Semaine d'Études Mathématiques et Entreprise: Étude statique et dynamique d'un problème d'empilement

Frédéric Valet, Salih Ouchtout, Marie Houillon

SEME 2018

Fives

Étude statique et dynamique d'un problème d'empilement

Encadrant universitaire :

Pr Yannick Privat

Réferent industriel :

Nicolas Watrin

Membres du groupe de recherche :

Salih Ouchtout

Frédéric Valet

Marie Houillon

24 décembre 2018

Table des matières

1 Introduction

Lors de la Semaine d'Etude Maths-Entreprise du 12 au 16 novembre 2018 organisée par l'Agence pour les Mathématiques en Interaction avec les Entreprises, la société FIVES a soumis un problème rencontré chez plusieurs de ses clients. Ce rapport illustre quelques pistes de résolution.

Cette étude proposée par Nicolas Watrin, représentant de FIVES, et encadrée par Yannick Privat, porte sur un problème d'optimisation de l'empilement de panneaux de même épaisseur. D'une part, un premier problème consiste à maximiser sur chacune des couches la densité de panneaux (problème de bin-packing) ; un second problème intervient lors de l'agencement des couches.

Ce rapport porte essentiellement sur la première question et se présente de la manière suivante. Après avoir explicité plus précisément le problème, l'approche mathématique qui en découle se révèle être compliquée à résoudre. Les algorithmes d'optimisation cherchant à déterminer des configurations optimales apparaissent très coûteux et ne permettraient pas de répondre au cahier des charges. C'est pourquoi dans une deuxième partie, plusieurs algorithmes de résolution heuristique seront proposés, tout en visant à s'approcher de l'empilement optimal. Pour conclure, certaines de ces méthodes sont implémentées, et leur efficacité est comparée sur un échantillon test. Les premiers résultats obtenus au cours de cette étude sont assez prometteurs.

2 Problématique

2.1 Problème posé par le groupe Fives

Fives, entreprise technologique dans le domaine de l'ingénierie, crée et fournit des machines, des projets et des lignes de production aux groupes industriels. En outre, elle gère des lignes de production d'entreprises d'ameublement : le problème qui nous était posé concernait une ligne d'acheminement d'une certaine quantité de plaques d'ameublement. Les produits concernés sont des plaques rectangulaires (pavé droit), dont on considérera dans toute la suite qu'elles sont de même épaisseur. Un robot guidé par logiciel prend des plaques fournies par la ligne de production, et se charge de les mettre sur une palette. L'étape suivante est donc de déplacer l'ensemble des plaques mises sur une palette d'un point de départ à un point d'arrivée. Les palettes étant chargées jusqu'à une hauteur maximale, le but est de minimiser les trajets faits par les machines, en chargeant d'une manière optimale chacune de ces palettes. Ce nombre de trajets fait par des machines peut être optimisé. Pour l'instant, les robots utilisés par Fives utilisent la méthode du shelf, détaillée plus bas : ils ne peuvent pas placer à n'importe quel endroit sur la palette les plaques en fonction de leur arrivage.

L'objectif de cette semaine est de donner des idées sur l'optimisation du chargement d'une palette, en respectant les règles suivantes :

1. Les plaques ne doivent pas dépasser les dimensions de la palette. Si la palette est de longueur L, de largeur l et de hauteur h, aucune des plaques ne doit dépasser ce pavé droit.
2. Lorsque les plaques sont posées sur la palette, elles doivent être stables. Une petite perturbation lors du placement de la plaque sur la palette ne doit pas faire bouger l'ensemble des plaques.
3. Aucune des plaques ne doit être déformée lors de la pose. Si une plaque est à moitié dans le vide, trop de poids à cet endroit peut faire courber ou casser cette plaque.
4. Une contrainte de stabilité dynamique : lors du déplacement de la palette, aucune des plaques ne doit tomber du chargement.

Il faut préciser que lors du trajet, on ne peut pas enrouler le chargement de plaques d'adhésif pour le stabiliser : après le trajet, les plaques doivent être immédiatement disponibles pour que les prochains robots puissent facilement y accéder.

Puisque toutes les plaques sont supposées de même hauteur, le problème revient à constituer des couches de plaques, que l'on superposera par la suite, et qui doivent satisfaire les contraintes précédemment citées. Une palette contiendra un nombre fini de couches de plaques.

Deux cas peuvent se produire quant à l'arrivage des plaques. Quelques fois, le format des prochaines plaques est inconnu ; d'autres fois, il y a suffisamment de plaques qui arrivent et dont les dimensions sont connues pour remplir entièrement une palette.

Grâce à l'échantillon de plaques fourni par l'entreprise, nous avons pu remarquer la fréquence de certaines plaques. Les images ci-dessous représentent les longueurs et les largeurs pour un échantillon de 500 : certaines dimensions semblent récurrentes. Le dernier graphique représente, pour un échantillon de 10000 plaques, la fréquence des dimensions des plaques. Plus une plaque apparaît souvent, plus le point la représentant est gros (il y a plus de 1000 plaques de dimension 780x575). Certaines largeurs sont assez fréquentes, telle 350, ce qui peut pousser à utiliser l'algorithme Shelf cité plus bas.

Dorénavant, les inconnues sont le placement des plaques sur chacune des couches, ainsi que l'agencement des couches pour remplir une palette. Suite à ces différentes règles, nous proposons quelques simplifications du problème qui devraient satisfaire les précédentes contraintes.

