons vu tout au long de cette thèse les limites des méthodes par construction et des méthodes inspirées de la programmation dynamique. Par conséquent, il serait intéressant d'approfondir leur étude.

2.1.1. Méthodes par construction

La qualité des placements obtenus par ces méthodes diminue avec les nombres de barres à découper et de barres disponibles. Les solutions peuvent être améliorées en étudiant de près les derniers placements locaux. Nous pouvons considérer deux approches :

- Une **solution approchée** consiste non plus à ne choisir systématiquement le meilleur placement local, mais à vérifier si après le placement local considéré (pour une barre mère ou pour un paquet) toutes les commandes ont été réalisées. Dans ce cas, le coût à comparer n'est plus celui du placement local mais celui du placement global. Ce changement éviterait un mauvais choix pour le dernier placement local. Comme les comparaisons se font en plus à chacune des étapes de la construction, le temps d'exécution en sera augmenté. Il faut alors comparer sur un nombre important d'exemples si cet accroissement du temps d'exécution est acceptable par rapport à l'amélioration de la qualité des placements. En effet, le logiciel est utilisé comme un outil d'aide à la décision ;
- La **solution exacte** peut être trouvée pour les problèmes de petites tailles. Il s'agit donc ici d'utiliser les méthodes exactes de résolutions de problèmes combinatoires (cf. section 2 du chapitre II) après avoir défini précisément les limites en terme de taille du problème. De plus, en fin de placement par une méthode par construction, lorsque les restes des barres mères et des barres filles sont suffisamment réduits, le problème restant peut alors être résolu par une méthode exacte.

2.1.2. Méthodes inspirées de la programmation dynamique

Les limites des méthodes DPO sont le temps de traitement et la taille mémoire nécessaire.

- *temps de traitement*

  Les méthodes DPO sont similaires à la programmation dynamique ou aux procédures par séparation et évaluation (PSE). Les évaluations les sous-ensembles de solutions (bornes inférieure et supérieure) utilisées dans les PSE peuvent être appliquées aux sous-ensembles de solutions des méthodes DPO afin de réduire le nombre de sous-ensembles explorés ;
- *taille mémoire*

  Il s'agit d'analyser les algorithmes afin de définir avec précision le minimum d'éléments nécessaires aux procédures par récurrence quitte à recalculer entièrement le placement par la suite. Une réduction de mémoire, même minime dans une procédure par récurrence, devient conséquente lorsque le nombre d'appels de la procédure augmente.

2.2. Extention des domaines d'utilisation des méthodes

Nous avons vu que le problème de décision de découpe s'insère dans une logique d'optimisation des coûts de production. Après avoir mis en évidence d'importantes notions industrielles de découpe, il est naturel de les répercuter en amont.

La qualité d'appréciation des méthodes de prévision est liée aux contraintes et des critères pris en compte par les méthodes développées. Nous avons montré que dans la pratique, la prise en compte des contraintes et des critères que nous avons mis en évidence apportent une acuité plus grande au niveau du choix des placements.

Les méthodes développées pour la résolution du problème de placement autoriseront une recherche efficace des placements lorsque les données (stock et commande) seront fixées. Ces méthodes seront utilisées en sous-routines par les méthodes statistiques. Les problèmes amont du problème de décision de découpe sont :

a. le problème de **gestion de commandes**
   Ce problème est directement lié à la façon dont ces commandes seront réalisées. Il s'agit de choisir la meilleure combinaison de commandes (d'après les dates de livraison aux clients) à réaliser chaque jour en fonction du stock disponible ou virtuel (d'après les dates d'approvisionnement du stock).
b. le problème de **choix de longueurs standards**
   Ce problème est un problème assez particulier. Connaissant les commandes à réaliser, le gestionnaire de stock doit commander les entités mères nécessaires à leur obtention tout en minimisant un critère (voir [WOL 49] pour le choix des barres mères, et annexe de [COS 14] pour les choix de bobines). Les dimensions des entités mères sont définies dans une plage donnée. La maximisation de l'utilisation d'entités mères est parfois exprimée [MOR 43] pour diminuer le coût de la matière première commandée.
c. le problème de **gestion des réapprovisionnements**
   Après avoir défini les longueurs standards de barres en stock, il s'agit de déterminer le nombre de barres de chaque longueur à réapprovisionner en fonction de l'historique des commandes et des dates de livraison des fournisseurs.
   Wolfson [WOL 49] est peut-être le premier à l'étudier pour le problème à une dimension. Il a montré qu'en doublant le stock, les chutes en sont très réduites. Il n'a cependant pas exprimé le surcoût dû au stockage et au coût de production (plus de déplacement de stock, d'installation de paquets et d'ordonnancement).

