Université Toulouse III - Paul Sabatier

Mémoire de fin d'études

Master : **Mathématiques et applications**
Option : **Recherche Opérationnelle et Optimisation**

Optimisation du bin packing 2D dans les camions et conteneurs

Auteur :
Stephane.D METSANOU TSAFACK

Entreprise :
Groupe Renault

Superviseur universitaire :
Cyril ALLIGNOL

Superviseur entreprise :
Martin LAINEE

2023-2024

Résumé

Le problème de bin packing en deux dimensions représente un défi majeur pour la gestion logistique et la chaîne d'approvisionnement chez Renault, où l'optimisation de l'espace de chargement des camions et conteneurs est essentielle. Avec l'augmentation des volumes de production et d'articles à stocker et à expédier, l'optimisation de l'algorithme interne est devenue d'autant plus cruciale pour minimiser le gaspillage d'espace et réduire les coûts logistiques. Dans ce contexte, nous avons développé une méthode heuristique basée sur un algorithme de recherche arborescente pour optimiser le chargement des camions et conteneurs. Les expériences numériques montrent que cette méthode offre des performances notables et se révèle compétitive par rapport à l'algorithme interne utilisé pour traiter ce problème. Ces résultats mettent en lumière le potentiel de notre approche pour améliorer le processus logistique chez Renault, et des pistes pour des améliorations futures sont également explorées.

Mots clés : Optimisation, Chargement de camions et conteneurs 2D, Heuristique, Logistique Industrielle.

Remerciements

Je souhaite tout d'abord exprimer ma profonde gratitude à mon tuteur de stage, Monsieur Martin LAINEE, pour son encadrement précieux. Sa disponibilité constante et ses conseils avisés ont été essentiels dans ma découverte et ma pratique de la recherche opérationnelle. Grâce à son expertise et à sa pédagogie, j'ai pu mener à bien ce travail dans un environnement exigeant et stimulant.

Je remercie également Madame Siham ESSODAIGUI pour son soutien indéfectible, sa pratique managériale exemplaire et sa compréhension, qui ont grandement contribué à la réussite de ce stage. Son approche humaine et son souci du détail ont enrichi mon épanouissement professionnel et personnel pendant cette période.

Mes remerciements vont aussi à Monsieur Cyril ALLIGNOL, mon tuteur universitaire, pour son assistance précieuse. Ses orientations ont été d'une grande aide dans la rédaction de ce rapport, et son expertise académique m'a permis de structurer et valoriser les résultats de mon travail de manière optimale.

Je n'oublie pas d'exprimer ma reconnaissance envers Messieurs Florian FONTAN, Mohamed-Amine KHATOUF, Alain NGUYEN, et Madame Catherine Mancel pour leur soutien précieux tout au long de ce stage.

Un merci particulier à l'équipe d'Intelligence Artificielle Appliquée (IAA) pour leur passion, leur bonne humeur communicative et leur humour, qui ont rendu cette expérience non seulement enrichissante mais aussi particulièrement agréable.

Enfin, je souhaite exprimer ma gratitude à ma famille et mes amis.

Table des matières

Liste des tableaux

Table des figures

Introduction

Le développement considérable des chaînes d'approvisionnement pour le transport de pièces de véhicules des différents fournisseurs vers les usines de production a conduit à une augmentation significative des volumes de marchandises transportées. En conséquence, des entreprises telles que le Groupe Renault (RG) [24], une multinationale de l'automobile, cherchent constamment à optimiser leur chaîne logistique pour le transport de ces pièces. Avec ses 40 usines et 1500 fournisseurs répartis dans plus de 17 pays, RG expédie chaque semaine plus de 6000 camions entre ses fournisseurs et ses usines, pour un coût annuel de plusieurs centaines de millions d'euros. Dans le but d'optimiser sa chaîne d'approvisionnement, RG a développé en interne un outil appelé Optim3D, qui, bien que performant, nécessite des améliorations, notamment en termes de temps de réponse. Pour adresser ce besoin, RG, en partenariat avec la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF) [1], a proposé cette problématique d'optimisation comme sujet du Challenge ROADEF 2022 [25]. Ce problème d'optimisation présente plusieurs défis bien connus en recherche opérationnelle. Tout d'abord, la définition de la route optimale que doit emprunter chaque camion en respectant les contraintes de temps de livraison constitue un problème de tournée de véhicules (VRP). Ensuite, la division des articles en piles optimales et leur affectation aux camions représente un problème classique de découpe et d'affectation. Enfin, le chargement des piles dans l'espace de chargement des camions, tout en respectant les contraintes de dimensions physiques et de poids des articles, est un problème de bin-packing (BP). De nombreuses contraintes supplémentaires telles que les contraintes de charge aux essieux des camions et l'orientation des piles compliquent encore davantage ce dernier problème. À l'issue du challenge, certaines méthodes proposées par les candidats se sont avérées intéressantes, et RG a décidé de s'en inspirer pour optimiser son outil interne, en particulier pour améliorer le taux de remplissage des camions. Par conséquent, l'objectif de ce projet de recherche est d'étudier et d'explorer diverses méthodes proposées par les candidats du challenge et de s'en inspirer pour concevoir un algorithme capable de fournir de meilleures solutions pour optimiser le remplissage de la flotte de camions. Ce travail s'appuie principalement sur les méthodes proposées pendant le challenge, en se concentrant particulièrement sur celle proposée par le candidat vainqueur [11]. Les principales contributions de cette étude sont les suivantes : tout d'abord, une revue exhaustive de la littérature sur le BP est établie. Ensuite, le BP est formulé et une méthode de résolution est proposée pour l'aborder. Enfin, diverses expériences numériques sont réalisées pour évaluer et comparer les performances entre l'approche proposée et l'outil interne.

Environnement de stage

Notre stage se déroule au sein de l'équipe de recherche opérationnelle, également appelée "Optimisation Combinatoire", chez RG. Cette équipe fait partie intégrante de l'équipe plus large de l'Intelligence Artificielle Appliquée (IAA). L'IAA regroupe plusieurs domaines spécialisés, notamment la configuration produit (modélisation et manipulation de la gamme véhicule), la recherche opérationnelle (optimisation, outils d'aide à la décision) et le traitement du langage naturel (apprentissage automatique de données textuelles). La mission principale de cette équipe est de concevoir, développer et maintenir des applications et des composants informatiques pour améliorer les processus internes de RG. Les stagiaires participent régulièrement à des réunions d'équipe, des séminaires et des ateliers, où ils peuvent présenter leurs progrès, discuter de leurs idées et recevoir des retours constructifs. Ce cadre encourageant et stimulant est conçu pour favoriser l'apprentissage et le développement professionnel des stagiaires, tout en contribuant aux objectifs de l'équipe de recherche opérationnelle de RG.

Objectif du stage

L'objectif principal du stage est de développer un algorithme pour optimiser le remplissage des camions dans le cadre de la chaîne d'approvisionnement de RG. Cela implique une revue approfondie de la littérature sur les problèmes de BP, ainsi que l'identification de méthodes pouvant servir de base pour de nouvelles approches. Sur cette base, une méthode sera développée, implémentée et évaluée. Ce travail vise à proposer un algorithme capable de remplacer l'algorithme interne de bin-packing à deux dimensions (BP2D) dans les cas où il s'avérerait plus efficace.

Organisation du rapport

Le rapport est organisé comme suit. Le Chapitre 1 présente une définition générale de notre problème, un examen de la littérature existante, ainsi qu'un aperçu du problème de bin packing en deux dimensions. Dans le Chapitre 2, nous proposons une modélisation mathématique de notre problème. Le Chapitre 3 introduit notre approche de résolution du problème, qui repose sur une méthode heuristique basée sur un algorithme de recherche arborescente. Le Chapitre 4 détaille les expériences numériques effectuées pour comparer les performances de l'approche proposée avec celles de l'algorithme interne. Enfin, dans le chapitre de conclusion, les principaux enseignements sont résumés et des pistes de recherche futures sont suggérées.

Chapitre 1

Définition du problème et revue de littérature

Comme mentionné dans l'introduction, le développement considérable des chaînes d'approvisionnement pour le transport de pièces de véhicules des différents fournisseurs vers les usines de production a conduit à une augmentation significative des volumes de marchandises transportées. En conséquence, RG cherche constamment à optimiser sa chaîne logistique pour le transport de ses pièces. Dans ce travail, nous nous concentrons sur l'optimisation du chargement de camions. Dans ce qui suit, nous décrivons en détail le problème dans son ensemble et passons en revue la littérature existante sur ce sujet.

1.1 Définition du problème

Ce problème de chargement de camions et conteneurs implique le transport d'un vaste nombre d'articles des fournisseurs vers les usines en utilisant des camions. Les fournisseurs produisent des articles qui peuvent être chargés dans les camions à partir des quais. Chaque fournisseur peut avoir plusieurs quais et produire différents types d'articles. De manière similaire, chaque usine peut avoir plusieurs quais où les articles peuvent être déchargés des camions. Un ensemble prédéfini d'articles doit être transporté d'un quai de fournisseur spécifique à un quai d'usine spécifique. La figure 1.1 illustre un exemple de trajets des camions où le camion A s'arrête chez le Fournisseur A, où il visite à la fois les quais jaune et vert, puis il se dirige vers le Fournisseur B, s'arrêtant au quai bleu, avant de se rendre chez le Fournisseur C, où il s'arrête aux quais bordeaux et cyan. Enfin, il arrive à l'usine Y et s'arrête aux quais vert et jaune. Le camion B voyage depuis le Fournisseur A, s'arrêtant aux quais jaune et vert, avant de se rendre chez le Fournisseur B, où il s'arrête uniquement au quai orange. Ensuite, il arrive à l'usine Y et s'arrête aux quais vert et jaune. Le camion C se rend chez le Fournisseur C et s'arrête aux trois quais avant de se rendre à l'usine Y, où il visite également tous les quais disponibles.

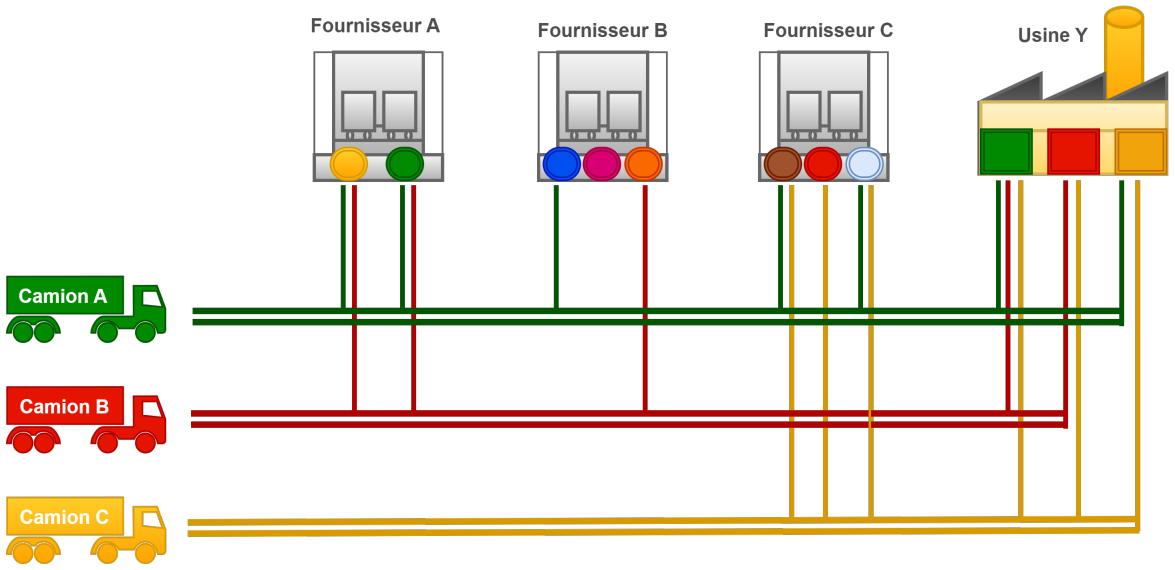


FIGURE 1.1 – Exemple de trajets des camions.

Chaque article doit être livré dans une fenêtre de temps déterminée, comprenant une date d'arrivée au plus tôt et une date d'arrivée au plus tard. Si un article est livré avant sa date d'arrivée au plutôt, des frais d'inventaire sont encourus pour chaque jour avant cette date. Il en est de même pour un article livré après sa date d'arrivée au plus tard. Tous les articles chargés dans un camion doivent être empilés, et chaque pile peut contenir un ou plusieurs articles. Diverses contraintes régissent la manière dont les articles peuvent être empilés, telles que des restrictions sur les dimensions, le nombre maximal d'articles empilés, et la densité maximale d'une pile.

Une flotte de camions planifiée est disponible. Chaque camion peut visiter plusieurs fournisseurs prédéfinis et charger des piles à partir de leurs quais respectifs. Ensuite, le camion se rend à une usine spécifique où il peut décharger les piles sur différents quais. L'ordre des visites chez des fournisseurs et des quais est préétabli, et les piles doivent être chargées dans les camions dans le même ordre. Chaque camion a une liste d'articles qu'il peut collecter. Des contraintes de placement des piles et de poids maximum par camion sont également prises en considération. Chaque camion planifié a un coût associé, facturé uniquement s'il est utilisé.

L'objectif principal est de livrer tous les articles et de minimiser les coûts totaux. Une solution inclut la définition de toutes les piles, leur attribution et chargement à des camions spécifiques, la connaissances des coordonnées tridimensionnelles de chaque article et bidimensionnelles de chaque pile dans le camion, ainsi que la liste des camions utilisés avec leurs caractéristiques de chargement.

Dans le cadre de notre travail, nous concentrerons notre attention sur le défi de l'optimisation du chargement des camions, plus précisément sur le BP2D, dans le but de maximiser leur capacité de chargement en utilisant les différentes piles définies au préalable.

Les problèmes de type BP, initialement étudiés dès les années 1950 dans un contexte unidimensionnel (BP1D), ont vu leur complexité évoluer au fil du temps. De nos jours, la littérature traite de problèmes en deux dimensions (BP2D), voire en trois dimensions (BP3D).

Le problème de BP1D est un problème d'optimisation combinatoire défini comme suit : étant donné un nombre illimité de conteneurs (bins) avec une capacité fixe c et un ensemble de n objets, chacun avec un poids spécifique w_i, BP1D vise à trouver le nombre minimum de conteneurs nécessaires pour emballer tous les objets sans dépasser la capacité de chaque conteneur. formellement, étant donné un ensemble $N = \{1, \ldots, n\}$ d'objets, un ensemble de conteneurs avec la même capacité $c > 0$, et w_i étant le poids respectif de chaque objet $i \in N$ avec $0 < w_i \leq c$. BP1D consiste à regrouper tous les éléments dans des conteneurs de manière à ce que la somme de leurs poids ne dépasse pas c, et que le nombre de conteneurs m utilisé soit minimal [5]. En d'autres termes, nous voulons créer une partition de N en un nombre minimal de sous-ensembles B_j possible :

$$\bigcup_{j=1}^{m} B_j = N \tag{1.1}$$

tel que :

$$\sum_{i \in B_j} w_i \leq c, \quad 1 \leq j \leq m, \quad i \in N \tag{1.2}$$

Le problème BP2D consiste en un ensemble de n articles rectangulaires, chacun ayant une hauteur H_i et une largeur W_i, pour $i = 1, \ldots, n$. L'objectif est de les emballer dans un nombre minimal de conteneurs rectangulaires, chacun ayant une hauteur H et une largeur W. Les articles ne doivent pas se chevaucher et nous ne considérons pas la rotation. Une formalisation simple de ce problème est la suivante :

$$\text{Minimiser } m \tag{1.3}$$

$$\sum_{j \in J} x_{ij} = 1, \quad \forall i \in I \tag{1.4}$$

$$\sum_{i \in I} W_i x_{ij} \leq W, \quad \forall j = 1, \ldots, m \tag{1.5}$$

$$\sum_{i \in I} H_i y_{ij} \leq H, \quad \forall j = 1, \ldots, m \tag{1.6}$$

$$x_{ij} \in \{0, 1\}, \quad \forall i \in I, \forall j = 1, \ldots, m \tag{1.7}$$

$$y_{ij} \in \{0, 1\}, \quad \forall i \in I, \forall j = 1, \ldots, m \tag{1.8}$$

où $x_{ij} = 1$ si l'article i est placé dans le conteneur j, et $y_{ij} = 1$ si l'article i est placé dans le conteneur j sans chevauchement.