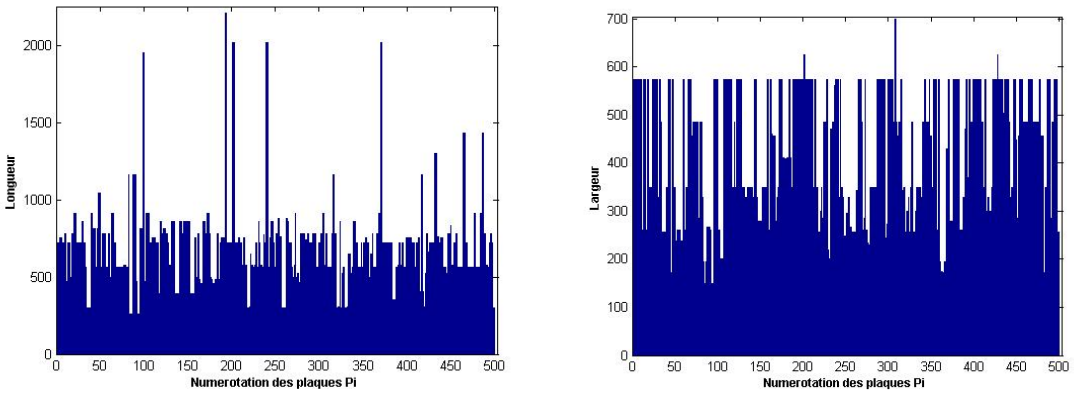

FIGURE 1 – Distribution des longueurs (à gauche) et des largeurs (à droite), pour les 500 premières plaques de l'échantillon.

FIGURE 2 – Fréquence d'apparition des plaques pour les 10000 éléments de l'échantillon

2.2 Réduction du problème

Quelques simplifications du problème peuvent permettre de satisfaire les contraintes et de le résoudre, tout du moins heuristiquement. Les contraintes précédemment citées peuvent naturellement se simplifier en fonction de la densité d'occupation de la palette. En effet, considérons la contrainte de déformation des plaques : si en dessous de chacune des plaques, le vide n'est presque pas présent, alors la plaque a très peu de chance de se déformer. De plus, si toutes les couches sont suffisamment denses, la contrainte de stabilité dynamique sera aussi satisfaite : seules les plaques avec des mouvements de liberté seront susceptibles de bouger lors du chargement, c'est-à-dire celles qui ont du vide autour d'elles. Ainsi, il apparaît pertinent de chercher à maximiser la densité sur chacune des couches.

Cet objectif de maximisation engendre un problème d'optimisation : il faut maximiser le taux de remplissage de chacune des couches. Ce problème se rapproche d'un fameux problème bien connu en algorithmique : le problème de bin packing, qui consiste à placer N objets de tailles différentes dans M corbeilles toutes identiques, et de minimiser le nombre de corbeilles utilisées.

Ce problème d'apparence simple est $NP - dur$, voir [5], c'est-à-dire que si l'hypothèse $P \neq NP$ [1] est vérifiée, ce problème ne peut pas être résolu de manière polynomiale en fonction des entrées. Ainsi, même

1. L'ensemble P est constitué des problèmes résolubles par un alogrithme en temps polynomial. Cependant, l'ensemble NP est constitué des problèmes dont, si on en propose une solution, on peut vérifier la véracité de cette solution en temps polynomial. On a l'inclusion $P \subset NP$, mais on ignore si $P = NP$: les problèmes pour lesquels on peut vérifier d'une manière polynomiale une solution n'ont pas tous un algorithme qui permet de les résoudre en temps polynomial !

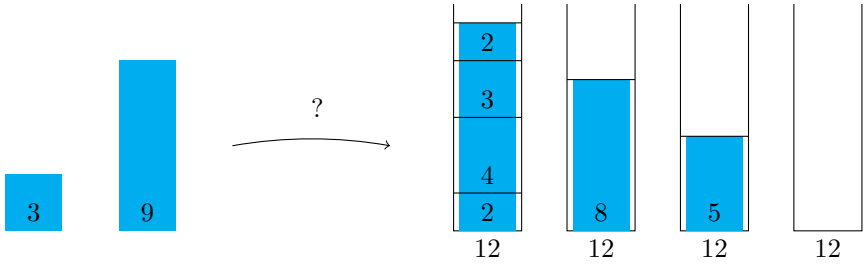

FIGURE 3 – Illustration d'un problème général de bin packing : mettre les objets de taille $\{2, 2, 3, 3, 4, 5, 8, 9\}$ dans un minimum de corbeilles de taille 12.

si des solutions optimales peuvent être proposées à ce problème, on ne peut pas garantir que le temps d'obtention de ces solutions par ordinateur soit raisonnable.