Les méthodes inspirées de la programmation dynamique sont d'une grande efficacité dans la résolution de problèmes de placement à une dimension. Nous avons décrit le principe de ces méthodes et nous espérons que cette approche prometteuse sera appliquée à d'autres types de problèmes combinatoires.

Références

RÉFÉRENCES

[AGR 1] AGRAWAL P.K. (1993), "Minimising Trim Loss in Cutting Rectangular Blanks of a Single Size from a Rectangular Sheet using Orthogonal Guillotine Cuts", European Journal of Operational Research, vol. 64, n°4, p.410-422, Pune (Inde).

[ANT 2] ANTONIO Julien et CHU Chengbin (1996), "Approximation Algorithms to Solve Real Life Multi-criteria Cutting Stock Problems", papier proposé pour publication dans Operational Research, INRIA Lorraine (FR).

[ANT 3] ANTONIO J., CHU C., SAUER N. et WOLFF P. (1995), "POEM Project and Solution for One-dimensional Multi-criteria Cutting Stock Problems", Operations Research Proceedings (SOR'95), Passeau (AL).

[BAB 4] BABES Malika et QUILLIOT Alain (1995), "Une approche basée sur un concept logique et un formalisme mathématique, afin de résoudre un problème d'emploi du temps", RAIRO Automatique Productique Informatique Industrielle, vol. 29, n°1, p. 7-38, Abière (FR).

[BAK 5] BAKER Brenda S., COFFMAN E. G. JR. et RIVEST Ronald L. (1980), "Orthogonal Packings in Two Dimensions", SIAM, vol. 9, n°4, p. 846-855, New Jersey (US).

[BEN 6] BENHAMAMOUCH Djilali (1979), "Un Algorithme de Découpe Optimale dans R^2", Thèse de 3e cycle de l'Université Pierre et Marie CURIE, Paris 6 (FR).

[BIS 7] BISCHOFF Eberhard E. et MARRIOTT Michael D. (1990), "A Comparative Evaluation of Heuristics for Container Loading", European Journal of Operational Research, vol. 44, n°2, p. 267-276, Swansea (GB).

[CAR 8] CARLIER Jacques et CHRETIENNE Philippe (1988), "Problèmes d'ordonnancement ; modélisation/complexité/algorithme", Edition Masson, Paris Milan (FR).

[CHE 9] CHEN C.H., FEIRING B.R. et CHENG T.C.E. (1994), "The Cutting Stock Problem - A Survey", International Journal of Production Economics, vol. 36, n°3, p. 291-305, Kowloon (Hong Kong).

[CHU 10] CHU Chengbin (1990), "Nouvelles Approches Analytiques et Concept de Mémoire Artificielle pour Divers Problèmes d'Ordonnancement", Chapitre 1 : Problèmes d'ordonnancement, Thèse présentée à l'Université de Metz (FR).

[CEL 11] CLERC T. ET SADGAL M. (1987), "Prototype d'un système de placement intelligent dans le plan", Congrés MICAD 1987, p. 91-503 (FR).

[COF 12] COFFMAN E.G.Jr., GAREY M.R. et JOHNSON D.S. (1984), "Approximation Algorithms for Bin-Packing - An Updated Survey", Bell Laboratories, Murray Hill, New Jersey (US).

RÉFÉRENCES

[COS 13] COSTA Marie-Christine (1982), "Formalisation et Résolution des Problèmes de Découpes Linéaires", RAIRO, vol. 16, n°1, p. 65-82, Paris (FR).

[COS 14] COSTA Marie-Christine (1980), "Problèmes de Découpes Linéaires - Formalisation et Solutions Economiques", Thèse à l'Université Pierre et Marie CURIE, Paris 6 (FR).

[COS 15] COSTA Marie-Christine (1984), "Une Etude Pratique de Découpes de Panneaux de Bois", Recherche opérationnelle/Operations Research, vol. 18, n°3, p. 211-219, Paris (FR).

[DAR 16] DAREMA F., KIRKPATRICK S. et NORTON V.A. (1987), "Parallel Algorithm for Chip Placement by Simultated Annealing", IBM J. Res. Development, vol. 31, p. 391-402.

[DEL 17] DELAPORTE Jean-Louis (1990), "Intégration des Fonctions de Conception et de Préparation de la Fabrication pour les Entreprises de Découpe", Thèse présentée à l'Université de Valenciennes et du Hainaut-Cambresis, n°d'odre : 89-22 (FR).

[DOW 18] DOWSLAND Kathryn A. and DOWSLAND William B. (1992), "Packing Problems", European Journal of Operational Research, vol. 56, n°1, p. 2-14, Swansea (GB).

