Université de Montréal

Problèmes de tournées de véhicules avec contraintes de chargement

par
Jean-François Côté

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée à la Faculté des études supérieures
en vue de l'obtention du grade de Philosophiæ Doctor (Ph.D.)
en informatique

Décembre, 2013

Université de Montréal
Faculté des études supérieures

Cette thèse intitulée:

Problèmes de tournées de véhicules avec contraintes de chargement

présentée par:

Jean-François Côté

a été évaluée par un jury composé des personnes suivantes:

Jacques Ferland,	président-rapporteur
Jean-Yves Potvin,	directeur de recherche
Michel Gendreau,	codirecteur
Fabian Bastin,	membre du jury
Claude Comtois,	représentant du doyen de la FES

Thèse acceptée le: 2014-02-21

SOMMAIRE

Cette thèse s'intéresse aux problèmes de tournées de véhicules où l'on retrouve des contraintes de chargement ayant un impact sur les séquences de livraisons permises. Plus particulièrement, les items placés dans l'espace de chargement d'un véhicule doivent être directement accessibles lors de leur livraison sans qu'il soit nécessaire de déplacer d'autres items. Ces problèmes sont rencontrés dans plusieurs entreprises de transport qui livrent de gros objets (meubles, électroménagers).

Le premier article de cette thèse porte sur une méthode exacte pour un problème de confection d'une seule tournée où un véhicule, dont l'aire de chargement est divisée en un certain nombre de piles, doit effectuer des cueillettes et des livraisons respectant une contrainte de type dernier entré, premier sorti. Lors d'une collecte, les items recueillis doivent nécessairement être déposés sur le dessus de l'une des piles. Par ailleurs, lors d'une livraison, les items doivent nécessairement se trouver sur le dessus de l'une des piles. Une méthode de séparation et évaluation avec plans sécants est proposée pour résoudre ce problème.

Le second article présente une méthode de résolution exacte, également de type séparation et évaluation avec plans sécants, pour un problème de tournées de véhicules avec chargement d'items rectangulaires en deux dimensions. L'aire de chargement des véhicules correspond aussi à un espace rectangulaire avec une orientation, puisque les items doivent être chargés et déchargés par l'un des côtés. Une contrainte impose que les items d'un client soient directement accessibles au moment de leur livraison.

Le dernier article aborde une problème de tournées de véhicules avec chargement d'items rectangulaires, mais où les dimensions de certains items ne sont pas connus avec certitude lors de la planification des tournées. Il est toutefois possible d'associer une distribution de probabilités discrète sur les dimensions possibles de ces items. Le problème est résolu de manière exacte avec la méthode L-Shape en nombres entiers.

Mots clés : Problème de tournées de véhicules, contraintes de chargement, objets en deux-dimensions stochastiques, méthode L-shaped.

SUMMARY

In this thesis, we study mixed vehicle routing and loading problems where a constraint is imposed on delivery sequences. More precisely, the items in the loading area of a vehicle must be directly accessible, without moving any other item, at delivery time. These problems are often found in the transportation of large objects (furniture, appliances).

The first paper proposes a branch-and-cut algorithm for a variant of the single vehicle pickup and delivery problem, where the loading area of the vehicle is divided into several stacks. When an item is picked up, it must be placed on the top of one of these stacks. Conversely, an item must be on the top of one of these stacks to be delivered. This requirement is called “Last In First Out” or LIFO constraint.

The second paper presents another branch-and-cut algorithm for a vehicle routing and loading problem with two-dimensional rectangular items. The loading area of the vehicles is also a rectangular area where the items are taken out from one side. A constraint states that the items of a given customer must be directly accessible at delivery time.

The last paper considers a stochastic vehicle routing and loading problem with two-dimensional rectangular items where the dimensions of some items are unknown when the routes are planned. However, it is possible to associate a discrete probability distribution on the dimensions of these items. The problem is solved with the Integer L-Shaped method.

Keywords : Vehicle Routing Problem, loading constraints, stochastic two-dimensional items, L-shaped method.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

À mes parents.