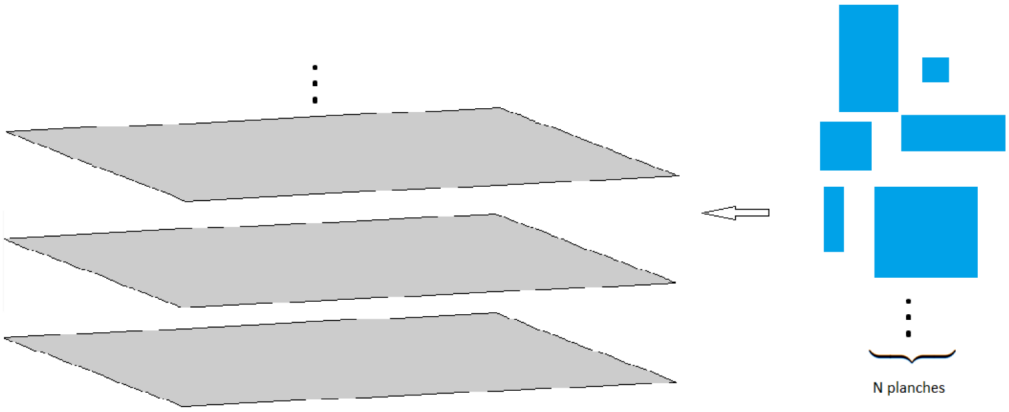

FIGURE 4 – Illustration de notre problème de bin packing

Deux méthodes de résolution de ce problème algorithmique sont proposées. Le première méthode consiste en la rédaction mathématique en terme de problème d'optimisation linéaire, détaillée ci-dessous, et qui vise à miminimiser le nombre de couches. Cette première méthode permet de déterminer une configuration optimale, mais n'est pas raisonnablement résoluble (le temps de calcul nécessaire á l'obtention des solutions étant trop élevé). Deuxièmement, une méthode algorithmique permet de combler ce manque d'efficacité par un temps de résolution raisonnable, et détermine en temps fini une configuration (peut-être pas optimale) contenant toutes les plaques. Face à la dualité entre configuration optimale et temps d'implémentation raisonnable, la seconde méthode est à privilégier.

3 Formalisation du problème sous la forme d'un problème d'optimisation

Le problème précédent se traduit sous la forme d'un problème d'optimisation linéaire, en suivant les idées de [6] . En effet, considérons que le problème consiste à placer M plaques sur un minimum de couches sur la palette de longueur L et de largeur l. Ces plaques ont pour dimension $(L_i, l_i)_{1 \leq i \leq m}$, chacune devant apparaître N_i fois. Puisque chacune de ces plaques doit être placée sur la palette, on obtient les conditions pour toute dimension i de plaques : $l_i \leq l$ et $L_i \leq L$. Cette décomposition de plaques en fonction des dimensions donne immédiatement :

$$\sum_{i=1}^{m} N_i = M.$$

Par la suite, l'ensemble des plaques sera désigné par $(P_j)_{1 \leq j \leq M}$. On définit une variable "activité" x_k de la manière suivante : sur une couche, on peut placer indépendamment de la géométrie l'ensemble de plaques $(P_{j_1^k}, \cdots, P_{j_{n_k}^k})$. Par exemple, on peut considérer que l'activité x_4 consiste à placer les plaques (P_1, P_8, P_{97}) sur une seule couche, indépendemment de la manière dont elles sont placées sur ces couches.

De cette fomulation point une première difficulté : le nombre d'activités est extrêmement grand, une estimée de borne supérieure sera donnée plus tard.

Le problème d'optimisation peut donc se formuler de la manière suivante. La contrainte est de placer,

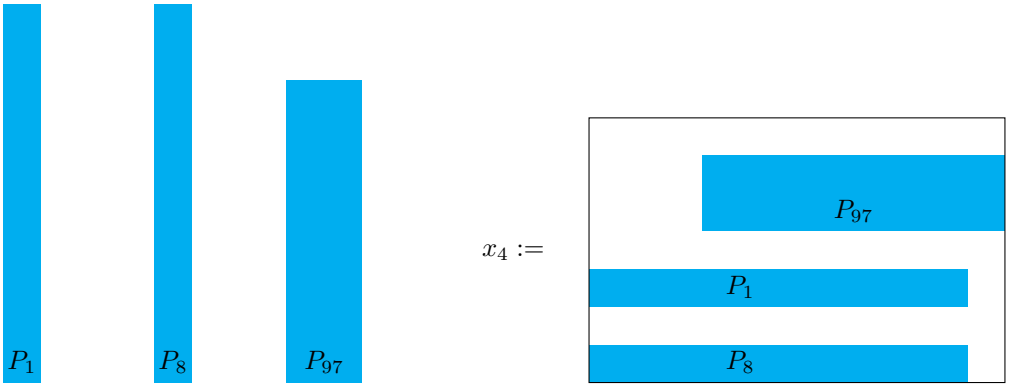

FIGURE 5 – Une interprétation de l'atcitivité x_4 : quelque soit la position des plaques P_1, P_8, P_{97} sur une couche, on considère l'activité x_4 comme une variable du problème

pour chaque format de plaques i, un nombre de plaques N_i :

$$\forall 1 \leq i \leq m \ , \ \sum_k a_{i,k} x_k \geq N_i, \tag{1}$$

où $a_{i,k}$ désigne le nombre de plaques de dimensions (L_i, l_i) placées par l'activité x_k. En reprenant l'exemple précédent, si l'on considère que les plaques P_1 et P_8 ont les dimensions (L_1, l_1), mais pas P_{97}, alors $a_{1,4} = 2$. En considérant que chaque activité x_k a un même coût, c'est à dire que le coût de placer des planches sur une seule couche ne dépend pas du nombre de planches placées sur cette couche, le but est de minimiser le nombre total de couches :

$$\sum_k c_k x_k,$$

avec $c_k = 1$ si l'activité k est considérée, et 0 sinon.

Attardons-nous sur le nombre de variables x_k. Chacune correspond au placement d'un certain nombre de plaques sur une couche, il y a donc au maximum 2^M activités possibles. À cette borne supérieure grossière, il faut ôter toutes les configurations non possibles : d'une part, à cause des dimensions limites, on ne pourra très certainement pas mettre toutes les plaques sur une seule couche ; de même, puisque les dimensions de la palette sont fixes, un grand nombre de configurations ne sera pas possible. De plus, à cause de conditions géométriques, considèrer que placer grâce aux dimensions un certain nombre de plaques sur une certaine couche ne sera pas suffisant : il faut aussi vérifier que leur emplacement sur cette même couche peut convenir, et qu'il n'y ait pas de chevauchement de deux plaques sur une même couche. Enfin, on peut remarquer que puisque les dimensions des plaques sont plus petites que celles de la palette, chacune des plaques apparaîtra dans au moins une des configurations x_k, quitte à être la seule plaque posée sur cette couche.