Cependant, une définition du BP2D reste complexe, et une généralisation utilise des techniques de programmation linéaire plus nuancées. De plus, la formulation est soumise à de nombreuses contraintes supplémentaires. Un agencement faisable pour le BP2D en 3 étapes consiste en un ensemble de conteneurs, chaque conteneur comprenant un ensemble de bandes, chaque bande comprenant un ensemble de piles, et chaque pile comprenant des articles ayant une largeur égale. Chaque motif ainsi défini peut être réduit à sa forme normale en déplaçant chaque article vers

sa position la plus haute et la plus à gauche, de sorte que l'espace vide n'apparaisse qu'au bas des piles, à droite de la dernière pile dans chaque bande et en dessous de la dernière bande. Dans [21], un modèle de programmation linéaire en nombres entiers (ILP) de taille polynomiale pour le BP2D à été proposée. Ce problème est en réalité NP-difficile [13], ce qui signifie qu'il est extrêmement complexe à résoudre pour des instances de taille significative. Néanmoins, son importance est incontestable dans de nombreux domaines industriels, notamment dans la découpe (industries du bois et du verre) et le emballage (transport et entreposage). Certaines applications peuvent nécessiter des contraintes et/ou des hypothèses supplémentaires. la section 1.2 présente un aperçu des principaux travaux récents sur le BP2D.

1.2 Revue de littérature

De nombreux travaux ont été réalisés sur le problème de BP2D. Ils proposent des méthodes de résolution de ce problème, que ce soit à travers des algorithmes gloutons, des algorithmes exacts ou des approches heuristiques.

1.2.1 Algorithmes gloutons

La plupart des algorithmes de la littérature sont de type glouton et peuvent être classés en deux familles :

— les algorithmes à une phase dans laquelle on charge directement les articles dans les conteneurs finis.

— les algorithmes à deux phases dont la première phase consiste à charger les articles dans une seule bande, c'est-à-dire un conteneur ayant une largeur W et une hauteur infinie. Dans la deuxième phase, la solution de la bande est utilisée pour construire un emballage dans des conteneurs finis.

De plus, la plupart des approches sont des algorithmes de niveaux, c'est-à-dire que le rangement du conteneurs est obtenu en plaçant les articles, de gauche à droite, en rangées formant des niveaux. Le premier niveau est le fond du conteneur, et les niveaux suivants sont produits par la ligne horizontale coïncidant avec le sommet de l'article le plus grand emballé sur le niveau inférieur. Trois stratégies classiques pour le rangement par niveaux ont été dérivées à partir d'algorithmes célèbres pour le cas unidimensionnel. Dans chaque cas, les articles sont initialement triés par hauteur décroissante et chargés dans la séquence correspondante. Soit j le numéro de l'article actuel, et s le dernier niveau créé :

— Algorithme Next-Fit Decreasing Height (**NFDH**) : l'article j est placé à gauche sur le niveau s, s'il convient. Sinon, un nouveau niveau ($s := s + 1$) est créé, et j est placé à gauche pour ce nouveau niveau.

— Algorithme First-Fit Decreasing Height (**FFDH**) : l'article j est placé à gauche sur le premier niveau où il convient, le cas échéant. Si aucun niveau ne peut accueillir j, un nouveau niveau est initialisé comme dans NFDH.

— Algorithme Best-Fit Decreasing Height (**BFDH**) : l'article j est placé à gauche sur le niveau pour lequel l'espace horizontal inutilisé est minimal. Si aucun niveau ne peut accueillir j, un nouveau niveau est initialisé comme dans NFDH.

Ces différents Algorithmes sont illustrés à la figure 1.2.

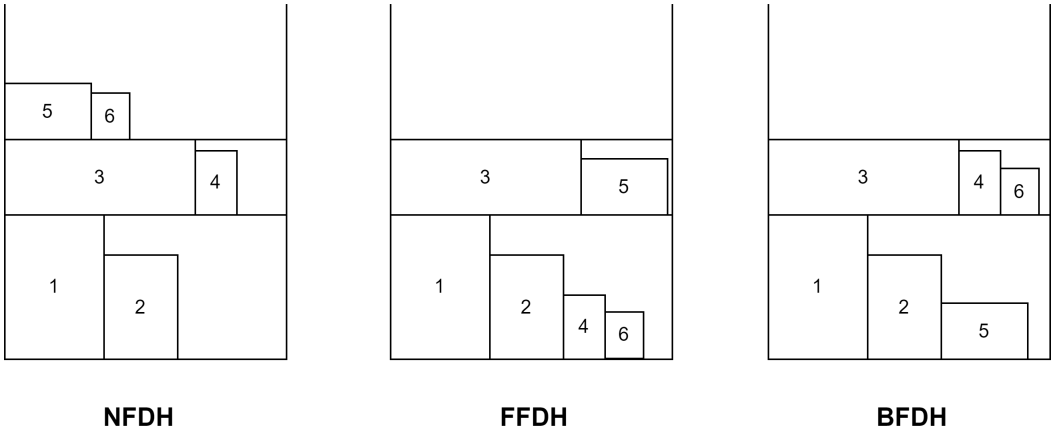


FIGURE 1.2 – Algorithme de rangement par niveaux

Algorithme en une phase

Deux algorithmes en une phase ont été présentés et évalués expérimentalement par Berkey and Wang [4].

L'algorithme Finite Next-Fit (FNF) empaquette directement les articles dans des conteneurs finis exactement de la même manière que l'algorithme HNF de la section 1.2.1.

L'algorithme Finite First-Fit (FFF) adopte plutôt la stratégie FFDH. L'élément actuel est placé sur le niveau le plus bas du premier conteneur où il s'insère. Si aucun niveau ne peut l'accueillir, un nouveau niveau est créé soit dans le premier conteneur approprié, soit en initialisant un nouveau conteneur (si aucun conteneur n'a assez d'espace vertical disponible).

Algorithmes en deux phases

Un algorithme en deux phases pour le problème de bin-packing fini, appelé Hybrid First-Fit (HFF), a été proposé par Chung et al. [6]. Dans la première phase, un emballage en bande [15] est obtenu par la stratégie FFDH. Une solution de bin-packing fini est ensuite obtenue en résolvant de manière heuristique un problème de bin-packing unidimensionnel (avec des tailles d'articles H_i et une capacité de conteneur H) à travers l'algorithme First-Fit Decreasing : initialiser le conteneur 1 pour charger le niveau 1, et, pour $i = 2, 3, \ldots$, charger le niveau courant i dans le conteneur indexé le plus bas où il convient, le cas échéant. Si aucun conteneur ne peut accueillir i, initialiser un nouveau.

Chung et al. [6] ont prouvé que, si les hauteurs sont normalisées à un, alors $HFF(I) \leq \frac{17}{8} \cdot OPT(I) + 5$ où $OPT(I)$ est la valeur de la solution optimale.

Berkey and Wang [4] ont proposé et évalué expérimentalement un algorithme en deux phases, appelé Finite Best-Strip (FBS), qui est une variation de HFF. La première phase est réalisée en utilisant la stratégie BFDH. Dans la deuxième phase, le problème de bin-packing unidimensionnel est résolu à travers l'algorithme Best-Fit Decreasing : charger le niveau actuel dans un conteneur disponible, en choisissant celui qui offre le moins d'espace vertical inutilisé parmi ceux où le niveau peut s'insérer, ou en créant un nouveau conteneur si nécessaire.

Frenk and Galambos. [12] ont proposé l'algorithme Hybrid Next-Fit (HNF) dont la performance dans le pire des cas est $HNF(I) \leq 3.382 \cdot OPT(I) + 9$ en supposant que les hauteurs et les largeurs sont normalisées à un.

Il existe également des algorithmes non basés sur les niveaux. La principale stratégie non basée sur les niveaux est connue sous le nom de Bottom-Left (BL), et consiste à placer l'article actuel dans la position la plus basse possible, à gauche. Baker et al.. [3] ont analysé les performances dans le pire des cas de l'algorithme résultant pour le problème de bin-packing en bande [15], et ont prouvé que :

— si aucun ordre d'articles n'est utilisé, BL peut être arbitrairement mauvais.

— si les articles sont ordonnés par largeur décroissante alors $BL(I) \leq 3 \cdot OPT(I)$.

Berkey and Wang.[4] ont proposé l'approche BL pour le cas des conteneurs finis. Leur algorithme Finite Bottom-Left (FBL) trie initialement les articles par largeur décroissante. L'article actuel est ensuite placé dans la position la plus basse à gauche de tout conteneur initialisé. Si aucun conteneur ne peut le contenir, un nouveau est initialisé.

Lodi et al. [18] ont proposé une approche différente non basée sur les niveaux, appelée directions alternatives (AD). L'algorithme initialise L conteneurs (L étant une borne inférieure sur la valeur de la solution optimale) en plaçant sur leurs fonds un sous-ensemble de l'ensemble des articles, en suivant une politique BFDH. Les articles restants sont placés, un conteneur à la fois, dans des bandes, alternativement de gauche à droite et de droite à gauche. Dès qu'aucun article ne peut être placé dans l'une ou l'autre direction dans le conteneur actuel, on passe au prochain conteneur initialisé ou un nouveau conteneur vide devient le conteneur actuel.

1.2.2 Algorithmes exacts

Martello and Vigo [22] ont proposé une approche énumérative pour obtenir la solution exacte du BP2D. Leur méthode commence par trier les articles par ordre décroissant de leur aire, suivi d'une procédure de réduction visant à déterminer l'emballage optimal de certains bacs afin de réduire la taille de l'instance. Ensuite, une première solution, appelée solution initiale et notée z^*, est obtenue de manière heuristique. Cette méthode repose sur un schéma de branchement à deux niveaux : l'arbre de décision de la branche externe attribue chaque article à un conteneur sans spécifier sa position exacte, tandis que l'arbre de décision de la branche interne détermine un emballage réalisable pour les articles actuellement assignés à un conteneur, éventuellement en énumérant tous les motifs possibles. Cependant cet algorithme génère de nombreuses redondances puisque le même article peut être packé à plusieurs reprises dans une position donnée pendant

l'énumération.

Afin d'éviter d'explorer des solutions redondantes, Fekete and Schepers[10] introduisent un modèle théorique basé sur les graphes. Ils montrent qu'une classe de emballage (c'est-à-dire un ensemble d'emballages ayant des propriétés communes) peut être associée à une paire de graphes d'intervalles. Ils proposent également un algorithme énumératif [9] pour construire de tels graphes d'intervalles. Leur méthode réduit considérablement le nombre de redondances par rapport à la méthode proposée par Martello and Vigo [22].

Clautiaux et al.[7] proposent deux méthodes de Branch and Bound pour le problème de BP orthogonal en deux dimensions (2OPP). Le premier algorithme (LMAO) est une amélioration de la méthode proposée par Martello and Vigo[22]. Au lieu de tester l'emballage des articles dans dans toutes les positions possibles, l'algorithme énumère les chargements des articles uniquement dans la position la plus basse à gauche. Le deuxième algorithme (TSBP) est une méthode de Branch and Bound en deux étapes basée sur une nouvelle relaxation du problème. Dans la première étape, toutes les solutions d'un problème relaxé sont énumérées. Pour chaque solution trouvée, une seconde méthode énumérative est utilisée pour rechercher une solution pour le problème initial correspondant à la solution actuelle. Cette méthode renvoie de meilleurs résultats sur plusieurs instances.

Han et al.[14] ont abordé une variante du problème BP2D, connue sous le nom de "multi-type two-dimensional bin packing problem" (M2BP). Dans cette variante, ils ont exploré trois approches distinctes : une heuristique gloutonne simple appelée "First Fit by Ordered Deviation" (FFOD), une méthode basée sur le recuit simulé (SA) et un algorithme exact basé sur la génération de colonnes avec Branch and Bound (CG). Pour l'approche CG, ils ont d'abord transformé le M2BP en un problème de couverture d'ensemble, puis ont suivi les étapes suivantes : ils ont commencé par la phase de génération de colonnes, où ils ont créé le problème de couverture d'ensemble initial à partir de la solution FFOD, appelé le Problème Maître Restreint (RMP), et ont résolu ce problème maître relaxé en utilisant la méthode de génération de colonnes. Cette étape est répétée jusqu'à ce qu'aucune colonne avec un coût réduit négatif ne puisse être générée. Ensuite, si la solution est entière, l'algorithme se termine, sinon, on passe à l'étape de Branch and Bound pour poursuivre la résolution. SA et FFOD seront décrits dans la section 1.2.3.

Nous avons examiné les avancées récentes des algorithmes exacts pour le problème de BP2D. Les résultats des expériences de calcul montrent que les instances de petite à moyenne taille peuvent être résolues de manière optimale à l'aide de méthodes exactes, tandis que les instances de plus grande taille ne peuvent être traitées que par des algorithmes d'approximation.

1.2.3 Approches heuristiques

Ces dernières années, les techniques de métaheuristique sont devenues un outil populaire pour résoudre de manière approximative les problèmes d'optimisation combinatoire difficiles. Lodi et al.[17, 20] ont développé des algorithmes de recherche tabou efficaces pour le BP2D et certaines de ses variantes. La recherche tabou est caractérisée par l'utilisation d'un schéma de recherche et d'un voisinage indépendants du problème de BP spécifique à résoudre. Ce cadre peut ainsi être

appliqué à pratiquement toutes les variantes du BP2D [19]. À partir d'une solution courante, les mouvements modifient l'emballage d'un sous-ensemble S d'articles pour vider un conteneur cible spécifié. Le conteneur cible est sélectionné en minimisant une fonction dépendant du contenu actuel des conteneurs. Une fois le conteneur cible déterminé, un nouveau chargement pour S est obtenu en exécutant un algorithme heuristique approprié. Si le mouvement permet de vider le bac cible, un nouvel article est sélectionné pour un nouveau chargement. Sinon, le voisinage est élargi en augmentant la taille du paramètre k, qui définit la taille et la structure du voisinage. Une solution initiale est obtenue en exécutant l'algorithme heuristique sur l'ensemble des articles, tandis que la solution de recherche tabou initiale consiste à charger un article par conteneur. L'exécution de l'algorithme s'arrête dès qu'une solution optimale est trouvée ou qu'une limite de temps est atteinte.

Liu et al.[16] ont proposé une heuristique pour résoudre le BP2D. Cette heuristique reformule le BP2D en un schéma de programmation dynamique, visant à minimiser la zone de gaspillage totale des conteneurs utilisés. Pour un ensemble de N items, représenté par un vecteur B de dimension N, la zone de gaspillage totale pour α items packés dans θ bacs est déterminée par l'équation de récurrence $\gamma(\alpha, \theta) = \min_{\alpha' < \alpha} \{\gamma(\alpha', \theta - 1) + h(\alpha, \alpha', \theta)\}$, où $h(\alpha, \alpha', \theta)$ est le coût de transition de l'état $(\alpha', \theta - 1)$ à l'état (α', θ). Ici, α est défini comme $|B|$. La récurrence est basée sur le calcul de la fonction de transition et est résolue par une heuristique dont l'idée principale consiste à essayer de remplacer un petit élément contenu dans la solution actuelle par un élément plus grand non inclus dans la solution.