REMERCIEMENTS

Tout d'abord, je désire remercier mes parents Ginette et Christian. Ils ont toujours cru en moi et ils m'ont toujours encouragé à aller plus loin. Sans eux, sans leur générosité et leur amour je ne me serais probablement pas rendu où je suis. Je suis heureux de vous avoir comme parents et je vous aime.

Je remercie mes professeurs Jean-Yves Potvin et Michel Gendreau qui, tout au long de ces quelques années, m'ont épaulé, écouté et qui ont contributué grandement à ce travail. Leur patience, leur sagesse et leurs idées m'ont soutenu à accomplir une des plus grandes choses de ma vie.

J'aimerais remercier par la suite mon supérieur Christian Lafrance chez Clear Destination. Il m'a toujours poussé à surpasser mes limites, sans quoi je n'aurais jamais entrepris le doctorat.

Je désire remercier Maria Grazia Speranza et Claudia Archetti de Brescia en Italie pour m'avoir donner la chance d'effectuer mon premier article et de m'avoir permi d'y séjourner aussi souvent. Elles ont toujours été généreuses, gentilles et accueillantes.

D'autres remerciements s'addressent aussi à mon ami Manuel Iori de Reggio-Emilia avec qui j'ai pu faire un superbe article. Sa détermination, son profesionnalisme et disponibilité m'ont énormément inspiré.

Enfin, je remercie ma copine Ivana pour tout son amour, son attention, son support, sa grande compréhension et spécialement sa patience tout au long de ce travail.

CHAPITRE 1

INTRODUCTION

L'accroissement de la vitesse des ordinateurs a permis à une multitude de chercheurs de considérer des problèmes mathématiques de plus en plus complexes. On pense, par exemple, aux problèmes de tournées de véhicules : une version de base avec contrainte de capacité a ainsi été abordée pour la première fois par Dantzig et Ramser en 1959 [54]. Par la suite, des problèmes de tournées de plus en plus complexes ont été étudiés avec l'ajout de depôts multiples, de fenêtres de temps ainsi que d'autres types de contraintes permettant de mieux modéliser la réalité des compagnies de transport.

Tous les problèmes de tournées de véhicules (VRP) ont comme problème de base le problème du voyageur de commerce (TSP). Pour un ensemble de villes donnée et une distance entre chaque paire de villes, un voyageur doit visiter toutes les villes de manière à ce que la distance totale parcourue soit minimale. Ceci revient à dire qu'il faut trouver un cycle Hamiltonien, soit un cycle traversant chacune des villes exactement une fois, de longueur minimale. Ce problème est $\mathcal{NP}$-Difficile et constitue un défi pour des millieurs de chercheurs depuis des dizaines d'années encore aujourd'hui.

Le problème du voyageur de commerce symmétrique (STSP) se définit sur un graphe $G = (V,E)$ où $V = \{1,...,n\}$ est un ensemble de n sommets (ou noeuds) et $E = \{(i, j) : i, j \in V\}$ est un ensemble d'arêtes. Un coût c_{ij} est encouru si l'arête (i, j) est empruntée. Dans le cas asymétrique (TSP), le problème se définit sur un graphe orienté $G = (V,A)$ où $A = \{(i, j) : i, j \in V\}$ est un ensemble d'arcs et où un coût c_{ij} est encouru si l'arc $(i, j) \in A$ est emprunté. Dans plusieurs applications et variantes, il est souvent plus efficace de modéliser le problème comme un STSP car l'ensemble des arêtes est beaucoup plus petit que l'ensemble des arcs du TSP.

Les problèmes de transport de marchandises se modélisent souvent comme un problème de tournées de véhicules avec contrainte de capacité (CVRP). Celui-ci est défini sur le graphe $G = (V,E)$ du STSP. L'ensemble des sommets V est partitionné en deux, soit d'une part le dépôt $V_0 = \{0\}$ et d'autre part $V_c = \{1,...,n\}$ l'ensemble des n clients.

Chaque client i a une demande q_i qui doit être livrée à l'emplacement du client. Une flotte de véhicules est disponible pour effectuer les tournées de livraison, en partant du dépôt et en y revenant à la fin. L'objectif est de minimiser la somme des distances parcourues tout en s'assurant que tous les clients sont désservis et que la demande totale sur chacune des tournées n'excède pas la capacité des véhicules. Ce problème fut traité dans une multitude d'articles suite aux travaux de Dantzig et Ramser. Nous référons les lecteurs aux livres suivants [81, 103, 156].