Le problème (1) se transforme en problème d'optimisation linéaire, en ajoutant des variables muettes ("slack variables" en anglais) $(y_1, \cdots, y_m)$, toutes positives ou nulles :

$$\forall 1 \leq i \leq m \ , \ \sum_k a_{i,k} x_k - y_i = N_i.$$

Ce problème d'optimisation linéaire n'est pas, à notre connaissance, résoluble tel quel : le coût computationnel, en particulier la durée de calcul et la quantité de mémoire nécessaire, serait trop important. Nous avons souligné le nombre très important de variables à déterminer ; nous pouvons aussi souligner que ce problème d'optimisation linéaire est un problème d'optimisation dit "combinatoire", c'est-à-dire à valeurs entières, et qu'on ne peut donc pas envisager la résolution par des méthodes utilisant la dérivée pour déterminer un minimum local.

Une dernière idée non référencée pour expliciter les contraintes géométriques est d'affecter à chaque plaque le numéro de la couche sur laquelle elle est située, puis de dire que sur chacune des couches :

— Chaque plaque située sur cette couche ne dépasse pas le bord de la palette.

— Deux plaques situées sur cette couche ne se chevauchent pas.

Ces contraintes mènent malheureusement aux mêmes difficultés que précédemment, et sans solution explicite de ce problème discret. Ce problème a été transformé de manière équivalente à une résolution sur des graphes, [1] en évoque certains aspects.

4 Méthodes heuristiques

Devant la complexité du problème mathématique d'optimisation à résoudre, la recherche de l'obtention de l'une des configurations optimales pour l'empilement des planches est chronophage. En effet, les

temps de calculs nécessaires pour y parvenir seraient extrêmement élevés, rendant inenvisageable l'obtention de la solution optimale. Fives souhaiterait en effet pouvoir calculer un empilement en moins de 5 minutes, voire en moins d'une minute si les algorithmes utilisés le permettent. Nous avons donc décidé de largement nous intéresser à la littérature relative aux méthodes heuristiques utilisées pour offrir une solution approchée, qui peut cependant être très satisfaisante au vu des résultats obtenus dans la littérature et des besoins exprimés par l'industrie.

La plupart des algorithmes utilisés pour résoudre le problème de bin-packing en deux dimensions sont des algorithmes dits "gloutons", c'est-à-dire que le choix de placement des pièces est effectué étape par étape, en essayant de minimiser localement l'espace occupé sur la palette. Typiquement, on pourra essayer de minimiser l'espace occupé sur une couche donnée plutôt que de s'intéresser à l'espace occupé par les planches sur la totalité des couches. Dans la suite de cette revue bibliographique, les termes d'"objets" désigneront les planches et ceux de "boîtes" désigneront des couches, comme on le retrouve lors de la résolution des problèmes de bin-packing.

Deux types d'algorithmes sont souvent utilisés lors du traitement de ce problème, comme dans [4] :

— des algorithmes plaçant directement les objets dans les boîtes de dimension finie ;
— des algorithmes plaçant d'abord les objets dans des boîtes de largeur fixe et de longueur infinie (appelées "bandes"), et utilisant ces bandes pour construire la solution au problème de bin-packing.

Dans ces algorithmes, plusieurs approches sont possibles pour le placement des objets dans les boîtes.

4.1 Algorithmes *shelf*

L'une des approches rencontrées le plus couramment est la configuration en "étagères", nommée *shelf algorithms* dans la littérature anglo-saxonne.
Dans cette méthode, les objets sont placés de gauche à droite, sur des rangées formant des niveaux délimités par la hauteur de l'objet le plus grand de la rangée. Ces rangées peuvent être remplies selon plusieurs stratégies :

— *Next-Fit strategy* : l'objet à placer est posé sur la dernière rangée créée, à côté du dernier objet placé, si l'espace restant le permet. Sinon, une nouvelle rangée est créée et l'objet est placé au début de la rangée.
— *First-Fit strategy* : les rangées sont explorées de la première rangée créée à la dernière, et l'objet à placer est mis sur la première rangée dans laquelle il rentre. S'il ne peut être placé sur aucune rangée existante, une nouvelle rangée est créée.
— *Best-Fit strategy* : l'objet à placer est mis sur la rangée dans laquelle il rentre et qui est jugée être la meilleure selon un critère fixé. S'il ne rentre dans aucune rangée, une nouvelle est créée. Les critères choisis peuvent par exemple être les suivants :
  — minimiser la hauteur restante au-dessus de l'objet dans la rangée ;
  — minimiser la longueur restante jusqu'au bout de la rangée ;
  — maximiser l'aire totale occupée dans la rangée ;
  — maximiser l'aire encore non occupée dans chaque rangée.

Que l'on choisisse l'algorithme plaçant d'abord les objets dans des bandes ou celui les plaçant directement dans les boîtes, les algorithmes *Next-Fit* et *First-Fit* peuvent être implémentés avec une complexité en $O(n \log n)$ [2].

Comme nous le verrons lors des tests effectués sur les différentes méthodes heuristiques, les algorithmes de type *shelf* offrent cependant des performances limitées, même avec des améliorations comme celles proposées dans [3].

4.2 Algorithmes *Guillotine*

Une approche différente du problème peut être adoptée : celle-ci est basée sur l'exploitation de rectangles non occupés (que l'on appellera *rectangles libres*) dans les boîtes.