L'algorithme FFOD introduit à la section 1.2.2 proposé par, Han et al.[14] pour le M2BP, renvoie une solution qui est soit satisfaisante, soit fournit une solution réalisable initiale dont la valeur fournit une borne supérieure z_{ub} et un point de départ pour l'algorithme SA ou CG. Il considère séquentiellement les objets et attribue l'objet soit à un conteneur existant, soit à un nouveau conteneur. L'attribution se fait en fonction du "coût d'opportunité"[14] minimum. Cette définition du "coût d'opportunité" permet d'emballer les objets de manière compacte dans les conteneurs tout en maintenant l'équilibre des charges de ressources sur chaque conteneur. Pour l'algorithme SA[14], l'espace des solutions, $\mathcal{F}$, est l'ensemble de toutes les partitions possibles des N objets en k sous-ensembles et de toutes les affectations possibles de types de conteneurs à chaque sous-ensemble au sein d'une partition. Les partitions de $\mathcal{F}$ ne sont pas contraintes à celles qui permettent uniquement des sous-ensembles faisables d'objets (c'est-à-dire des sous-ensembles d'objets pouvant être assignés à au moins l'un des types de bacs disponibles sans violer aucune contrainte de ressources). Un état de solution, S, est toute partition. Le mécanisme de génération d'états de solution et de structure du voisinage est fait par un schéma de transfert à sens unique. En considérant un état de solution S, le transfert à sens unique génère un nouvel état en déplaçant un objet d'un conteneur à un autre ou en déplaçant un objet vers un nouveau conteneur. Tous les transferts à sens unique possibles à partir d'un état donné définissent la structure du voisinage. Le fonctionnement général de l'algorithme est le suivant : il commence par obtenir une solution initiale S à l'aide de l'heuristique FFOD, puis détermine la température initiale T_0 et la fonction de refroidissement $g(T)$. Ensuite, il exécute une boucle principale jusqu'à ce que la température

soit figée ou qu'un critère d'arrêt soit atteint. À chaque itération, il génère un voisinage aléatoire S' de S et évalue la variation d'énergie Δ entre les deux solutions. Si $\Delta < 0$, il accepte toujours le voisinage S' comme la nouvelle solution courante S. Sinon, il évalue la probabilité d'accepter le voisinage S' en fonction de la température actuelle T et de Δ, et décide de l'accepter avec une certaine probabilité. La température est ensuite mise à jour selon la fonction de refroidissement $g(T)$. L'algorithme se termine par la sortie de la meilleure solution trouvée S^* et de sa valeur z_{ub}, qui est utilisée comme borne supérieure de la solution optimale.

Abeysooriya et al.[2] présentent une étude sur le BP2D irrégulier avec plusieurs conteneurs homogènes (2DIBPP). Ils proposent une méthode heuristique basée sur l'algorithme "Jostle" [8] initialement conçu pour le probleme de BP en bandes pour résoudre le 2DIBPP. Ils explorent également différentes stratégies de rotation des pièces et de critères de placement, révélant que certains choix peuvent générer de meilleurs résultats que d'autres. Les performances des algorithmes qu'ils proposent sont évaluées sur différents ensembles d'instances avec des formes convexes et non convexes, démontrant leur efficacité pour résoudre différentes variantes du BP2D.

Martinez-Sykora et al.[23] présentent un modèle de programmation en nombres entiers pour déterminer l'association entre les articles et les conteneurs, suivi d'un modèle de programmation en nombres entiers mixtes pour placer les articles dans les conteneurs. Leur méthode prend en compte la rotation des articles. Les résultats indiquent que cette approche offre de bonnes performances en termes de qualité lorsqu'elle est évaluée sur divers ensembles d'instances présentant différentes propriétés.

1.3 Conclusion

Après avoir examiné la littérature sur les solutions existantes pour le problème de bin-packing en deux dimensions (BP2D) et ses variantes, nous avons conclu qu'il n'était pas réaliste de développer un algorithme exact pour notre problème. Parmi les approches heuristiques et métaheuristiques discutées dans ce chapitre, nous avons déterminé qu'une approche heuristique est la plus appropriée. Cela est dû à sa capacité à produire de bonnes solutions en un temps relativement court et à son efficacité pour traiter des volumes de données importants sans dégradation significative des performances. En effet, nous devons gérer de grandes instances comportant jusqu'à 260 000 articles et plusieurs milliers de camions, tout en garantissant un temps de calcul maîtrisé.

Chapitre 2

Problème de placement 2D : Modélisation mathématique

Dans ce chapitre, nous proposons une formulation mathématique de notre problème de BP2D. Nous commencerons par présenter une description détaillée de notre problème. Ensuite on proposera notre modèle mathématique à travers les diverses notations, paramètres, variables de décision et contraintes impliquées.

2.1 Description du problème

Le problème de placement dans les camions implique des piles empilées sur le plancher bidimensionnel. Chaque pile contient des articles qui sont empilés les uns au-dessus des autres. Il y a trois dimensions corrélées (longueur, largeur, hauteur) ainsi qu'une dimension de poids supplémentaire. Nous appellerons les dimensions horizontales la longueur et la largeur. La position de tout élément (article ou pile) est décrite dans une solution par sa position dans les trois axes X, Y et Z. Chaque camion a une origine, correspondant à la position la plus à gauche en bas (cf. Figure 2.1).

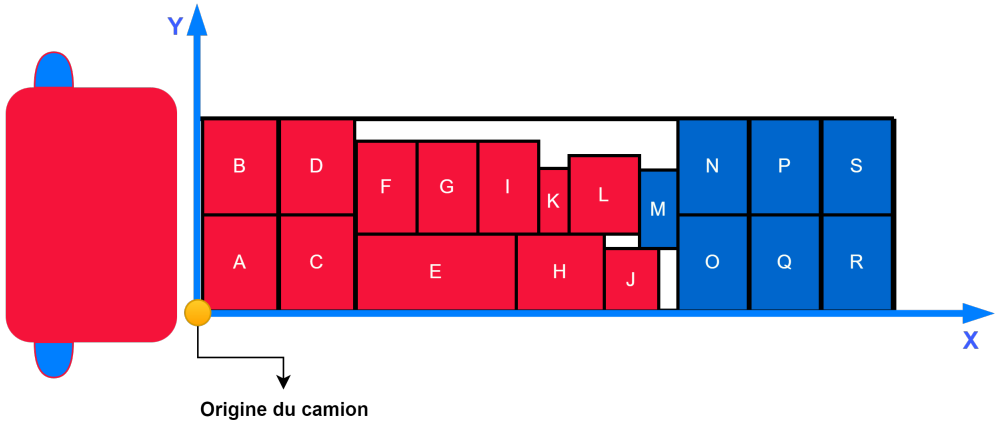


FIGURE 2.1 – Le camion vu de dessus.

Pour la suite du document, on ne considère que le placement bidimentionnel c'est a dire le placement des piles sur le plancher bidimentionnel et que la position de tout élément (pile) est décrite dans une solution par sa position dans les deux axes X, Y pour notre problème de placement 2D.

2.1.1 Les articles

Un article i est un emballage contenant un produit IR_i. Il est caractérisé par ses dimensions spatiales longueur/largeur (IL_i, IW_i), son poids IM_i, et un code d'empilage IS_i. Par convention, la longueur est plus grande que la largeur.

Le code d'empilage est utilisé pour définir quels articles peuvent être empilés dans la même pile : seuls les articles avec le même code d'empilage peuvent être empilés dans la même pile. Les articles ayant le même code d'empilage partagent la même longueur et la même largeur. Mais l'inverse n'est pas vrai : des articles avec des longueurs/largeurs identiques peuvent ne pas partager le même code d'empilage.

Un article i est associé à une empilabilité maximale ISM_i. Cela représente le nombre maximal d'articles i dans une pile. Les articles peuvent être tournés dans une seule dimension (horizontale), c'est-à-dire que le bas d'un article reste le bas. Un article peut être orienté en longueur (la longueur de l'article est parallèle à la longueur du camion) ou en largeur (la longueur de l'article est parallèle à la largeur du camion). Dans la Figure 2.1, la pile A est orientée en largeur, tandis que la pile E est orientée en longueur. Un article i peut avoir une orientation forcée IO_i, auquel cas la pile s contenant l'article i doit avoir la même orientation.

Un article i a une hauteur de chevauchement $\widehat{IH_i}$, qui est utilisée pour le calcul de la hauteur des piles : si l'article i_1 est empilé au-dessus de l'article i_2, alors la hauteur totale est $IH_{i_1} + IH_{i_2} - \widehat{IH_{i_1}}$ (cf. Figure 2.2). Pour la majorité des articles, la hauteur de chevauchement est égale à zéro.

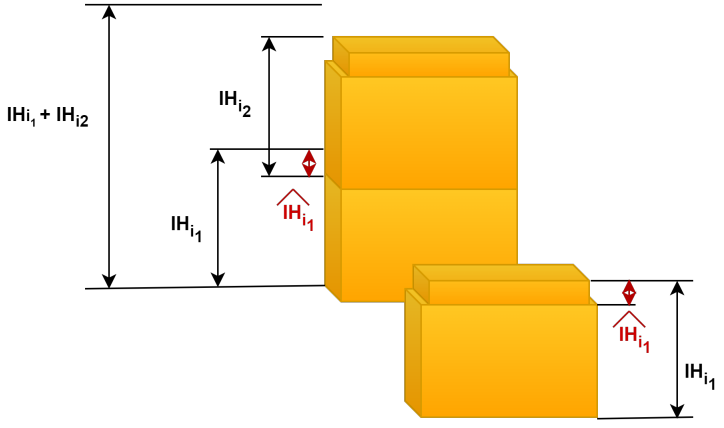


FIGURE 2.2 – Exemple d'une pile de 2 articles avec une hauteur de chevauchement.

2.1.2 Les piles

Une pile s contient des articles, empilés les uns au-dessus des autres, et est chargée dans un camion. Une pile s est caractérisée par ses dimensions de longueur et largeur (sl_s, sw_s), son poids (sm_s) et son groupe/fournisseur (sg_s). La longueur et la largeur d'une pile sont celles de ses articles (tous les articles d'une pile partagent la même longueur/largeur). L'orientation d'une pile est l'orientation de tous ses articles. Il y a une orientation unique (en longueur ou en largeur) pour tous les articles d'une pile. Une pile est également caractérisée par les coordonnées de son origine (sx^o_s, sy^o_s) et de ses points extrêmes (sx^e_s, sy^e_s).

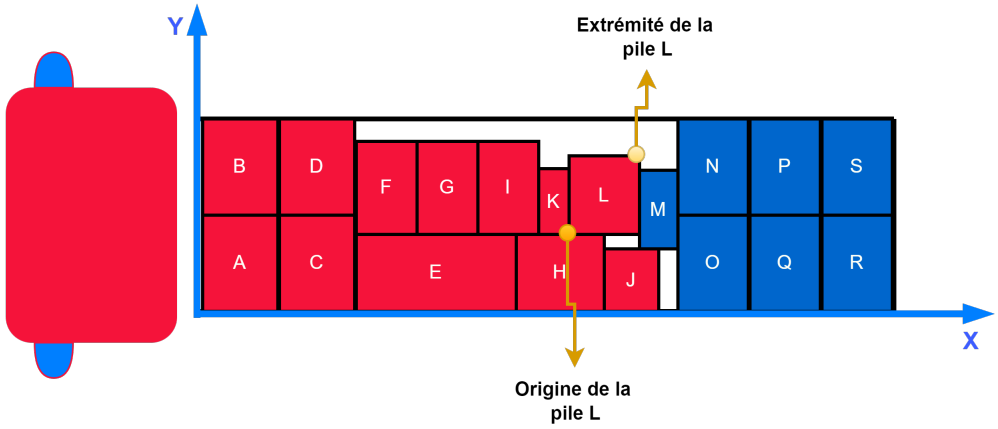


FIGURE 2.3 – Points origine et extrême d'une pile.

Comme illustré à la Figure 2.3, le camion contient un ensemble de pile représenter chacune par une lettre et chaque pile appartient à un unique groupe représenté par sa couleur. L'origine est le point le plus bas à gauche de la pile (le point le plus proche de l'origine du camion), tandis que le point extrême est le point le plus haut à droite de la pile (le point le plus éloigné de l'origine du camion). Les coordonnées des points d'origine et des points extrêmes d'une pile doivent satisfaire les hypothèses suivantes :

— $sx^e_s - sx^o_s = sl_s$ et $sy^e_s - sy^o_s = sw_s$ si la pile s est orientée en longueur.

— $sx^e_s - sx^o_s = sw_s$ et $sy^e_s - sy^o_s = sl_s$ si la pile s est orientée en largeur.

Le poids de la pile sm_s est la somme des poids des articles qu'elle contient :

$$sm_s = \sum_{i \in I_s} IM_i. \tag{2.1}$$

La hauteur de la pile sh_s est la somme des hauteurs de ses articles, moins leur hauteur de chevauchement (cf. Figure 2.2) :

$$sh_s = \sum_{i \in Is} IHi - \sum_{i \in Is \text{ et } i \neq \text{bottom\_item}} \widehat{IH_i} \tag{2.2}$$

On supposera que la hauteur de chacune nos piles est inférieure à celle du camion et elle ne sera pas pris en compte.

2.1.3 Les camions

Un camion t est caractérisé par ses dimensions longueur/largeur (TL_t, TW_t) (cf. Figure 2.4), son poids maximal autorisé de chargement TM_t^m, et son coût TC_t. Le coût du camion représente un coût fixe qui ne dépend pas du nombre d'articles chargés dans le camion. Toutes les piles emballées dans le camion t doivent respecter une densité maximale TEM_t, assurant ainsi que le poids est correctement réparti sur la surface du camion, évitant ainsi des zones de surpoids qui pourraient déséquilibrer le camion.

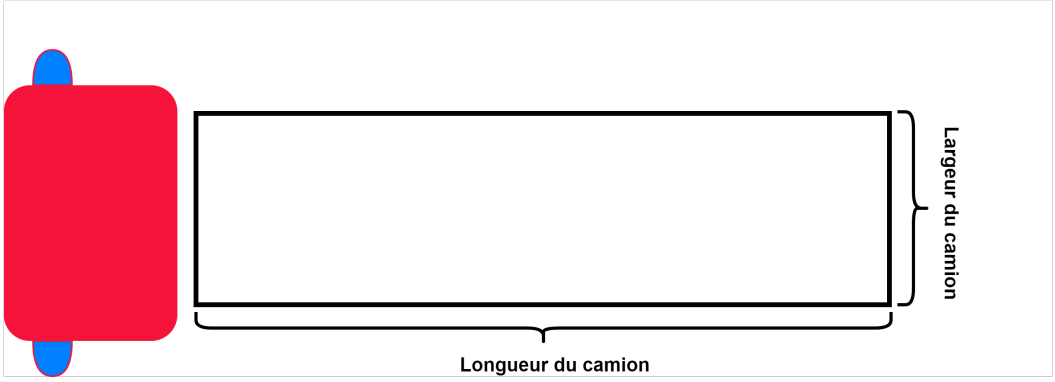


FIGURE 2.4 – Dimensions d'un camion.

2.1.4 Contraintes

Il existe quatre ensembles de contraintes :

- **Contraintes sur les articles** qui garantissent la compatibilité entre les articles et les camions, en précisant quels articles peuvent être chargés dans quels camions. Plus précisément :
  (a) Dimensions et poids des articles : Les articles sont caractérisés par leur longueur, largeur, hauteur et poids. Un camion ne peut charger un article que si ses dimensions et son poids respectent les limites de ce camion.
  (b) Code de stackabilité : Chaque article a un code de stackabilité qui détermine quels articles peuvent être empilés ensemble. Seuls les articles avec le même code de stackabilité peuvent être placés dans la même pile.
  (c) Fenêtre temporelle : Chaque article a une fenêtre de livraison spécifique (temps d'arrivée le plus tôt et le plus tard), et seuls les camions dont les temps de livraison tombent dans cette fenêtre peuvent transporter cet article.
  (d) Orientation forcée : Certains articles ont une orientation forcée, ce qui signifie que ces articles doivent être placés dans une orientation spécifique lorsqu'ils sont chargés dans le camion.