[DOW 19] DOWSLAND Kathryn A. (1993), "Some Experiments with simulated annealing techniques for Packing Problems", European Journal of Operational Research, vol. 68, n°4,1993, p. 389-399, Swansea (GB).

[DOW 20] DOWSLAND William B. (1984), "Two and Three Dimensional Packing Problems and Solution Methods", New Zealand Operational Research, vol.13, n°1, University of Swansea (BG).

[DOW 21] DOWSLAND William B. (1991), "Three-Dimensional Packing-Solution Approaches and Heuristic Development", International Journal of Production Research, vol. 29, n°8, p. 1673-1685, Swansea (GB).

[DYC 22] DYCKHOFF Harold (1981), "A New Linear Programming Approch to the Cutting Stock Problem", Operations Research, vol. 29, p. 1092-1104, Hagen (AL).

[DYC 23] DYCKHOFF Harold (1990), "A Typology of Cutting and Packing Problems", European Journal of Operational Research, vol. 44, n°2, p. 145-159, Aachen (AL).

[DYC 24] DYCKHOFF Harald et FINKE Ute (1992), "Cutting and Packing in Production and Distribution; A Typology and Bibliography", ed. Physica-verlag, A Springer-Verlag company, Aachen (AL).

[ELO 25] ELOMRI Amina (1992), "Méthodes d'Optimisation dans un Contexte Productique", thèse présentée à l'Université de Bordeaux 1, n°810 (FR).

RÉFÉRENCES

[FAU 26] FAURE Robert (1979), "Précis de Recherche Opérationelle", éditeur Dunod Décision, Paris (FR).

[GAR 27] GAREY M. R. et JOHNSON D. S. (1981), "Approximation Algorithms for Bin Packing Problems: a Survey", Analysis and Design of Algorithms in Combinatorial Optimisation, Ausiello, G & Lucertini, M. Eds, Springer, p. 149-172, Murray Hill (US).

[GAR 28] GAREY M.R. et JOHNSON D.S. (1978), "Computers and intratability : a guide to theory of NP-completeness", ed. W.H. Freemen, San Francisco (US).

[GAT 29] GATEAU Dominique, PORTMANN Marie-Claude, XIE Xiolan, BANCON Georges et TERVER Claude (1988), "Progiciel C.I.A.P. (Computer Integrated Automatic Packing) - Progiciel de Placement d'Objets Parallélépipédiques dans un Emballage Parallélépipédique - Notice de Présentation Générale de Programmation et d'Utilisation", Inria-Lorraine, Projet Sagep, Vandœuvre (FR).

[GEO 30] GEORGE John A., GEORGE Jennifer M. George et LAMAR Bruce W. (1995), "Packing different-sized circles into a rectangular container", EJOR, vol. 84, p. 693-712, Canterbury (US).

[GEO 31] GEORGE J.A. et ROBINSON D.F. (1980), "A Heuristic for Packing Boxes into a Container", COmput. & Ops Res., vol. 7, p. 147-156, University of Canterbury (Nouvelle Zealand).

[GIL 32] GILMORE P. C. et GOMORY R. E. (1961), "A Linear Programming Approach to the Cutting-Stock Problem", Operations Research, vol.9, New York (US).

[GIL 33] GILMORE P. C. et GOMORY R. E. (1963), "A Linear Programming Approach to the Cutting-Stock Problem - Part II", Operations Research, vol.11, New York (US).

[GIL 34] GILMORE P. C. et GOMORY R. E. (1965), "Multistage Cutting Stock Problems of Two and More Dimensions", Operations Research, vol. 13, p.94-120, New York (US).

[GON 35] GONDRAN M. et MINOUX M. (1979), "Graphes et Algorithmes", Collection des Etudes et Recherches, éditions Eyrolles, Paris (FR).

[GOU 36] GOULIMIS Constantine (1990), "Optimal Solutions for the Cutting Stock Problem", European Journal of Operational Research, vol. 44, n°2, p. 197-208, Londre (GB).

[HEN 37] HENRION M. (1978), "Automatic space-planning : a postmortem ?", Artificial Intelligence and Pattern Recognition in Computer Aided Design, North-holland, p. 175-191, Pittsburg (US).

RÉFÉRENCES

[HER 38] HERTZ J.C. (1972), "Recursive Computational Procedure for Two-dimensional Stock Cuting", IBM Journal of Research and Development, p.462-469, Courbevoie (FR).

[JOH 39] JOHNSON David S. (1974), "Fast Algorithms for Bin Packing", Journal of Computer and System Sciences, vol. 8, p. 272-314, Massachusetts (US)

[KIR 40] KIRKPATRICK C.D., GELLAT Jr et VECCHI M.P. (1983), "Optimisation by simulated annealing", Science, vol. 220, n°4598, p.671-680.