La famille d'algorithmes que nous étudions dans cette partie rassemble les algorithmes de type *Guillotine*. Le raisonnement de base est le suivant : pour chaque boîte, on dispose d'une liste de rectangles libres disjoints. Un nouvel objet est placé dans un des coins de l'un des rectangles libres. L'espace libre dans ce rectangle est désormais une surface ayant une forme de L, et cette surface est divisée en deux rectangles, soit par une division verticale, soit par une division horizontale. On met alors la liste des rectangles libres de la boîte à jour en retirant le rectangle dans lequel a été placé l'objet, et en ajoutant les deux rectangles libres obtenus après division de la surface en L.

2. i.e. le nombre d'opérations nécessaires pour appliquer ces algorithmes à n objets est de l'ordre de $n \log n$ lorsque n devient très grand

Cet algorithme ne "gâche" pas d'espace libre dans la boîte comme le ferait un algorithme de type *shelf*, mais on peut cependant remarquer qu'il est possible qu'un objet qui rentre dans une boîte n'y soit pas placé, si l'emplacement qui correspond est à cheval sur deux rectangles libres.

Pour mettre cet algorithme en place, deux choix de stratégies doivent être effectués : tout d'abord, il faut établir un critère permettant de choisir dans quel rectangle libre un objet sera placé parmi tous les rectangles libres disponibles. Ensuite, il faut choisir selon quel axe la division du nouvel espace libre en forme de L sera effectuée.

Concernant le choix du rectangle libre à utiliser, on présente différentes possibilités rencontrées dans la littérature :

— *Best Area Fit* : le rectangle libre choisi est celui possédant la plus petite aire.
— *Best Short Side Fit* : pour chaque configuration possible, soit w_f la largeur du rectangle libre, h_f sa hauteur, w la largeur de l'objet à placer et h sa hauteur. On considère le minimum de la différence entre la longueur du rectangle libre et la longueur du côté de l'objet correspondante, dans les sens vertical et horizontal, c'est-à-dire $\min(w_f - w, h_f - h)$. La configuration retenue est celle pour laquelle cette valeur est minimale.
— *Best Long Side Fit* : avec les mêmes notations que dans la méthode précédente, on choisit la configuration qui minimise $\max(w_f - w, h_f - h)$.

Il reste à choisir l'axe selon lequel sera effectuée la division de l'espace libre en forme de L pour le transformer en deux rectangles libres disjoints. Plusieurs stratégies peuvent être envisagées, comme par exemple :

— Règle de l'axe le plus long (resp. court) : c'est la convention la plus simple, selon laquelle la division est effectuée horizontalement si $w_f < h_f$ (resp. $w_f \geq h_f$) et verticalement sinon.
— Règle de la longueur restante la plus courte (resp. la plus longue) : avec les mêmes notations que précédemment, on choisit de diviser horizontalement si $w_f - w < h_f - h$ (resp. $w_f - w \geq h_f - h$ et verticalement sinon.

Le principal inconvénient des algorithmes de type *Guillotine* est qu'il est possible d'avoir à placer des objets qui pourraient rentrer dans une boîte donnée, mais qui seront en pratique placés dans une autre boîte car il ne rentrent dans aucun des rectangles libres. Cette problématique mène à l'introduction d'un autre type de méthodes de placement appelée méthode des rectangles maximaux, qui utilise justement une liste de rectangles libres non disjoints, permettant d'exploiter les rectangles libres d'aire maximale dans la boîte.

4.3 Algorithmes des rectangles maximaux

L'algorithme proposé maintenant se base également sur une liste de rectangles libres associée à chaque boîte. La différence majeure entre cette liste et celle utilisée dans les algorithmes *Guillotine* est qu'ici, les rectangles libres ne sont pas disjoints.

En effet, lorsqu'un nouvel objet est placé dans un rectangle libre, réduisant l'espace libre dans ce rectangle à une surface en forme de L, on ne crée plus deux rectangles disjoints : les deux rectangles libres créés sont respectivement le rectangle de longueur maximale pouvant être placé dans la surface en forme de L, et le rectangle de largeur maximale.

Le principal avantage de cette méthode est qu'elle assure que si un objet rentre dans une boîte, il y sera placé par l'algorithme.

Le fait que les rectangles libres ne soient pas disjoints deux à deux demande un traitement supplémentaire lors de la mise à jour de la liste des rectangles libres : il est tout d'abord nécessaire de modifier les rectangles libres dont l'intersection avec l'objet placé est non nulle. Après ce traitement, il est possible que certains rectangles libres soient contenus dans d'autres rectangles libres ; ces rectangles doivent être supprimés. Pour obtenir l'algorithme complet, nous renvoyons à [3].

Concernant le choix du rectangle libre dans lequel un objet sera placé, on peut utiliser les mêmes règles que pour les algorithmes de type *Guillotine*.

Une variante propre à l'algorithme des rectangles maximaux, appelée méthode *Bottom Left* ou *Tetris*, peut aussi être utilisée. La règle de placement d'un objet dans une boîte est la suivante : l'objet doit être placé et orienté de façon à ce que l'ordonnée de son arête supérieure soit minimale. Si plusieurs positions sont possibles, on choisit celle pour laquelle l'abscisse du coin inférieur gauche de l'objet est minimale.

4.4 Méthodes d'amélioration générales

Les méthodes présentées précédemment ont pour objectif de placer, dans la première boîte disponible, un objet quelconque. En particulier, on ne choisit pas l'ordre dans lequel les objets vont être placés dans les boîtes. Il est possible, dans certains cas d'usage, que ceci soit une contrainte pratique, si on n'a aucune visibilité sur les prochains objets que l'on devra placer et qu'on remplit les boîtes au fur et à mesure de l'arrivée des objets. En revanche, dans le problème exposé par Fives, on a connaissance de tous les objets

qui devront être empilés, et on peut donc proposer des améliorations dans lesquelles une sélection des objets à placer peut être effectuée.