- **Contraintes sur les piles** qui contrôlent la composition des piles, principalement en définissant quels articles peuvent être emballés ensemble dans une même pile. Les règles suivantes s'appliquent :
  (a) Code de stackabilité : Comme mentionné précédemment, les articles avec des codes de stackabilité différents ne peuvent pas être empilés ensemble.
  (b) Hauteur maximale des piles : La hauteur d'une pile est la somme des hauteurs des articles moins toute hauteur de chevauchement applicable. Une pile ne peut pas dépasser la hauteur maximale autorisée pour un camion.
  (c) Poids maximal de la pile : Le poids d'une pile est la somme des poids des articles empilés. Chaque pile doit respecter les limites de poids maximales définies pour le camion.
- **Contraintes de placement** liées à la manière dont les piles sont placées dans les camions :
  (a) Ordre de placement : L'ordre dans lequel les articles sont chargés dans le camion doit respecter l'ordre de chargement défini par le fournisseur et les docks de l'usine. Par exemple, les articles d'un fournisseur doivent être placés ensemble selon un ordre prédéfini.
  (b) Positionnement dans le camion : Les piles doivent être positionnées dans le camion de manière à respecter les contraintes de dimension, de non chevauchement et d'adjacence.
- **Contraintes de poids** qui imposent les poids maximaux autorisés des piles dans le camion, en tenant compte de la répartition du poids sur les essieux :
  (a) Poids total autorisé : Le poids total de toutes les piles dans le camion ne doit pas dépasser le poids maximal autorisé pour le camion.
  (b) Répartition du poids sur les essieux : Il est crucial de s'assurer que le poids est bien réparti sur les essieux avant et arrière du camion, en respectant les limites de poids spécifiques pour chaque essieu afin d'assurer la stabilité du camion pendant le transport.

Nous supposerons que les contraintes sur les articles et celles sur les piles sont satisfaites.

2.1.5 Objectif

L'objectif est de minimiser le coût total de transport. Le coût de transport est le coût total des camions ; c'est un coût fixe qui dépend uniquement du nombre de camions, et non du nombre d'articles chargés.

2.2 Modélisation du problème

2.2.1 Notations

Par convention, les paramètres sont déclarés avec des lettres majuscules, les variables avec des lettres minuscules et les ensembles avec des lettres majuscules ornées d'un tilde (ex : $\widetilde{U}$). Les paramètres relatifs aux camions sont préfixés par T et ceux relatifs aux piles par S. Les paramètres et les variables relatifs aux contraintes de poids seront définis dans la section 2.2.4.

2.2.2 Hypothèses

Hypothèses sur les articles

On suppose que les contraintes sur les articles sont satisfaites [25].

Hypothèses sur les piles

(S1) Tous les articles emballés dans une pile partagent le même fournisseur, la même usine, le même code de capacité d'empilage et le même quai de fournisseur.

(S2) Pour toute pile s emballée dans le camion t, si $TF_t =$ non, alors tous les articles de la pile s doivent partager le même quai d'usine.

(S3) Pour toute pile s emballée dans le camion t, si $TF_t =$ oui, la pile s peut contenir des articles avec deux quais d'usine ayant des ordres de chargement consécutifs.

— Il y a une restriction : pour chaque code de capacité d'empilage SC présent parmi les piles d'un camion t, une seule pile avec le code de capacité d'empilage SC peut contenir des articles avec deux quais d'usine.

(S4) Si un article i d'une pile s a une orientation forcée IO_i, alors tous les articles de la pile s doivent partager la même orientation ($sos = IO_i$).

— Par conséquent, il ne peut pas y avoir deux orientations forcées différentes des articles dans la même pile.

(S5) Pour toute pile s emballée dans le camion t, le poids total des articles emballés au-dessus de l'article du bas associé au produit γ ne doit pas dépasser le poids maximal $TMM_{t\gamma}$:

$$\sum_{i \in Sif_s \text{ et } i \neq \text{bottom\_item}} IM_i \leq TMM_{t\gamma}$$

(S6) Le nombre d'articles emballés dans une pile s ne doit pas dépasser la plus petite "capacité d'empilage maximale" ISM_i des articles i présents dans la pile s. En d'autres termes, pour chaque article i d'une pile s, le nombre d'articles de s ne doit pas dépasser ISM_i.

(S7) La densité d'une pile s ne doit pas dépasser la densité maximale de pile définie pour le camion t dans lequel la pile s est chargée :

$$\frac{SM_s}{SL_s \times SW_s} \leq TEM_t$$

Paramètres

Les paramètres sont définis dans les fichiers d'entrée de chaque instance.

— $\widetilde{S}$: ensemble de piles s

— $\widetilde{T}$: ensemble de camions t

— SL_s : longueur de la pile s

— SW_s : largeur de la pile s

— SM_s : poids de la pile s

— SO_s : orientation forcée de pile s

— SG_s : quai d'usine de la pile s

— TL_t : longueur du camion t

— TW_t : largeur du camion t

— TM_t^m : poids de chargement maximal autorisé du camion t

— TEM_t : densité maximale des piles dans le camion t

— TC_t : coût du camion t

— α^T : coefficient du coût de transport dans la fonction objective

Variables

— $\widetilde{TS_t}$: ensemble de piles chargée dans le camion t

— sx_s^o, sy_s^o : coordonnées du point d'origine de la pile s

— sx_s^e, sy_s^e : coordonnées du point d'extrémité de la pile s

2.2.3 Objectif

La fonction objectif est :

$$Min \quad (\alpha^T * \sum_{t \in \widetilde{T}} TC_t) \tag{2.3}$$

2.2.4 Contraintes

Contraintes de placement

(P1) Le placement d'une pile s dans un camion t ne doit pas dépasser les dimensions du camion :

$$\forall t \in \widetilde{T}, \quad \forall s \in \widetilde{TS_t}, \quad sx_s^e \leq TL_t \text{ et } sy_s^e \leq TW_t \tag{2.4}$$

(P2) Les piles emballées dans un camion t ne peuvent pas se chevaucher :

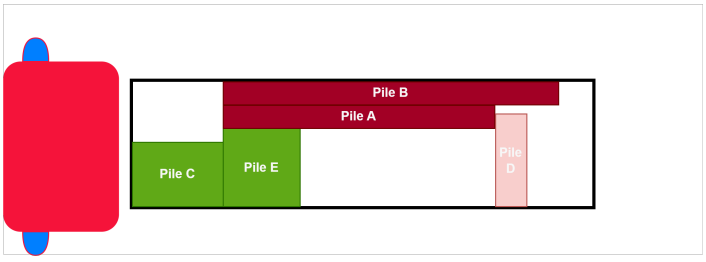
$$\begin{aligned} \forall t \in \widetilde{T}, \quad \forall s_1, s_2 \in \widetilde{TS_t} \text{ avec } sx_{s_1}^o \leq sx_{s_2}^o, \\ \text{si } (sx_{s_2}^o < sx_{s_1}^e) \text{ alors } sy_{s_2}^o \geq sy_{s_1}^e \text{ ou } sy_{s_2}^e \leq sy_{s_1}^o \end{aligned} \tag{2.5}$$

Si s_1 et s_2 se chevauchent sur l'axe X, alors ils ne peuvent pas se chevaucher sur l'axe Y.

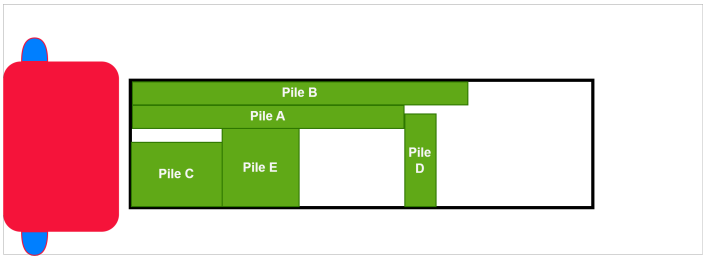
(P3) Toute pile doit être adjacente à une autre pile sur sa gauche sur l'axe X, ou s'il y a une seule pile dans le camion, la pile unique doit être placée à l'avant du camion (adjacent au conducteur) :

$$\forall t \in \widetilde{T}, \quad \forall s_1 \in \widetilde{TS_t} \text{ avec } sx^o_{s_1} > 0, \quad \exists s_2 \in \widetilde{TS_t} \text{ avec } (sx^e_{s_2} = sx^o_{s_1}) \\ \text{et } (sy^o_{s_2} \in [sy^o_{s_1}, sy^e_{s_1}] \text{ ou } sy^e_{s_2} \in [sy^o_{s_1}, sy^e_{s_1}]) \tag{2.6}$$

Dans la Figure 2.5a, le placement des piles A et B est interdit car elles ne sont ni adjacentes à la partie gauche d'une autre pile dans le camion, ni positionnées à l'avant du camion (par conséquent, le placement de la pile D est également interdit puisqu'elle est adjacente uniquement à la pile B), tandis que le placement des piles les piles C et E est autorisé. Dans la figure 2.5b le placement de toutes les piles est autorisé. La raison est qu'en cas de freinage brusque par le conducteur, toute pile doit être retenue soit par une autre pile, soit par l'avant du camion.



(a) Exemple de placement de piles interdit.



(b) Exemple de placement de piles autorisé.

FIGURE 2.5 – Exemple de placement de piles.

Contraintes de poids

Un camion est composé d'un tracteur (incluant la cabine du conducteur) et d'une remorque (où les piles sont chargées), comme illustré dans les figures 2.6a et 2.6b. Les paramètres seront définis pour le tracteur ou la remorque, tandis que les variables seront définies uniquement pour la remorque.

— Paramètres

  — CM : poids du tracteur

- CJ^{fm} : distance entre les essieux avant et moyen du tracteur
- CJ^{fc} : distance entre l'essieu avant et le centre de gravité du tracteur
- CJ^{fh} : distance entre l'essieu avant et le harnais du tracteur
- EM : poids de la remorque vide
- EJ^{hr} : distance entre le harnais et l'essieu arrière de la remorque
- EJ^{cr} : distance entre le centre de gravité de la remorque et l'essieu arrière
- EJ^{eh} : distance entre le début de la remorque et le harnais
- EM^{mr} : poids maximal sur l'essieu arrière de la remorque
- EM^{mm} : poids maximal sur l'essieu du milieu de la remorque

— Variables

- tm_t : poids des piles chargées dans le camion t
- ej^e : distance entre le centre de gravité des piles et le début de la remorque
- ej^r : distance entre le centre de gravité des piles et l'essieu arrière de la remorque
- em^h : poids sur le harnais de la remorque
- em^r : poids sur l'essieu arrière de la remorque
- em^m : poids sur l'essieu du milieu de la remorque

— formules pour calculer les poids sur le harnais et les essieux

$$ej^e = \frac{\sum_{s\in\widetilde{TS_t}} \left(sx_s^o + \frac{sx_s^e - sx_s^o}{2}\right) \times sm_s}{tm_t} \tag{2.7}$$

$$ej^r = EJ^{eh} + EJ^{hr} - ej^e \tag{2.8}$$

$$em^h = \frac{tm_t \times ej^r + EM \times EJ^{cr}}{EJ^{hr}} \tag{2.9}$$

$$em^r = tm_t + EM - em^h \tag{2.10}$$

$$em^m = \frac{CM \times CJ^{fc} + em^h \times CJ^{fh}}{CJ^{fm}} \tag{2.11}$$

(W1) Les poids des piles chargées dans un camion t ne doivent pas dépasser le poids maximal autorisé du camion :

$$\forall t \in \widetilde{T}, \quad tm_t \leq TM_t^m \text{ avec } tm_t = \sum_{s\in TS_t} sm_s \tag{2.12}$$

(W2) Les poids sur l'essieu du milieu et sur l'essieu arrière d'un camion ne doivent pas dépasser les poids maximaux autorisés pour ces deux essieux :

$$em^m \leq EM^{mm} \quad \text{et} \quad em^r \leq EM^{mr} \tag{2.13}$$

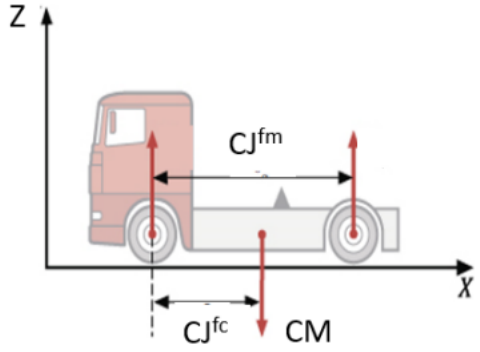
La contrainte 2.13 doit être respectée lorsque le camion est sur la route. Par exemple, si le camion ramasse le fournisseur A, puis le fournisseur B, puis se rend à l'usine, la contrainte doit être respectée avec seulement les piles du fournisseur A dans le camion (trajet depuis le fournisseur A vers le fournisseur B) et avec tous les articles des fournisseurs A et B (trajet depuis le fournisseur B à l'usine). En d'autres termes, la vérification de la contrainte 2.13 se fait en 2 étapes :

(W21) Les variables $tm_{t_a}, ej^{e_a}, ej^{r_a}, em^{h_a}, em^{r_a}, em^{m_a}$ sont calculées uniquement avec les piles du fournisseur A et la contrainte 2.14 doit être respectée.

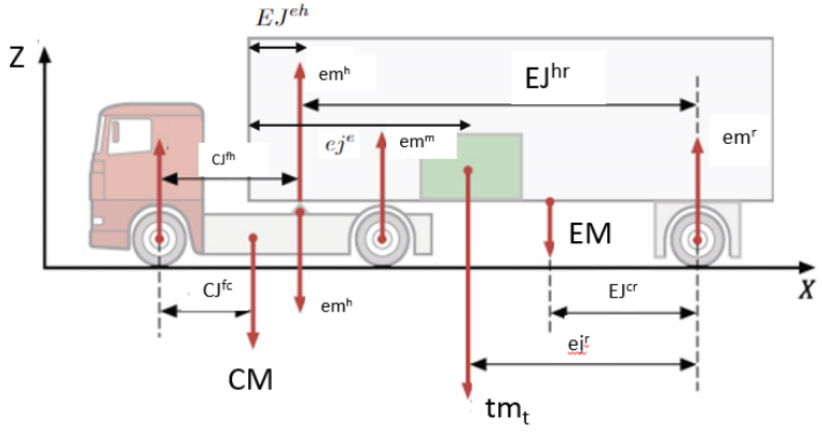
$$em^{m_a} \leq EM^{mm} \quad \text{et} \quad em^{r_a} \leq EM^{mr} \tag{2.14}$$

(W22) Les variables $tm_{t_{ab}}, ej^{e_{ab}}, ej^{r_{ab}}, em^{h_{ab}}, em^{r_{ab}}, em^{m_{ab}}$ sont calculées avec les piles des fournisseurs A et B, et la contrainte 2.15 doit être respectée.

$$em^{m_{ab}} \leq EM^{mm} \quad \text{et} \quad em^{r_{ab}} \leq EM^{mr} \tag{2.15}$$



(a) Paramètres du tracteur.



(b) Paramètres et variables de la remorque.

FIGURE 2.6 – Paramètres du tracteur et de la remorque. [25]

2.3 Conclusion

Dans ce chapitre, nous avons présenté une modélisation mathématique détaillée de notre problème de placement en deux dimensions (BP2D). Nous avons décrit les éléments constitutifs du problème, tels que les articles, les piles, les camions et les contraintes associées. Cette modélisation est cruciale pour comprendre les défis inhérents au problème et pour orienter le développement des algorithmes dans les chapitres suivants.