[LAA 41] LAARHOVEN P.M.J. et AARTS P.M.J. (1987), "Simulated Annealing : Theory and applications", D. Reidel Publishing Company, Holland.

[LIR 42] LIROV Y. (1992), "Knowledge based approach to the cutting stock problem", Mathematic Computing Modelling, vol. 16, p. 107-125.

[MOR 43] MOREAU Guy (1973), "Méthodes pour la résolution des problèmes d'optimisation de découpe", Thèse de l'Université Claude Bernard Lyon, n° d'ordre : 99 (FR).

[PER 44] PEREIRA L. M. (1978), "Artificial Intelligence Techniques in Automatic Layout Design", Artificial Intelligence and Pattern Recognition in Computer Aided Design, North-holland, p. 159-173.

[PRO 45] PROTH Jean-Marie (1992), "Annexe 1 : Le recuit simulé", Conception et Gestion des Systèmes de Production, ed. Gestion PUF, Paris (FR).

[SIA 46] SIARRY Patrick (1995), "La Méthode du Recuit Simulé : Théorie et Applications", APII., vol. 29, n°4-5, p. 535-561, Châtenay-Malabry (FR)

[SOU 47] SOUILAH Abdelghani (1994), Annexes de "Les Systèmes Cellulaires de Production : L'Agencement Inter-Cellules", thèse de l'Université de Metz (FR).

[WAT 48] WATANABE T., NAGAI Y., YASUNOBU C., LIZUKA Y. , SASAKI K. et YAMANAKA (1985), "An Expert System for Computer Room Facility Layouts", 5ième Journées Internationnales : Les systèmes experts et leurs applications, Avignon, p. 765-774 (FR).

[WOL 49] WOLFSON M.L. (1965), "Selecting the Best Lengths to Stock", Operations Research, vol. 13, p. 570-585, Pittsburg (PA).

LES PROBLEMES DE PLACEMENT :
ETUDE ET RESOLUTION DE QUELQUES PROBLEMES REELS

Résumé :

L'objectif de cette thèse consiste à caractériser les problèmes de découpe tels qu'ils se présentent dans l'industrie et d'apporter des méthodes efficaces pour résoudre ces problèmes. Tel qu'ils se présentent dans la littérature, ces problèmes consistent à positionner un sous-ensemble de barres filles sur un sous-ensemble de barres mères de plus grandes tailles de façon à minimiser le nombre de barres mères utilisées.

Cependant, pour résoudre les problèmes de notre partenaire industriel, nous avons dû prendre en compte un large éventail de contraintes et de critères. Afin de les satisfaire, nous avons développé deux groupes de méthodes :

(i) Les méthodes par construction, applicables lorsque les coûts ne sont pas mesurables. Elles réalisent le placement pas à pas en fonction des paramètres de contrôle du système fournis par les utilisateurs. Elles s'apparentent aux méthodes de traitement des problèmes avec critères qualitatifs souvent utilisées par les industriels.

(ii) Les méthodes inspirées de la programmation dynamique (Dynamic Programming Oriented methods - DPO) sont utilisées lorsque des coûts peuvent être associés aux solutions. Elles sont utilisées lorsque l'ensemble des critères à optimiser peuvent s'exprimer à l'aide d'un coût unique.

Les algorithmes développés ont été implantés avec succès chez notre partenaire industriel. Les méthodes DPO sont particulièrement efficaces (gain de 8% sur nos exemples) pour résoudre des problèmes comportant de nombreux critères.

Mots Clefs : Découpe, Placement, Optimisation.

CUTTING STOCK PROBLEMS :
STUDY AND SOLUTION OF SOME REAL LIFE PROBLEMS

Abstract :

The purpose of this thesis is to characterize real life cutting stock (or packing) problems and to provide efficient methods for a large spectrum of industrial problems. Usually, the packing problem consists in assigning a subset of output bars to a subset of input bars to minimize the number of input bars used.

Since our concern is to solve industrial problems, we have been obliged to take into account a large number of contraints and criteria. To face this complex situation, two types of approaches have been introduced, that is :

(i) Building methods, used in the case of criteria which are not precisely defined ; in this case, we propose some parameters which can influence the characteristics of the solutions. These methods are close to the way of thinking of the users.

(ii) The Dynamic Programming Oriented methods (DPO), which are put at work when the various criterion values can be combined into a unique cost.

The developed methods are set up successfully in the factories of our industrial partner. The DPO methods are particularly efficient (8% saving on ours examples) for solving problems with several real-life criteria.

Keywords : Cutting, Packing, Cargo-Loading, Container Loading, Trim-Loss.