Nous allons donc exposer plusieurs méthodes visant à améliorer la densité de l'empilement global, indépendamment de l'algorithme de placement utilisé.

- *Choisir la boîte de destination* : plutôt que de placer un objet dans la première boîte qui peut le contenir, on peut décider de le placer dans la dernière boîte ayant été créée ou encore dans la boîte offrant la meilleure configuration possible, selon un certain critère établi par la méthode de placement choisie.
- *Trier la liste des objets à placer* : plutôt que de placer les objets dans les boîtes dans un ordre aléatoire, on peut décider de trier la liste des objets. On peut par exemple décider de trier les objets (de manière croissante ou décroissante) selon leur surface, leur largeur, leur longueur, leur périmètre ou la différence entre leur largeur et leur longueur.
- *Réaliser un choix global* : dans cette méthode, au lieu de placer le prochain objet dans la liste des objets à placer, on essaye de placer tous les objets restants, et on choisit de placer celui qui offre la meilleure configuration possible, selon un certain critère établi par la méthode de placement choisie. Il est donc inutile de trier les objets, puisqu'ils sont tous inspectés à chaque nouveau placement.

5 Proposition d'algorithmes et implémentations

5.1 Pistes proposées

De part les méthodes heuristiques présentées dans la littérature et résumées ci-dessus, l'algorithme le plus pertinent pour le placement des objets paraît être l'algorithme des rectangles maximaux : dans les résultats présentés par [3], il apparaît clairement que cette méthode est supérieure en termes d'optimisation de l'espace occupé dans les boîtes aux méthodes de type *shelf*. Puisque les méthodes de rectangles maximaux sont une amélioration des méthodes de type *Guillotine*, seules les méthodes de rectangles maximaux ont été implémentées.

Ces constats nous amènent à tester en priorité plusieurs des variantes de l'algorithme des rectangles maximaux, ainsi que quelques variantes de l'algorithme *shelf*, afin d'avoir un point de comparaison quant au gain obtenu avec l'algorithme des rectangles maximaux. La démarche adoptée dans les algorithmes *shelf* s'apparente en effet fortement aux démarches qui sont utilisées actuellement pour réaliser les empilements de planches sur les palettes au sein des entreprises avec lesquelles Fives est en collaboration. Il semble donc que Fives puisse avoir un intérêt à proposer des résultats avec les deux approches, afin d'illustrer le gain que peut offrir une nouvelle méthode d'empilement à leur client.

Les méthodes précédemment citées et leurs améliorations globales sont aussi implémentées. En particulier, le cas où les pièces à empiler sont triées de la plus grande à la plus petite est pertinent à traiter : dans les cas-tests effectués dans la littérature, cette stratégie s'avère souvent être la plus efficace.

5.2 Tests effectués

Ayant accès à l'implémentation des méthodes des rectangles maximaux et de types *shelf* [2], nous avons testé plusieurs des différentes méthodes heuristiques présentées dans la littérature.
En raison du temps limité dont nous disposions pour effectuer les tests, des contraintes ont du être prises :

- seules des stratégies de type "First Bin" sont utilisées, c'est-à-dire que les objets sont systématiquement placés dans la première boîte dans laquelle ils peuvent l'être.
- Nous supposons, comme c'est indiqué dans la problématique, que la liste des objets à placer est entièrement connue.
- Les objets à placer sont les 500 premières planches dans l'échantillon-test qui nous a été fourni par Fives.
- La palette sur laquelle l'empilement est effectué est de dimension $1.5m \times 2.5m$.

En sortie, l'algorithme donne la liste des couches créées ainsi que le format et le l'emplacement des planches composant chaque couche.
L'algorithme rend aussi les données pertinentes pour la répartition exhaustive des plaques sur chacune des couches, et des données permettant de comparer les différentes méthodes :

- Le nombre de couches qui composent l'empilement ;
- La densité moyenne des couches de l'empilement, définie par la moyenne sur toutes les couches du rapport de la surface occupée par les planches et de la surface totale d'une couche.
- L'écart-type de la densité sur les couches.

— Le temps de calcul nécessaire à la réalisation de l'empilement.

L'objectif est de minimiser le nombre de couches, ce qui est équivalent à maximiser la densité des couches. La valeur de la densité ainsi que son écart-type nous donnent de plus une indication à propos de l'aire occupée par les planches sur chaque couche. Une densité trop faible augmente le risque que certaines planches soit placée sur des "trous", ce qui peut provoquer leur déformation. De plus, une forte densité d'occupation des couches contribue à améliorer la stabilité de l'empilement.

Les résultats obtenus peuvent être observés sur la figure 6. Les dénominations utilisées sont les suivantes :

- Pour l'algorithme de placement :
  - *maxrect* : algorithme des rectangles maximaux
  - *shelf* : algorithme de type shelf
- Pour le critère de choix de l'emplacement de la planche à placer :
  - *bssf* : Best Short Side Fit
  - *blsf* : Best Long Side Fit
  - *blr* : Bottom Left Rule
  - *bal* : Best Area Fit
  - *blr* : Best Width Fit
  - *bal* : First Fit
- Pour la méthode de choix de la prochaine planche à placer :
  - Online : les planches sont prises dans l'ordre de l'échantillon, sans tri préalable.
  - Tri surface : les planches sont triées par surface décroissante.
  - Tri larg long : les planches sont triées par largeur décroissante, puis par longueur décroissante.
  - Tri long larg : les planches sont triées par longueur décroissante, puis par largeur décroissante.
  - Batch : la prochaine planche à placer est celle qui offre la meilleure configuration possible selon le critère de choix d'emplacement utilisé.