Chapitre 3

Approche de résolution

Optimiser le chargements des camions chez RG revient à trouver le meilleur plan de chargement de milliers de piles dans un ensemble de camions tout aussi importants. Dans ce cadre, nous présentons notre méthode proposée pour résoudre ce problème introduit dans les chapitres précédents. Notre méthode est basée sur un algorithme de recherche arborescente itérative (IBS), une heuristique efficace pour explorer les espaces de solution de grande taille. Cette approche est complétée par un schéma de branchement adapté servant de critère de sélection et de filtrage spécifique pour optimiser la qualité des solutions trouvées. La génération de la solution se compose de trois étapes principales. Tout d'abord, une solution initiale est obtenue en utilisant la méthode Root() du schéma de branchement (ligne 2 Alg. 1). Cette solution sert de point de départ pour les améliorations ultérieures. Ensuite, les solutions sont stockées et gérées dans une structure appelée Solution (ligne 2 Alg. 1). Cette structure permet de conserver les meilleures solutions trouvées pendant l'exploration et de les comparer pour sélectionner les plus prometteuses. Enfin, la méthode applique une stratégie de sélection de traitement et de filtrage de solution (Alg. 2). À chaque itération, les nœuds enfants générés à partir du nœud courant sont évalués et ceux qui présentent le plus grand potentiel d'amélioration sont ajoutés à la file d'attente pour une exploration plus approfondie. Ce processus itératif continue jusqu'à ce que toutes les files d'attente soient vides ou qu'un critère d'arrêt prédéfini soit atteint (lignes 4,7... Alg. 2). Dans les sections suivantes, nous fournirons une description détaillée de chaque composant de la méthode proposée et des détails de mise en œuvre.

Algorithm 1 Algorithme de Recherche Arborescente Itérative

```
startTime ← getTime()
solution ← initialiserSolutionPool(branchingScheme, taille)
maxSQ ← parameter.minSizeOfTheQueue
while maxSQ < parameter.maxSizeOfTheQueue do
    file ← [set(), set()]
    historique ← [dict(), dict()]
    nœudCourant ← branchingScheme.Root()
    file[0].add(nœudCourant)
    profondeurCourante ← 0
    Exploration arborescente
    mis à jour de maxSQ
end while
retourner solution.best()
```

Algorithm 2 Algorithme d'exploration arborescente

```
while True do
    while file[profondeurCourante] n'est pas vide do
        nœudCourant1 ← file[profondeurCourante].pop()
        if parameters.timeLimit then
            break
        end if
        if not branchingScheme.better(parameters.goal, solution.best()) then
            break
        end if
        enfants ← branchingScheme.children(nœudCourant1)
        for chaque enfant dans enfants do
            mis a jour de la profondeurEnfant
            if branchingScheme.better(enfant, solution.worst()) then
                solution.add(enfant)
            end if
            if not leaf(enfant) then
                mise a jour file et historique
            end if
        end for
    end while
end while
```

3.1 Description de l'algorithme

3.1.1 Objectifs

L'objectif principal de cet algorithme est de trouver des solutions optimales ou quasi-optimales pour des problèmes de grande taille en utilisant une approche de recherche guidée par des faisceaux.

Cela permet de réduire l'espace de recherche et d'améliorer les performances de l'algorithme par rapport à des méthodes de recherche exhaustives.

3.1.2 Concepts et Structures de Données

Paramètres de l'algorithme

— *Schema de branchement* : il détermine comment les nœuds enfants sont générés et sélectionnés à partir d'un nœud parent. Elle est présenté de façon détaillée a la section 3.2

— Paramètres de contrôle de l'algorithme :

  — *La taille minimale de la file* (tmf) : Elle représente la taille initiale de la file pour chaque profondeur. Elle détermine principalement le nombre de nœuds fils qui seront sélectionnés et introduits dans la file à chaque profondeur.

  — *Le facteur de croissance* (fc) : C'est un nombre réel qui nous permet d'incrémenter le taille des collections après chaque itération.

  — *La taille maximale de la file* (tMf) : Elle représente la taille de la file au delà de laquelle l'algorithme s'arrêtera.

  — *Le nombre maximum de nœud* (nMn) :

  — *Le temps limite de calcul* (tm).

Structures de données

— Liste de Solutions : Une collection (sets) contenant les meilleures solutions trouvées pendant la recherche, c'est-à-dire les nœuds les plus prometteurs. Cette liste a une taille fixe et est initialisée avec le nœud racine. Elle est mise à jour chaque fois qu'un meilleur nœud est trouvé. Si un meilleur nœud est découvert et que la liste est pleine, la pire des solutions contenues dans la liste est supprimée et le nouveau nœud est ajouté. Pour une instance de notre problème (input155064901), la solution est représentée sous le format illustré à la figure 3.1.

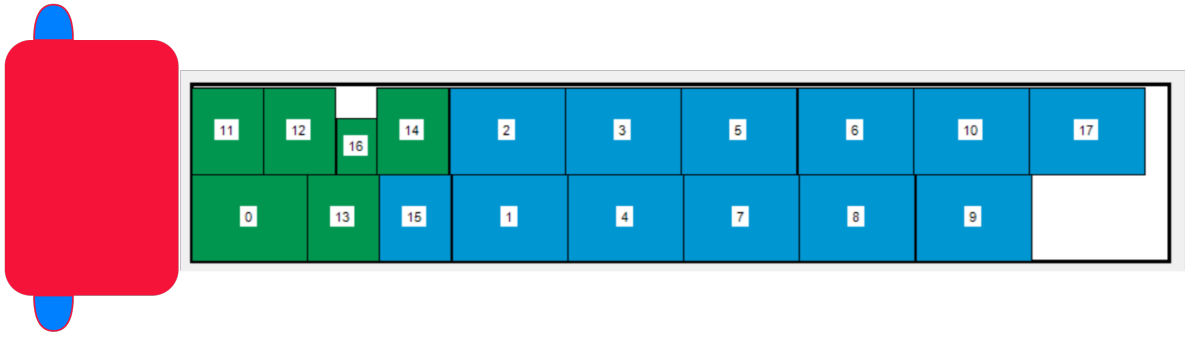
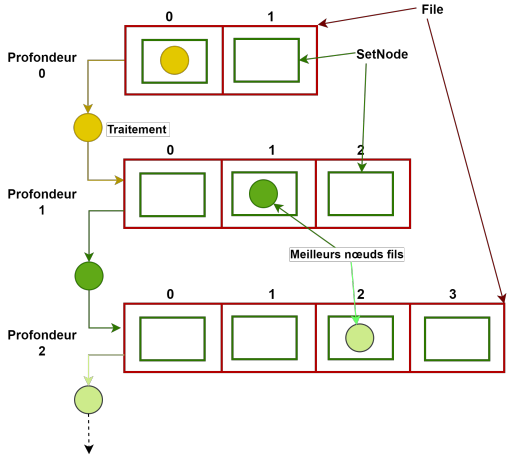


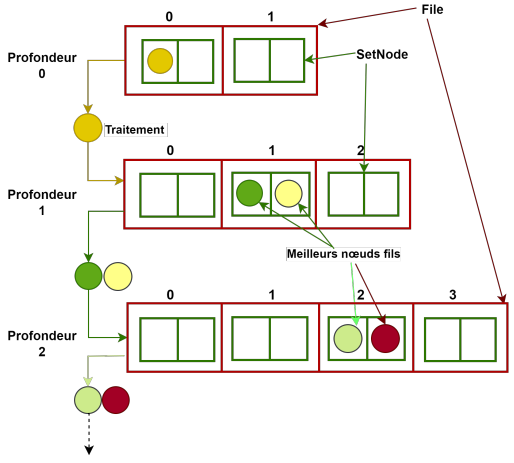
FIGURE 3.1 – Format de solution.

— La File : Une liste de collections de nœuds dont la taille initiale est un nombre de collections arbitraire. Cette taille croît au fur et à mesure que la profondeur dans l'arbre de recherche augmente. Chaque collection a une taille initialisée à tmf, laquelle varie après chaque itération (appel récursif de l'IBS) en fonction du fc. Elle contient les nœuds les plus prometteurs à explorer et correspond à un niveau de profondeur dans l'arbre de recherche. Chaque fois qu'un nœud est exploré, il est supprimé de la collection et ajouté dans l'historique.

Dans la Figure 3.2, pour $fc = 2$,$tmf = 1$ et la taille de la liste de collections initialisée à 2, nous avons dans le premier appel récursif (Figure 3.2a), la taille de la file qui augmente de 1 chaque fois que le niveau de profondeur augmente de 1 tandis que la taille des collections($tmf = 1$) reste fixe et la collection à l'indice de la profondeur courante est traitée. Au deuxième appel récursif (Figure 3.2b), le même scénario se reproduit à la seule différence que la taille des collections passe de 1 à 2 ($tmf \times fc$).



(a) IBS : Premier appel récursif.



(b) IBS : Deuxième appel récursif.

FIGURE 3.2 – Exemple d'état de file entre 2 appels récursifs.

— L'historique : Il contient les nœuds déjà explorés pour éviter de revisiter les mêmes nœuds

et pour gérer la domination entre les nœuds (les nœuds dominés ne sont pas revisités).

— Un nœud : C'est la structure qui contient les détails de solutions de notre problème dans la construction de l'arbre. Une liste complète de ses attributs clés, y compris leurs types, valeurs par défaut et descriptions brèves, peut être trouvée dans le Tableau 3.1.

Attribut	**Type**	**Valeur par défaut**	**Description**
id	int	-1	Identifiant unique du nœud
father	Node	null	Référence au nœud père (null si c'est un nœud racine)
itemTypeId	int		Identifiant du type de l'élément inséré
rotate	boolean		Indique si le dernier élément inséré a été pivoté
x	double		Coordonnée x du coin inférieur gauche du dernier élément inséré
y	double		Coordonnée y du coin inférieur gauche du dernier élément inséré
lastBinDirection	Direction	Direction.X	Direction du dernier bin
uncoveredItems	List UncoveredItem	Tableau vide	Liste des éléments non couverts
itemNumberOfCopies	List Integer	new Tableau vide	Nombre de copies de chaque élément
numberOfBins	int	0	Nombre de bins dans le nœud
numberOfItems	int	0	Nombre d'éléments dans le nœud
guide	int	0	Guide
itemArea	double	0	Surface de l'élément
itemWeight	double	0	Poids de l'élément
currentArea	double	0	Surface actuelle
waste	double	0	Surface perdue
xeMax	double		Maximum xe de tous les éléments dans le dernier bin
xsMax	double	0	Maximum xs de tous les éléments dans le dernier bin
profit	double	0	Profit
middleAxleOverweight	double	0	Surcharge de l'essieu central
rearAxleOverweight	double	0	Surcharge de l'essieu arrière

TABLE 3.1 – Structure d'un nœud

3.1.3 Fonctionnement de l'algorithme

Initialisation

— Création des files et des historiques : L'algorithme commence par initialiser plusieurs files et historiques pour gérer les nœuds à différents niveaux de profondeur.

— Ajout du nœud racine : Le nœud racine, représentant l'état initial du problème, est ajouté à la file de départ. Pour notre problème, le nœud racine représente un camion avec sa remorque vide.

Boucle principale

— Exploration des nœuds : Tant que la file actuelle n'est pas vide, le nœud en tête de file est extrait et exploré.

— Génération des enfants : À partir du nœud courant, les nœuds enfants sont générés en utilisant le schéma de branchement.

— Mise à jour des solutions : Chaque enfant est évalué. Si un enfant améliore la solution actuelle, il est ajouté au pool de solutions.

— Ajout à la file et à l'historique : Les enfants non dominés par un des nœuds de la file et respectant les contraintes de l'algorithme sont ajoutés à la file pour exploration future, en retirant éventuellement le moins bon nœud de la file si celle-ci est déjà pleine.

— Vérification des contraintes : L'algorithme vérifie en permanence les contraintes de temps et le nombre maximum de nœuds explorés pour s'assurer que la recherche reste efficace.

Conditions de fin

— L'algorithme se termine lorsque toutes les files sont vides ou que les contraintes de temps et de nombre de nœuds explorés sont atteintes.

Remarque

Il est possible d'exécuter notre algorithme de manière non itérative. Pour ce faire, nous devons définir une taille de file adéquate correspondante. Cette option est particulièrement utile lorsque nous traitons des instances de très grande taille avec des types très variés. Elle permet de produire des résultats acceptables dans un temps très raisonnable.

3.2 Schéma de branchement

Le schéma de branchement est la structure clé de notre algorithme de recherche arborescente. Il définit les fonctions permettant de déterminer le guide pour la recherche, la stratégie de branchement, la génération des nœuds fils d'un nœud courant, le critère de choix d'un fils par rapport

à un autre, le critère de dominance entre les nœuds, et la stratégie d'insertion des piles dans le camion, entre autres.

3.2.1 Prédécesseurs

Le schéma de branchement initialise une solution pour notre problème de placement de piles dans des camions en utilisant une instance donnée et des paramètres spécifiques. Il joue un rôle crucial dans la préparation des données nécessaires pour résoudre ce problème de placement. Plus précisément, il crée des structures de données pour stocker les prédécesseurs des différents types de piles, essentiels pour déterminer les ordres possibles d'insertion des piles dans les camions. Pour ce faire, on commence par initialiser les listes de prédécesseurs, dont la taille de chaque liste est égale au nombre de types de piles (lignes 2-5 Algo 3). Par la suite, pour chaque objet, en fonction de la stratégie choisie (dominance entre piles), on construit l'ensemble de ses prédécesseurs en comparant les dimensions, les orientations et les groupes des autres piles, afin d'assurer que les piles peuvent être placées de manière optimale tout en respectant les contraintes du problème (lignes 7-18 Algo 3).

Algorithm 3 Algorithme de construction de prédécesseurs

```
1: Entrée : instance, paramètres
2: Initialisation :
3: prédécesseurs ← liste vide pour chaque type de pile
4: prédécesseurs1 ← liste vide pour chaque type pile
5: prédécesseurs2 ← liste vide pour chaque type pile
6: for chaque type de pile i do
7:    for chaque type de pile j do
8:       if i ≠ j et i et j sont du même groupe et j domine i then
9:          if les dimensions de i et j sont égales ou peuvent être inversées mais orientations différentes
            then
10:            Ajouter j à prédécesseurs[i]
11:         else if les dimensions de i et j sont égales, mais orientations différentes then
12:            Ajouter j à prédécesseurs1[i]
13:         else if les dimensions de i et j sont inversées, mais orientations différentes then
14:            Ajouter j à prédécesseurs2[i]
15:         end if
16:      end if
17:   end for
18: end for
```

Cet algorithme permet de mettre en relation les différents types de piles, un aspect essentiel pour prendre des décisions de placement optimales. Les informations sur les prédécesseurs aident à éviter les configurations sous-optimales en plaçant les objets dans un ordre qui respecte les contraintes de dimension et de groupe, contribuant ainsi à une solution plus efficace.

3.2.2 Dominance

La fonction de dominance compare deux objets pour déterminer si l'un "domine" l'autre selon des règles spécifiques. La dominance ici signifie que, selon certains critères, un objet est considéré comme meilleur ou plus prometteur que l'autre. La comparaison peut se faire entre deux piles ou entre deux nœuds.

Dominance entre piles

Les critères pris en compte ici sont le profit et le poids.

— Dominance par rapport au profit : On considère qu'une pile 1 domine une pile 2 si le profit de la pile 1 est supérieur ou égal à celui de la pile 2. Cette stratégie est conçue pour les situations où les contraintes de poids sur les essieux ne sont pas actives.

— Dominance par rapport au poids : On considère qu'une pile 1 domine une pile 2 si le poids de la pile 1 est inférieur ou égal à celui de la pile 2. Cette stratégie est adaptée aux cas où la contrainte de poids sur l'essieu central est active.

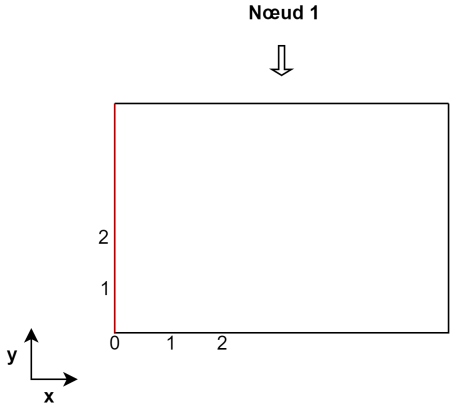
Dominance entre nœuds

Les critères pris en compte ici sont les poids des essieux, les coordonnées x et y des éléments non couverts d'un nœud, et la dominance x de ces éléments non couverts ou la surface gaspillée dans le camion.

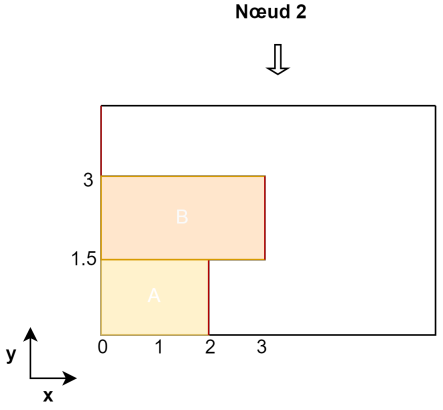
— Dominance par rapport aux poids sur les essieux : Les vérifications assurent qu'un nœud ($node_1$) n'a pas un poids excessif sur les essieux par rapport à un autre nœud ($node_2$), selon des paramètres spécifiques au problème comme la valeur du guide (VG). Par exemple, si $VG = 1$, la fonction de dominance retourne $false$ si le poids sur l'essieu médian du dernier camion de $node_1$ est strictement supérieur à celui du camion de $node_2$, et retourne $true$ pour l'essieu arrière si $VG = 2$.

— Éléments non couverts (ENC 3.1.2) : Un élément non couvert d'un nœud est un objet dans le camion, chargé ou non, qui définit une position candidate où une pile peut être placée. Cet objet peut être partiellement ou entièrement la largeur du camion ou pile . Un objet non couvert est représenté par un tuple (id, xs, xe, ys, ye), définissant une zone disponible (l'extrémité droite de l'élément non couvert). Ici, xs et ys indiquent les coordonnées de l'origine de l'élément non couvert, et xe et ye indiquent les coordonnées de l'extrémité. L'identifiant d'un élément non couvert est déterminer par rapport à sa position dans le camion et croit avec la coordonnée y. Si l'élément non couvert est une partie du bord du camion (fond du conteneur), son identifiant a pour valeur -1. Dans la figure 3.3, nous avons par exemple, pour le nœud 1 (Fig 3.3a), un camion vide qui contient un seul élément non couvert qui est le bord du camion, avec les caractéristiques $(-1, 0, 0, 0, TW_t)$. Pour le nœud 2, nous avons un camion non vide qui contient trois éléments non couverts : une partie de la largeur du camion caractérisée par $(-1, 0, 0, SW_A + SW_B, TW_t - (SW_A + SW_B))$, la pile

A caractérisée par $(0, 0, 2, 0, SW_A)$, et la pile B caractérisée par $(1, 0, 3, SW_A, SW_A + SW_B)$. Pour chaque élément non couvert, une position candidate pour l'insertion d'une pile doit être situé sur son extrémité droite (ligne rouge).



(a) Camion vide



(b) Camion non vide

FIGURE 3.3 – Exemple d'éléments non couverts dans un nœud.

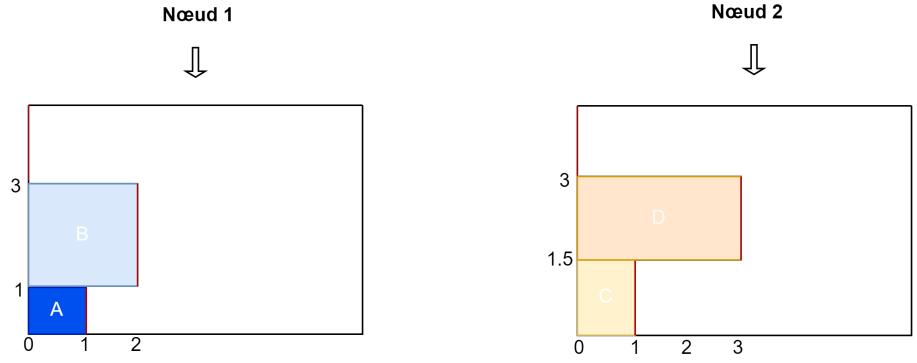
Une boucle compare les coordonnées x des éléments non couverts en tenant compte de leur dominance. Les coordonnées x de l'extrémité droite de ces éléments non couverts sont stockées dans x_1 et x_2. Les positions des derniers éléments non couverts de $node_1$ et $node_2$ sont stockées dans pos_1 et pos_2.

- Si $x_1 > x_2$, $node_1$ ne domine pas $node_2$.
- Si les positions y (ys) des éléments non couverts sont égales, les indices et les coordonnées x de dominance sont mis à jour.
- Si ys de $node_1$ est inférieur à ys de $node_2$, l'indice de $node_2$ est décrémenté et sa coordonnée x de dominance est mise à jour.
- Si ys de $node_1$ est supérieur à ys de $node_2$, l'indice de $node_1$ est décrémenté et sa coordonnée x de dominance est mise à jour.

— La boucle continue jusqu'à ce que les deux indices soient zéro, ce qui signifie que tous les éléments non couverts ont été comparés et $node_1$ domine $node_2$.

Pour illustrer un exemple de dominance entre deux nœuds, considérons la figure 3.4. Nous avons le nœud 1 (Fig 3.4a) qui possède trois éléments non couverts : le bord du camion $(-1, xs = xe = 0, ys = sy^e_B = 3, ye = TW_t)$, la pile B $(1, xs = sx^o_B = 0, xe = sx^e_B = 2, ys = sy^o_B = 1, ys = sy^e_B = 3)$, et la pile A $(0, xs = sx^o_A = 0, xe = sx^e_A = 1, ys = sy^o_A = 0, ye = sy^e_A = 1)$. Le nœud 2 (Fig 3.4b) possède également trois éléments non couverts : le bord du camion $(-1, xs = xe = 0, ys = sy^e_B = 3, ye = TW_t)$, la pile D $(1, xs = sx^o_D = 0, xe = sx^e_D = 3, ys = sy^o_D = 1.5, ye = sy^e_D = 3)$, et la pile C $(0, xs = sx^o_C = 0, xe = sx^e_C = 1, ys = sy^o_C = 0, ye = sy^e_C = 1.5)$.

Dans cet exemple, nous avons au départ $pos_1 = pos_2 = 2$. L'élément non couvert à la position 2 dans les deux nœuds est le bord du camion $(sxs = sx^e = 0, sy^o = 3)$. Nous avons donc $y_1 = y_2 = sy^o$. On met à jour les indices et les coordonnées x de dominance pour les deux nœuds. Nous avons donc $pos_1 = pos_2 = 1$. L'élément non couvert à la position 1 dans le nœud 1 est la pile B, donc $x_1 = sx^e_B = 2, y_1 = sy^o_B = 1$, et l'élément non couvert à la position 1 dans le nœud 2 est la pile D, donc $x_1 = sx^e_D = 3, y_2 = sy^o_D = 1.5$. Nous avons $y_1 < y_2$, on décrémente pos_2 et on met à jour la coordonnée x_2. On a donc $pos_2 = 0$ et $x_2 = sx^e_C = 1, y_2 = sy^o_C = 0$. À ce stade, nous avons $y_2 < y_1$, donc on décrémente pos_1 et on met à jour la coordonnée x_1. On a donc $pos_1 = 0$ et $x_1 = sx^e_A = 1, y_1 = sy^o_A = 0$. Nous avons donc $pos_1 = pos_2 = 0$. Cela signifie que tous les éléments non couverts dans les deux nœuds ont été parcourus, donc le nœud 1 domine le nœud 2.

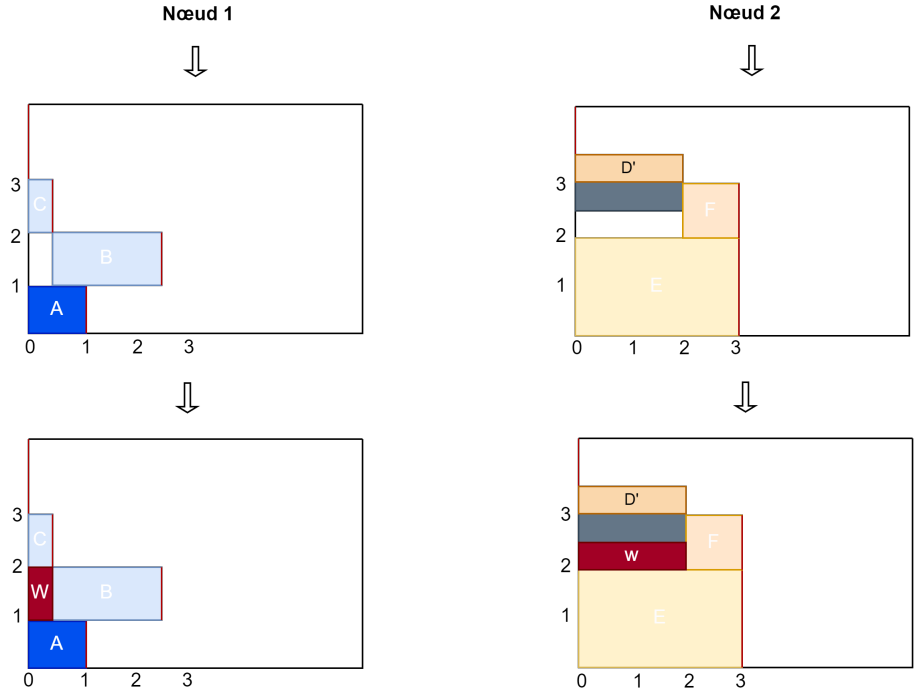


(a) (b)

FIGURE 3.4 – Exemple de dominance entre 2 nœuds.

— La surface gaspillée dans le camion : Cette métrique représente la différence/ratio entre la surface utilisée dans le camion et la somme des aires de toutes les piles chargées dans le camion. La surface utilisée dans le camions se calcule a l'aide des éléments non couvert. c'est la somme des $sx^e \times (sy^e - sy^o)$ de tous les éléments non couverts. Pour illustrer un exemple pour ce critère entre deux nœuds, considérons la figure 3.5. Nous avons le nœud 1 (Fig 3.5a) qui possède quatre éléments non couverts : le bord du camion $(-1, xs = xe = 0, ys = 3, ye = TW_t)$, la pile

C $(2, 0, SL_C = 0.5, sy^o_C = 2, sy^e_C = 3)$, la pile B $(1, sx^o_B = 0.5, sx^e_B = 2.5, sy^o_B = 1, sy^e_B = 2)$, et la pile A $(0, sx^o_A = 0, sx^e_A = 1, sy^o_A = 0, sy^e_A = 1)$. Le nœud 2 (Fig 3.5b) possède également quatre éléments non couverts : le bord du camion $(sx^o = sx^e = 0, sy^o = 3.5)$, la pile D′ $(SL_{D'} = 2, SW_{D'} = 0.5, sx^o_{D'} = 0, sx^e_{D'} = 2, sy^o_{D'} = 3, sy^e_{D'} = 3.5)$ une fraction de la pile D $(SL_D = 2, SW_D = 1, sx^o_D = 0, sx^e_D = 2, sy^o_D = 2.5, sy^e_D = 3.5)$ qui est le nouvel élément non couvert après le chargement de la pile F dans le camion, la pile F $(SL_F = 1, SW_F = 1, sx^o_F = 2, sx^e_F = 3, sy^o_F = 2, sy^e_F = 3)$, et la pile E $(SL_E = 3, SW_E = 2, sx^o_E = 0, sx^e_E = 3, sy^o_E = 0, sy^e_E = 2)$. Pour le nœud 1, la somme des aires des piles est égale à 3.5 et la surface utilisée est égale à 4 $(sx^e_A \times SW_A + sx^e_B \times SW_B + sx^e_C \times SW_C)$. Pour le nœud 2, la somme des aires des piles est égale à 9 et la surface utilisée est égale à 10 $(sx^e_E \times SW_E + sx^e_F \times SW_F + sx^e_{D'} \times SW_{D'})$. Nous obtenons donc que la surface gaspillée (rectangle rouge W Fig 3.5) pour le nœud 1 et nœud 2 est respectivement de 0.5 et 1. Le nœud 1 est ainsi meilleure que le nœud 2.



(a) (b)

FIGURE 3.5 – Exemple de surface gaspillée entre 2 nœuds.

3.2.3 Génération de nœuds fils

Chaque nœud représente un état possible dans une solution d'un problème d'optimisation, tel que le placement des piles dans un camion. La construction des nœuds fils à partir d'un nœud parent

se fait principalement en deux étapes. Tout d'abord, une liste d'insertions valides est fabriquée. Ensuite, pour chaque insertion valide, on génère le nœud fils correspondant, dont les caractéristiques sont construites à partir de celles du nœud parent et de l'insertion valide.

Insertion

Il est important de déterminer des placements valides des piles au sein d'un nœud donné de la structure arborescente de branchement. A cet effet, nous évaluons systématiquement chaque type d'article par rapport à ses prédécesseurs pour garantir que l'insertion respecte les contraintes nécessaires, telles que le nombre de copies déjà placées et leur orientation. En vérifiant la faisabilité d'insérer des articles à la fois dans les bacs existants et dans de nouveaux bacs, nous construisons une liste d'insertions potentielles qui peuvent être utilisées dans les étapes suivantes du processus d'optimisation. Cette capacité à explorer diverses configurations est essentielle pour optimiser l'utilisation de l'espace et respecter les contraintes de poids et de dimensions, contribuant ainsi à trouver une solution efficace pour le problème de placement.

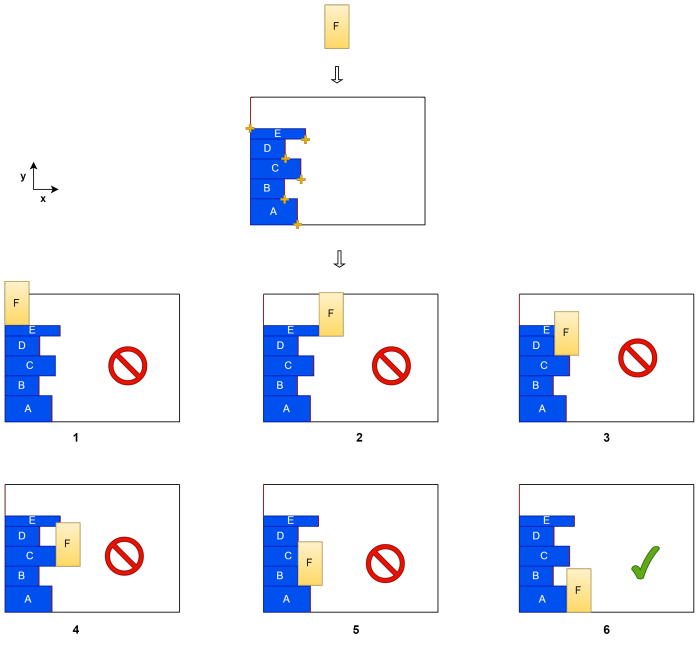
Une insertion est un objet caractérisé par un tuple $(id, rotation, nb, x, y)$. Ici id est l'identifiant de la pile associée a l'insertion, $rotation$ est la valeur booléenne qui permet de déterminer si la pile a été tournée ou pas, nb permet de déterminer dans quel camion a été fait l'insertion (nouveau camion ou camion courant) et x, y représentent les coordonnées du point origine de l'insertion. Partant d'un nœud parent (configuration du dernier camion dans le nœud), on vérifie la possibilité d'insérer une pile donnée dans cette configuration, en respectant les contraintes de piles (section 2.2.4) et de poids (section 2.2.4) mais aussi par rapport à ses prédécesseurs pour garantir que l'insertion respecte les contraintes nécessaires, telles que le nombre de copies déjà placées et leur orientation. Pour ce faire, nous commençons par récupérer les informations sur le type de pile et de camion, puis on calcule les dimensions nécessaires en fonction de leur orientation (lignes 2-10 Algo 4). Par la suite, nous validons la faisabilité de l'insertion en vérifiant les contraintes de hauteur et de poids du conteneur, et en déterminant la position horizontale initiale pour éviter les collisions avec d'autres piles (lignes 11-19 Algo 4). En fin nous vérifions que la pile respecte les limites de longueur et les contraintes spécifiques supplémentaires (lignes 19-24 Algo 4). Si toutes les conditions sont remplies, l'insertion de la pile est ajoutée à la liste des insertions possibles (lignes 25-28 Algo 4). Si, pour un nœud donné, aucune insertion valide n'est trouvée pour tous les types de piles, cela signifie que le camion correspondant ne peut plus recevoir de piles et qu'un nouveau camion est nécessaire.