	Densité moyenne	Ecart-type de la densité	Nombre de couches	Temps d'exécution (s)
Online				
$maxrect - bssf$	0.878977	0.0621116	45	0.00943959
$maxrect - blsf$	0.84157400000000004	0.134997	47	0.00945027
$maxrect - baf$	0.859869	0.110083	46	0.00923009
$maxrect - blr$	0.859869	0.14682	46	0.00860567
Tri surface				
$maxrect - bssf$	0.91986	0.0591491	43	0.0043101
$maxrect - blsf$	0.898954	0.117761	44	0.00415113
$maxrect - baf$	0.91986	0.0439613	43	0.00502864
$maxrect - blr$	0.941761	0.0527853	42	0.00336981
Tri larg long				
$maxrect - bssf$	0.91986	0.0839841	43	0.0033139
$maxrect - blsf$	0.91986	0.0859056	43	0.00410095
$maxrect - baf$	0.91986	0.0561876	43	0.0561876
$maxrect - blr$	0.941761	0.0951748	42	0.00376862
$shelf - bwf$	0.859869	0.153419	46	0.00154953
$shelf - ff$	0.859869	0.153419	46	0.00143136
Tri long larg				
$maxrect - bssf$	0.91986	0.0523706	43	0.00439116
$maxrect - blsf$	0.878977	0.134063	45	0.00410573
$maxrect - baf$	0.898954	0.129123	44	0.00374735
$maxrect - blr$	0.91986	0.120553	43	0.00290699
batch				
$maxrect - bssf$	0.91986	0.169591	43	0.0153466
$maxrect - blsf$	0.878977	0.14521	45	0.024346
$maxrect - baf$	0.898954	0.141206	44	0.0182033
$maxrect - blr$	0.732481	0.136378	54	0.0145794

FIGURE 6 – Résultats obtenus lors des tests des différents algorithmes pour notre cas-test.

Comme on peut le voir, l'algorithme qui donne le meilleur résultat pour notre cas-test est l'algorithme des rectangles maximaux associé à la méthode *Bottom Left*, lorsque les rectangles sont pris du plus grand au plus petit en terme de surface. On peut noter que ce n'est pas la configuration optimale dans le cadre

de tests menés dans [3], dans lesquels les rectangles à placer étaient choisis de façon aléatoire. Il peut donc être intéressant, pour un problème donné, de tester plusieurs méthodes afin de pouvoir choisir celle qui sera la plus adaptée au jeu de données considéré.
En utilisant cette combinaison de méthodes, on obtient notamment une densité de l'empilement égale à 94%, et un écart-type de 5%, ce qui correspond à un nombre de couches égal à 42. On rappelle que la densité d'une couche est définie comme le rapport entre l'aire occupée par les planches et l'aire totale disponible sur la couche. Ce résultat de densité paraît très satisfaisant au vu du problème posé. Il reste à définir si cet empilement est bien stable, car si nos algorithmes visent à maximiser la densité des couches afin de garantir leur stabilité, ils n'assurent pas la stabilité des couches ni la non-déformation des pièces. En empilant les couches dans l'ordre où elles sont créées, on ne peut qu'assurer qu'aucune pièce ne se retrouvera dans le "vide". Sinon, en effet, elle aurait été placée dans la couche précédente. En revanche, une pièce donnée peut tout à fait n'avoir qu'une petite partie de sa surface en contact avec la couche précédente.
Une illustration de quelques couches créées par la méthode du tri par aire décroissante et le placement par rectangles maximaux est proposée sur la figure 7.

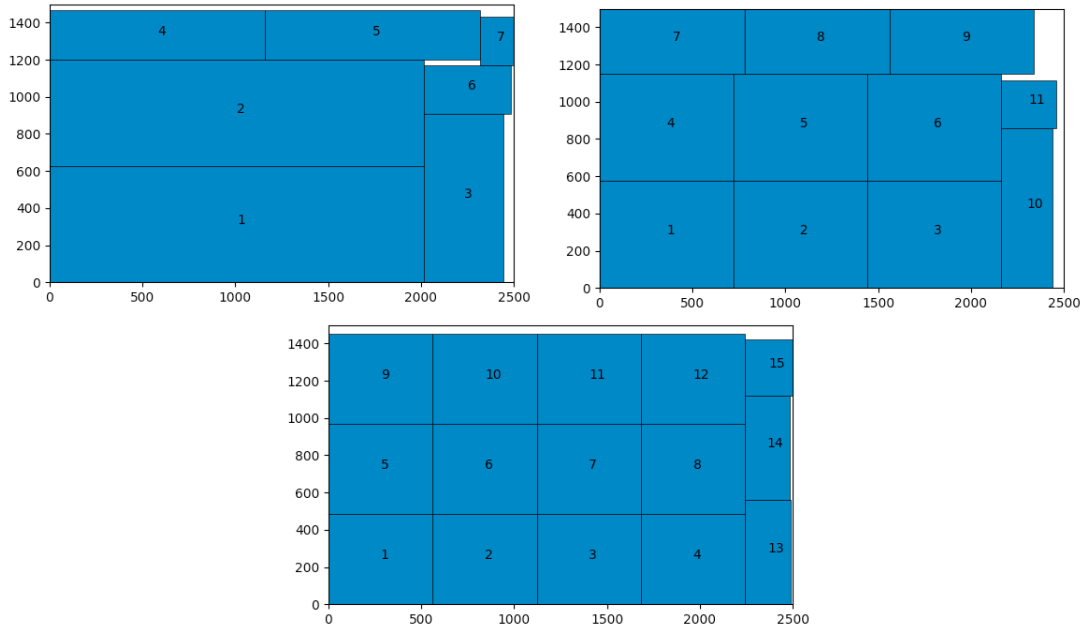

FIGURE 7 – Couches 1, 20 et 35 sur 42 créées au total par l'algorithme des rectangles maximaux et le tri par surface décroissante.