Pour illustrer un exemple de création d'une insertion , considérons la figure 3.6. Dans cet exemple, nous supposerons que les contraintes de poids sont respectées dans tous les cas. Nous avons la pile F avec une orientation forcée (la longueur de la pile parallèle à la largeur du camion) qui doit être insérée dans le camion contenant les piles A à E, et six objets non couverts : la partie supérieure du camion et les cinq piles (Fig 3.6a). Le point d'origine de chaque position candidate pour l'insertion de la pile F est identifié par les signes plus jaunes (un signe pour chaque objet non couvert). En positionnant la pile F sur chaque position valide (contraintes de hauteur), nous constatons que les cas 1 à 5 produisent des insertions non valides. Les cas 1 et 2

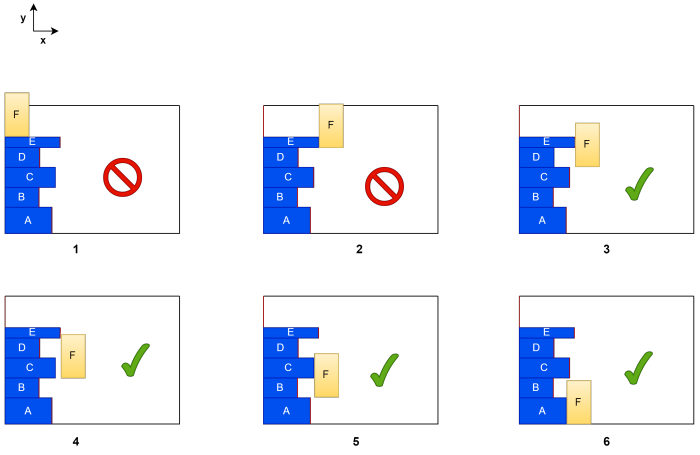
Algorithm 4 Algorithme de création d'insertions

Entrée : instance, nœud
insertionV ← []
itemType ← instance.pile
binTypeId ← camion.id
binType ← Camion[binTypeId]
orientation ← binType.Orientation
(xj, yj) ← récupérer les dimensions de l'item selon itemType et orientation
(xi, yi) ← récupérer les dimensions du camion selon binType et direction
ys ← position de départ verticale selon le position de l'objet non couvert
ye ← ys + yj
if ye > yi **then**
    fin
    lastBinWeight ← camion.poids
    if lastBinWeight > camion.poidsmax **then**
        fin
        xs ← 0
        if camion non vide **then**
            ajuster xs pour éviter la collision avec les objets non couverts
        end if
        xe ← xs + xj
        if xe > xi **then**
            fin
        end if
    end if
end if
insertion ← créer une nouvelle instance d'Insertion
mettre à jour les propriétés de insertion
ajouter insertion à insertionV

violent les contraintes de dimensions (largeur du camion) et les cas 3 à 5 violent la contrainte de chevauchement entre les piles. Par contre, le cas 6 respecte toutes les contraintes et génère donc une insertion valide (Fig 3.6a). En ajustant la position horizontale initiale pour chaque position initiale afin d'éviter les collisions avec d'autres piles, les cas 1 et 2 restent invalides tandis que les cas 3 et 4 deviennent valides, en plus du cas 6 qui l'était déjà (Fig 3.6b). Finalement, l'ajout de la pile F nous produit quatre insertions valides.



(a)



(b)

FIGURE 3.6 – Exemple de création d'insertion.

nœuds fils

Une fois les insertions valides créées, nous passons à la construction des nœuds fils (cf Algo 5). Nous initialisons un nouveau nœud en copiant les propriétés de son père et en assignant les valeurs spécifiques de l'insertion (type de pile, rotation, position). Si un nouveau camion est nécessaire (l'insertion se fait dans un nouveau camion), le nombre de camions est incrémenté, et la direction du nouveau camion est mise à jour en fonction de l'orientation de l'insertion. Sinon, les valeurs du camion actuel sont conservées. Les dimensions et positions de la pile insérée sont ensuite calculées, puis les objets non couverts sont mis à jour en ajoutant ou modifiant les espaces vides créés par l'insertion (cf. Algo 6). Les propriétés d'insertion, telles que les copies de piles, le nombre de piles, la surface, le poids, le profit et le score de groupe, sont mises à jour. Les poids des camions pour chaque groupe concerné sont ajustés en conséquence. La surface occupée par les piles est calculée, vérifiée et ajustée si nécessaire. Un identifiant unique est alors attribué au nœud, et le nœud fils est retourné. Si les contraintes sur la charge aux essieux sont activées, les valeurs du poids sur les essieux du camion sont mises à jour et vérifiées pour s'assurer qu'elles ne sont pas violées avant de générer le nœud fils.

Algorithm 5 Algorithme de construction des nœuds fils

```
1: Entrée : nœud-père, insertion
2: Initialiser nœud fils avec les propriétés du père
3: Attribuer type de pile, rotation, position de l'insertion au fils
4: if nouveau camion then
5:     camion++
6:     camion.orientation ←insertion.orientation
7: end if
8: Calculer les dimensions et positions de l'objet inséré
9: Mettre à jour les objet non couverts
10: Mettre à jour les propriétés d'insertion
11: Ajuster le poids du camion
12: Calculer la surface occupée par les piles
13: Mettre à jour les valeurs du poids sur les essieux du camion
14: if contraintes de charge aux essieux vérifiées then
15:     Attribuer un identifiant unique au nœud
16: end if
17: Retourner le nœud fils mis à jour
```

Pour un nouveau nœud fils, nous mettons à jour les objets non couverts de ce nœud en fonction des paramètres de l'insertion et du camion suivant l'algorithme 6. Si un nouveau camion a été nécessaire, nous ajoutons un nouvel élément non couvert en bas à gauche du nouveau camion si la coordonnée ys de l'insertion est supérieure à zéro. Ensuite, nous ajoutons un nouvel élément non couvert représentant l'espace occupé par l'élément inséré. Si l'extrémité y de l'insertion (ye) est inférieure à la largeur du camion (TW_t), nous ajoutons un autre élément non couvert en haut de l'élément inséré (lignes 2-9 Algo 6). Si un nouveau camion n'a pas été nécessaire, nous itérons sur les éléments non couverts du nœud parent de ce nœud fils. Pour chaque élément non couvert

de ce parent, nous déterminons s'il doit être conservé ou modifié en fonction de ses coordonnées par rapport à l'élément inséré. Les éléments non couverts qui ne chevauchent pas l'élément inséré sont copiés tels quels dans le nœud fils. Ceux qui chevauchent l'élément inséré sont divisés en nouvelles parties non couvertes, selon leurs positions relatives à l'élément inséré (lignes 11-30 Algo 6). Nous assurons ainsi que tous les espaces restants non couverts après l'insertion de l'élément sont correctement mis à jour et enregistrés.

Algorithm 6 Algorithme de mise à jour des éléments non couverts

```
1: Entrée : nœud, insertion
2: if si le camion est vide then
3:    if ys de l'insertion > 0  then
4:       ENC ← (-1, 0, 0, 0, ys) et listeENC.add(ENC)
5:    end if
6:    ENC←insertion et listeENC.add(ENC)
7:    if ye de l'insertion < largeur du camion then
8:       ENC ←(-1, 0, 0, ye, yi) et listeENC.add(ENC)
9:    end if
10: else
11:    for chaque ENC dans nœud parent do
12:       if ENC.ye ≤ ys then
13:          newENC ← ENC et listeENC.add(newENC)
14:       else if ENC.ys ≤ ys then
15:          if ENC.ys < ys then
16:             newENC ← ENC et newENT.ye ← ys et listeENC.add(newENC)
17:          end if
18:          newENC2 ← insertion et listeENC.add(newENC)
19:          if ENC.ye > ye then
20:             newENC ← ENC et newENT.ys ← ye et listeENC.add(newENC)
21:          end if
22:       else if ENC.ys ≥ ye then
23:          newENC ← ENC et listeENC.add(newENC)
24:       else
25:          if ENC.ye > ye then
26:             newENC ← ENC et newENT.ys ← ye et listeENC.add(newENC)
27:          end if
28:       end if
29:    end for
30: end if
```

Pour illustrer un exemple, prenons le cas de la section 4. Pour le placement de la pile F, l'algorithme d'insertion a généré quatre insertions valides. À partir de ces insertions valides, nous construisons un nœud fils pour chaque insertion en mettant à jour les objets non couverts en fonction des paramètres de l'insertion et du camion, suivant l'algorithme 6. Ici, nous considérons que la contrainte de poids sur les essieux n'est pas activée. Pour chacune de ces insertions valides (3.6), nous obtenons les quatre nœuds fils illustrés dans la figure 3.7.

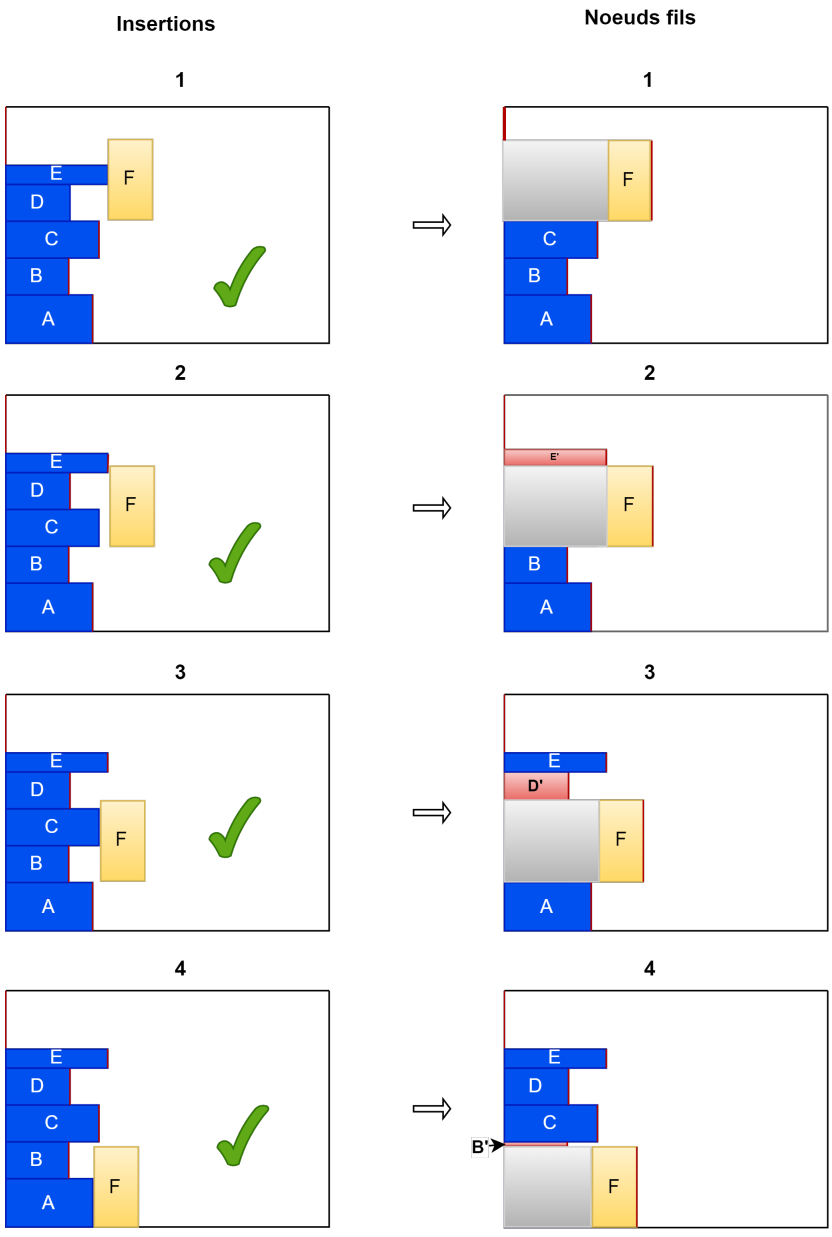


FIGURE 3.7 – nœuds fils.

Le nœud fils numéro 2 a désormais cinq éléments non couverts (cf Fig 3.8), dont trois éléments provenant du nœud parent, à savoir : le bord du camion $(-1, 0, 0, sy^e_E, TW_t)$, la pile B $(1, 0, sx^e_B, sy^o_B, sy^e_B)$ et la pile A $(0, 0, sx^e_A, 0, sy^e_A)$. Les deux nouveaux éléments non couverts sont : la pile E', une portion de la pile E $(3, sx^o_E, sx^e_E, sy^e_F, sy^e_E)$, et la pile F $(2, sx^o_F, sx^e_F, sy^o_F, sy^e_F)$.

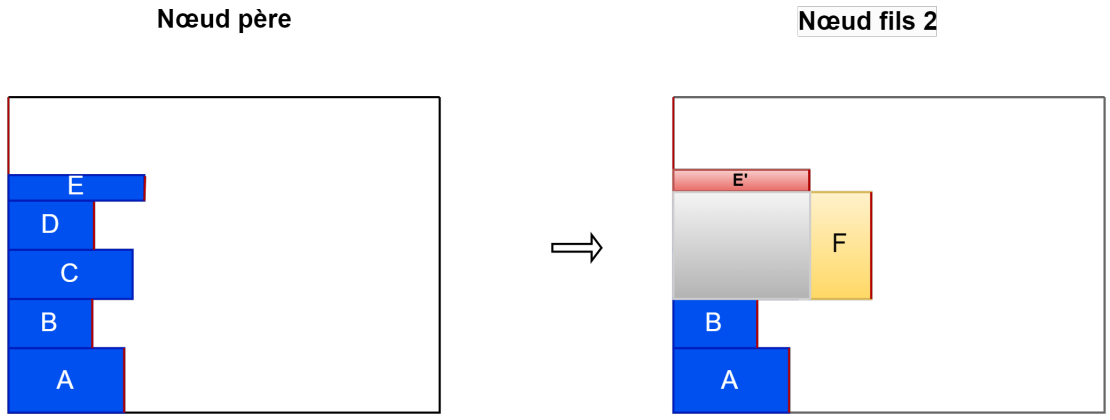


FIGURE 3.8 – nœud parent et nœud fils.

3.3 Conclusion

Ce chapitre a été consacré à la résolution de notre problème de BP2D en se concentrant sur les algorithmes heuristiques et métaheuristiques. Nous avons décrit en détail le fonctionnement de notre algorithme proposé, en expliquant les concepts et les structures de données employés. Le schéma de branchement a été présenté, avec une attention particulière aux prédécesseurs, à la dominance et à la génération de nœuds fils. Enfin, nous avons abordé l'implémentation de l'algorithme et évalué son efficacité en termes de temps de calcul.

Chapitre 4

Résultats

Dans ce chapitre, nous présentons une comparaison des résultats de notre méthode sur les jeux de données disponibles. Nous comparons également la méthode actuelle avec l'algorithme de l'outil interne. De plus, nous évaluons la méthode en utilisant différents modes d'exécution (itératif ou non itératif). Enfin, nous discutons des résultats observés et fournissons une analyse détaillée de leurs implications.

4.1 Instances

Une instance est représentée par un fichier : inputitems.txt. Ce fichier est au format CSV (valeurs séparées par des virgules). Un fichier inputitems.txt contient toutes les informations relatives au camion ainsi qu'à l'ensemble de piles à charger dans ce camion. Le descriptif d'une instance de notre jeu de données est présenté dans le tableau 4.2. Pour nos expériences numériques, nous disposons de deux ensembles de données décrits dans le tableau4.1.

Nom	**Nombre instances**	**Nombre de piles**
OyakMont	2595	44625
Sovab	7130	97005

TABLE 4.1 – Jeu de donnée

Camion	**ID**	**longueur**	**largeur**	**hauteur**	**poids max autorisé**	
	paramètres pour le calcul du poids sur les essieux					
Pile	**ID**	**longueur**	**largeur**	**poids**	**orientation**	**groupe**

TABLE 4.2 – Descriptif d'une instance

4.2 Résultats

Appel itératif

Le tableau 4.3 présente les résultats obtenus en exécutant notre algorithme sur notre jeu de données Sovab (tableau 4.1) en mode itératif. Ici, nous avons comme valeurs de paramètres : $cf = 1.5$, $tmf = 2$. Nous faisons varier la taille maximale de la file et nous comparons les résultats. Cette comparaison est axée sur le temps de calcul et le nombre de camions utilisés.

tMf	Temps (ms)	Nombre de camions
5	26,668	9,927
10	26,695	9,771
20	44,201	9,697
25	57,700	9,706
50	75,475	9,661
100	203,819	9,659
500	916,164	9,655

TABLE 4.3 – Résultats de l'algorithme itératif en fonction de la taille maximale de la file

Le tableau 4.3 montre que, dans le mode itératif, le temps de calcul augmente de manière non linéaire avec l'augmentation de la taille maximale de la file. Par exemple, pour une taille de file de 5, le temps de calcul est de 26,668 ms, alors qu'il passe à 916,164 ms pour une taille de file de 500. Cela montre que la complexité de l'algorithme augmente significativement avec la taille de la file, ce qui est attendu, car une plus grande file permet à l'algorithme d'explorer un plus grand nombre de configurations possibles.

Cependant, le nombre de camions requis diminue légèrement avec l'augmentation de la taille de la file, passant de 9,927 camions pour une taille de file de 5 à 9,655 pour une taille de 500. Cette réduction, bien que modeste, indique que l'algorithme parvient à mieux optimiser l'utilisation des camions à mesure qu'il dispose de plus de configurations à analyser.

Appel non itératif

Le tableau 4.4 présente les résultats obtenus en exécutant notre algorithme sur notre jeu de données Sovab (tableau 4.1) en mode non récursif. Ici, la taille de la file est fixe ($tmf = tMf$), et nous comparons les résultats pour différentes valeurs de la taille de la file. Cette comparaison est axée sur le temps de calcul et le nombre de camions utilisés.

Taille de la file	Temps (ms)	Nombre de camions
1	30,009	10,955
5	27,668	10,021
10	26,695	9,863
20	27,217	9,758
25	28,843	9,731
50	43,792	9,714
100	77,728	9,682
500	266,121	9,660

TABLE 4.4 – Résultats de l'algorithme non itératif en fonction de la taille de la file

Le tableau 4.4 présente les résultats obtenus en mode non itératif, où la taille de la file est fixe. Ici aussi, on observe une augmentation du temps de calcul avec l'augmentation de la taille de la file, bien que la progression soit moins rapide que dans le mode itératif. Par exemple, pour une taille de file de 1, le temps est de 30,009 ms, alors qu'il atteint 266,121 ms pour une taille de file de 500.

En termes de nombre de camions, le mode non itératif montre également une réduction progressive avec l'augmentation de la taille de la file. Cependant, cette diminution devient marginale au-delà d'une taille de file de 100, où l'on observe un point de rendement décroissant. Cela suggère qu'au-delà de ce point, l'ajout de configurations supplémentaires dans la file n'apporte pas de gains significatifs en termes de réduction du nombre de camions, alors que le temps de calcul continue d'augmenter de manière disproportionnée.

Les résultats des deux modes indiquent que l'augmentation de la taille de la file permet à l'algorithme d'explorer plus de configurations, réduisant ainsi le nombre de camions nécessaires. Toutefois, cette réduction s'accompagne d'une augmentation significative du temps de calcul, particulièrement marquée dans le mode itératif. Pour une taille de file d'environ 100, dans les deux modes, on observe un bon compromis entre la réduction du nombre de camions et le temps de calcul. Au-delà de cette taille, les bénéfices en termes de nombre de camions deviennent marginaux, tandis que le coût en termes de temps de calcul augmente de manière exponentielle. Cela indique qu'il existe un point de rendement décroissant autour d'une taille de file de 100, au-delà duquel l'amélioration en efficacité ne justifie plus l'augmentation du temps de calcul.

Comparaison avec l'algorithme existant

Cette section compare les performances de l'algorithme développé dans ce projet à celles de l'algorithme actuellement utilisé chez Renault.

Les résultats des tests réalisés pour nos deux algorithmes, en utilisant nos deux jeux de données décrits dans le tableau 4.1, sont présentés dans le tableau 4.5 et le tableau 4.6. Les comparaisons

portent sur plusieurs critères, notamment le temps de calcul, le nombre total de camions (NTC), la somme totale de la longueur des camions utilisée (ML), et la moyenne de cette dernière (MLM).

Critère	Algorithme interne	Algorithme développé
Temps (ms)	91,968	75,475
NTC	9,674	9,661
ML(mm)	87,911,639	88,000,350
MLM(mm)	9,087	9,426

TABLE 4.5 – Résultats pour l'algorithme interne et l'algorithme développé sur le jeu de données Sovab

Critère	Algorithme interne	Algorithme développé
Temps (ms)	34,000	360,475
NTC	3,395	3,393
ML(mm)	32,262,162	28,724,004
MLM(mm)	10,975	8,465

TABLE 4.6 – Résultats pour l'algorithme interne et l'algorithme développé sur le jeu de données OyakMont

Dans chacun des deux tableaux précédents, le premier critère analysé est le temps de calcul. L'algorithme développé montre un temps de calcul de 75,475 ms, contre 91,968 ms pour l'algorithme interne sur le jeu de données Sovab. Cette réduction d'environ 17,9% souligne une optimisation non négligeable. Cette amélioration peut s'expliquer par la gestion optimisée des structures de données, qui permet une meilleure performance sur des instances de taille similaire.

Concernant le nombre total de camions utilisés (NTC), l'algorithme développé montre une légère amélioration avec 9,661 camions contre 9,674 pour l'algorithme interne sur Sovab, et 3,393 contre 3,395 sur OyakMont. Bien que cette réduction soit marginale, elle peut avoir un impact positif sur les coûts logistiques en réduisant le nombre de trajets nécessaires et les frais associés au transport. Cette diminution, bien que légère, reflète une amélioration de l'efficacité dans la gestion des ressources disponibles.

Cependant, les résultats pour les mètres linéaires (ML) et la moyenne des mètres linéaires (MLM) révèlent des tendances différentes entre les deux jeux de données. Sur Sovab, l'algorithme développé utilise légèrement plus de mètres linéaires (88,000,350 mm contre 87,911,639 mm) et présente un MLM supérieur (9,426 mm contre 9,087 mm). Cette légère augmentation suggère que l'algorithme développé exploite mieux l'espace disponible dans les camions, bien que cela entraîne une utilisation légèrement plus élevée des ressources en termes de longueur totale.

À l'inverse, sur le jeu de données OyakMont, l'algorithme développé utilise moins de mètres linéaires (28,724,004 mm contre 32,262,162 mm), ce qui indique une meilleure optimisation de l'espace de chargement, mais avec un temps de calcul considérablement plus long (360,475 ms contre 34,000 ms). Ce résultat souligne l'efficacité de l'algorithme développé en termes de minimisation de l'espace utilisé, mais également un compromis en termes de temps de calcul, particulièrement pour des instances plus complexes ou plus volumineuses.

4.3 Conclusion

Les résultats obtenus montrent que l'algorithme développé a le potentiel de surpasser l'algorithme existant en termes de certains critères d'optimisation, notamment la réduction du nombre de camions utilisés et l'amélioration de l'utilisation de l'espace disponible. Cependant, le coût en temps de calcul reste une considération importante, surtout dans des contextes industriels où la rapidité des opérations est cruciale. Les perspectives d'amélioration incluent l'optimisation du temps de calcul pour les instances où l'algorithme développé montre des performances de temps plus faibles.

Chapitre 5

Conclusion et perspectives

La chaîne logistique au sein des entreprises est devenue un enjeu crucial pour maintenir la compétitivité sur le marché mondial. Ce document présente une étude approfondie du problème de placement en deux dimensions (BP2D) dans le contexte industriel de Renault, en se concentrant sur l'optimisation du chargement des camions et conteneurs. L'étude s'inscrit dans le cadre de la gestion des flottes de camions pour le transport des pièces de véhicules de différents fournisseurs vers les usines. À travers ce travail, nous avons exploré diverses facettes du problème de placement en deux dimensions (BP2D) et ses variantes. Nous avons conclu que les approches exactes, bien qu'efficaces pour des instances de petite taille, deviennent souvent impraticables pour des instances plus grandes en raison des contraintes de temps de calcul. Les approches heuristiques et métaheuristiques se sont avérées plus adaptées, permettant de trouver des solutions de haute qualité dans des délais raisonnables, comme le montre notre revue exhaustive de la littérature existante. Sur cette base, nous avons développé une méthode heuristique pour résoudre le problème de placement en deux dimensions en tenant compte des contraintes spécifiques telles que le placement des piles, l'absence de chevauchement entre les piles, la charge maximale sur les camions, et le poids sur les différents essieux. Un algorithme de recherche arborescente a été mis en place pour traiter ce problème, ce qui a permis d'obtenir des résultats satisfaisants sur des instances de taille considérable. Cependant, il est important de noter que la contrainte de charges aux essieux n'a pas été prise en compte dans notre algorithme, car notre attention s'est principalement portée sur le problème du placement en 2D. Dans le nouvel algorithme développé, l'intégration de cette contrainte est envisageable, mais elle présente des défis spécifiques. En effet, la contrainte des charges sur les essieux peut être temporairement violée au début ou au cours du processus de chargement, mais elle doit être respectée à la fin, une fois l'équilibre final atteint. La principale difficulté réside dans la nécessité de prévoir et de gérer dynamiquement cette contrainte tout au long du processus de chargement, afin de garantir qu'elle soit respectée en fin de processus, même si elle est momentanément enfreinte. Cela requiert une approche algorithmique plus complexe, capable de modéliser non seulement la répartition finale des charges, mais aussi d'anticiper et de corriger les déséquilibres potentiels au fur et à mesure du chargement. En conclusion, bien que les résultats obtenus soient prometteurs, ce travail ouvre la voie à de futures recherches visant à affiner la méthode proposée afin qu'elle puisse prendre en compte toutes les contraintes spécifiques au problème. Une approche en trois dimensions

pourrait être envisagée pour mieux gérer le problème, c'est-à-dire aborder le bin packing en 3D. En effet, placer les différentes piles au fur et à mesure de leur construction pourrait améliorer la qualité de la solution et permettre de mieux contrôler certaines contraintes, telles que le poids sur les essieux.

Bibliographie

[1] Société française de recherche opérationnelle et d'aide à la décision. `https://www.roadef.org/`.

[2] Ranga P. Abeysooriya, Antonio. Martinez-Sykora, and Julia A Bennell. Jostle heuristics for the 2d-irregular shapes bin packing problems with free rotation. *International Journal of Production Economics*, 195 :12–26, 2018.

[3] B S Baker, E G Co1man Jr., and R L Rivest. Orthogonal packing in two dimensions. *SIAM Journal on Computing*, 9 :846–855, 1980.

[4] J.O. Berkey and P.Y. Wang. Two-dimensional finite bin packing algorithms. *J. Oper. Res. Soc.*, 38 :423–429, 1987.

[5] Guadalupe Carmona-Arroyo, Jenny Betsabé Vázquez-Aguirre, and Marcela Quiroz. *One-Dimensional Bin Packing Problem : An Experimental Study of Instances Difficulty and Algorithms Performance*. March 2021. URL `https://www.researchgate.net/publication/350365638_One-Dimensional_Bin_Packing_Problem_An_Experimental_Study_of_Instances_Difficulty_and_Algorithms_Performance`. Accessed : 2024-06-25.

[6] F K R Chung, M R Garey, and D S Johnson. On packing two-dimensional bins. *SIAM Journal on Algebraic and Discrete Methods*, 3 :66–76, 1982.

[7] François Clautiaux, Jacques Carlier, and Aziz Moukrim. Exact method for the two-dimensional orthogonal packing problem. *Elsevier Science*, April 2005.

[8] K.A. Dowsland, W.B. Dowsland, and J.A. Bennell. Jostling for position : local improvement for irregular cutting patterns. *J. Oper. Res. Soc.*, 49(6) :647–658, 1998. ISSN 0160-5682.

[9] Sándor Fekete and Jörg Schepers. An exact algorithm for higher-dimensional orthogonal packing. *Operations Research*, Volume(Number) :Pages, Year. Revised for publication.

[10] Sándor P Fekete and Jörg Schepers. A combinatorial characterization of higher-dimensional orthogonal packing. *Mathematics of Operations Research*, 29(2) :353–368, 2004.

[11] Florian Fontan. Github profile. `https://github.com/fontanf`, 2024.

[12] J.B. Frenk and G.G. Galambos. Hybrid next-fit algorithm for the two-dimensional rectangle bin-packing problem. *Computing*, 39 :201–217, 1987.

[13] M.R. Garey and D.S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W.H. Freeman & Co., New York, NY, USA, 1979.

[14] Bernard T Han, George Diehr, and Jack S Cook. Multiple-type, two-dimensional bin packing problems : Applications and algorithms. *Annals of Operations Research*, 50(1-4) :239–261, 1994.

[15] E. G. Coffman Jr., M. R. Garey, D. S. Johnson, and R. E. Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 9 :801–826, 1980.

[16] Ya Liu, Chengbin Chu, and Kanliang Wang. A new heuristic algorithm for a class of two-dimensional bin-packing problems. *International Journal of Advanced Manufacturing Technology*, 57(9-12) :1235–1244, 2011. doi : 10.1007/s00170-011-3351-1.

[17] A Lodi, S Martello, and D Vigo. Approximation algorithms for the oriented two-dimensional bin packing problem. *European Journal of Operational Research*, 112 :158–166, 1999.

[18] A Lodi, S Martello, and D Vigo. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. *INFORMS Journal on Computing*, 11 :345–357, 1999.

[19] A. Lodi, S. Martello, and D. Vigo. Recent advances on two-dimensional bin packing problems. *Discrete and Applied Mathematics*, 123 :379–396, 2002.

[20] A Lodi, S Martello, and D Vigo. Neighborhood search algorithm for the guillotine non-oriented two-dimensional bin packing problem. In Stefan Voss, Silvano Martello, Ibrahim H. Osman, and Gérard Roucairol, editors, *Metaheuristics : Progress in Complex Systems Optimization*, pages 139–150. Springer, 2004.

[21] P. T. Silvano Martello. *Knapsack Problems : Algorithms and Computer Implementations*. John Wiley and Sons, 1990.

[22] Silvano Martello and Daniele Vigo. Exact solution of the two-dimensional finite bin packing problem. *Management Science*, 44(3) :388–399, 1998.

[23] A. Martinez-Sykora, R. Alvarez-Valdes, J.A. Bennell, R. Ruiz, and J.M. Tamarit. Matheuristics for the irregular bin packing problem with free rotations. *European Journal of Operational Research*, 258(2) :440–455, 2017.

[24] Renault Group. Renault group, car manufacturer - official website. `https://www.renaultgroup.com/`.

[25] Société française de Recherche Opérationnelle et d'aide à la décision. Challenge roadef/euro 2022 : Trucks loading problem. `https://www.roadef.org/challenge/2022/en/`, 2022.