Sur la figure 8, on peut voir l'empilement 3D composé de 42 couches généré par notre algorithme.

Il est intéressant de noter que les temps de calcul nécessaires à la construction des empilements sont négligeables, et ce peu importe la méthode utilisée.

6 Conclusions

Pour rappel, l'objectif du travail demandé était de déterminer des méthodes permettant de ranger des plaques rectangulaires de tailles variables dans des containers de taille fixée, en maximisant dans chaque container la surface occupée par les plaques.

La première étape consistait à mettre ce problème sous la forme d'un problème d'optimisation linéaire, mais aucune solution qui puisse être calculée en temps raisonnable ne peut être fournie, puisqu'il s'agit d'un problème NP-dur.

En revanche, nous avons pu trouver dans la littérature de nombreuses méthodes heuristiques permettant de résoudre de façon approchée ce problème dit de "bin-packing", et certaines d'entre elles offrent des résultats prometteurs quand à la densité de l'empilement obtenu. En particulier, la méthode des rectangles maximaux semble très bien convenir à la résolution du problème, notamment lorsqu'on la combine avec un tri préalable des planches de la plus grande (en termes de surface) à la plus petite.

L'inconvénient de ces méthodes, bien qu'offrant des résultats satisfaisants quant à la densité des couches obtenues, est qu'elles ne permettent toutefois pas d'assurer que l'empilement produit par l'algorithme est stable, ni qu'aucune plaque ne sera déformée. Ce problème devra être traité, par exemple en réarrangeant l'organisation des couches ou en ajoutant des critères physiques d'acceptabilité pour les empilements.

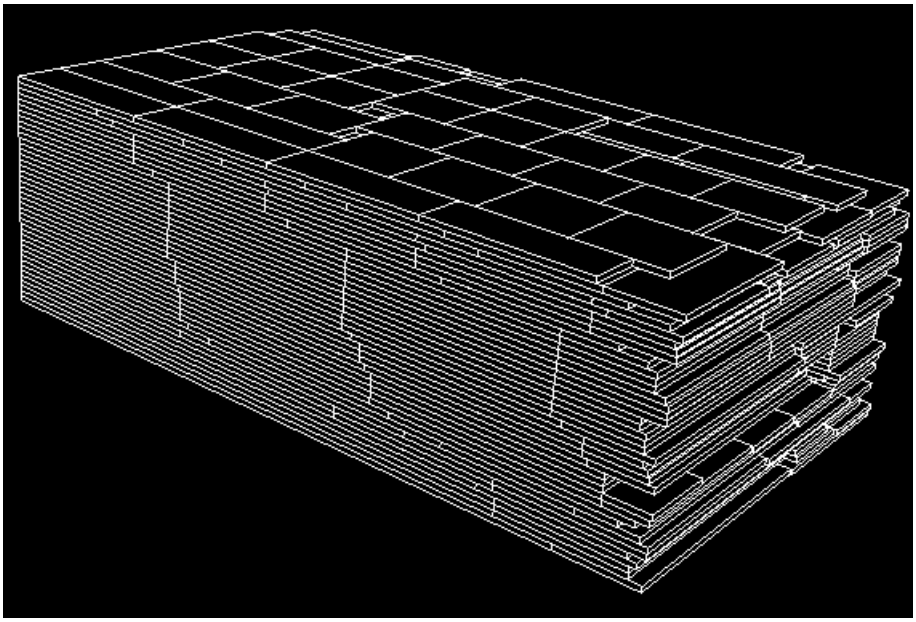

FIGURE 8 – Illustration de l'empilement de 42 couches formées de 500 planches

Un point crucial dans ce manuscrit est que les temps de calcul nécessaires pour mettre en oeuvre les algorithmes que nous avons présentés sont négligeables, dans le cadre de la problématique proposée par Fives, où le nombre de pièces à empiler est d'environ 500. Dans ce cadre, il peut paraître judicieux de ne pas se tenir à une seule méthode de tri ou d'empilement, mais de générer plusieurs empilements (un par algorithme) et de choisir le plus satisfaisant parmi tous ceux obtenus. D'autre part, toutes les méthodes implémentées ci-dessus consistent à choisir la "meilleure place" pour la prochaine planche à placer. Il pourrait être intéressant, plutôt que de prendre les planches une par une, de tester toutes les configurations possibles pour les n prochaines planches à placer, et de choisir la configuration la plus satisfaisante.

Références

[1] Joseph El Hayek. *Le problème de bin-packing en deux-dimensions, le cas non-orienteé : résolution approchée et bornes inferieures.* PhD thesis, Universite de technologie de Compiegne, 2006.

[2] Jukka Jylanki. C++ code on bin packing methods, https ://github.com/juj/rectanglebinpack, 2010.

[3] Jukka Jylanki. A thousand ways to pack the bin – a practical approach to two-dimensional rectangle bin packing, 2010.

[4] Andrea Lodi. Algorithms for two-dimensional bin packing and assignment problems. *Doktorarbeit, DEIS, Universita di Bologna*, 16, 1999.

[5] David Johnson Michael Garey. *Computers and Intractabillity.* W. H. Freeman and Company, 1979.

[6] R.E. Gomory P.C. Gilmore. A linear programming approach to the cutting-stock problem. *Operations Research*, 9(6) :849–859, Nov-dec 1961.