Ces dernières années, une toute nouvelle classe de problèmes de tournées de véhicules a été abordée. Cette classe contraint le placement des items à l'intérieur des véhicules afin de faciliter la séquence des livraisons. L'objectif principal est d'éviter que le chauffeur ait à déplacer des items dans le camion afin de livrer l'item courant. En effet, de telles manipulations augmentent les temps de livraison et donc les coûts. Dans ces problèmes, les véhicules sont munis d'un espace de chargement dont l'organisation peut varier. Par exemple, cet espace peut correspondre à une ou plusieurs piles de type "dernier entré, premier sorti", ou à un espace rectangulaire en deux dimensions, etc.

Les recherches réalisées dans cette thèse ont ainsi pour but d'approfondir les connaissances portant sur les problèmes de tournées où le chargement des véhicules est pris en compte. Il reste en effet de nombreux aspects à explorer afin d'automatiser le plus possible l'optimisation des tournées dans ce domaine.

La première partie de la thèse porte sur une méthode de résolution exacte pour un problème de confection d'une tournée avec cueillettes et livraisons pour un seul véhicule. Dans ce problème, l'espace de chargement du véhicule est divisé en un certain nombre de piles, forçant les cueillettes et livraisons à respecter une contrainte de type dernier entré, premier sorti. Ainsi, les items à recueillir seront nécessairement placés sur le dessus de l'une des piles au moment de leur cueillette, tandis que les items à livrer devront nécessairement être sur le dessus de l'une des piles au moment de leur livraison. Ce problème fut d'abord introduit dans mon mémoire de maîtrise où une heuristique a été proposée afin de le résoudre. Dans ce travail, nous élaborons plutôt une méthode de séparation et évaluation avec plans sécants ("branch-and-cut"). Nous proposons plusieurs formulations et inégalités valides qui sont comparées sur un ensemble d'instances tests.

Dans le deuxième travail, nous présentons une méthode de résolution exacte pour un problème de chargement de boîtes rectangulaires (en deux dimensions) que l'on rencontre dans de nombreux problèmes de livraison. L'espace de chargement des véhicules correspond également à une forme rectangulaire en deux dimensions. Une contrainte de déchargement est imposée sur le placement des boîtes dans le véhicule. Plus précisésement, les boîtes d'un client donné doivent être directement accessibles au moment du déchargement, sans qu'il soit nécessaire de déplacer les boîtes des autres clients. Ce problème apparaît comme sous-problème lors de la confection de tournées de véhicules avec chargement de boîtes rectangulaires. En effet, étant donné une tournée de livraison, il faut alors vérifier si toutes les boîtes peuvent entrer dans le véhicule. Nous proposons une méthode de séparation et évaluation avec plans sécants ("branch-and-cut") pour résoudre le problème de chargement, ainsi que plusieurs méthodes de prétraitement et de vérification de la réalisabilité d'une solution permettant d'accélérer les calculs. L'article apporte plusieurs contributions : une foule de prétraitements pour réduire l'espace de recherche, borne inférieures et des inégalités valides. Ces méthodes permettent d'obtenir d'excellent résultats en comparaison à ce qui se fait dans la littérature en plus d'être facilement applicable aux problèmes de chargement en trois-dimensions.

Le dernier travail aborde un problème souvent rencontré en pratique lors de la livraison d'objets en deux dimensions. Plus précisément, il arrive que certains objets aient des dimensions qui sont encore inconnues au moment de la planification des tournées. Dans plusieurs cas, toutefois, il est possible d'avoir une description de ces objets, ce qui permet d'identifier leur type. À partir d'une base de données historique, on peut alors induire une distribution de probabilités discrète touchant aux dimensions de ces objets et à leur poids. Dans ce travail, nous faisons appel à la méthode de résolution développée précédemment pour le problème de chargement. Cette dernière est intégrée à une méthode *L-shaped* en nombre entiers qui permet d'aborder ce problème de nature probabiliste. Nous proposons un nouveau type de *lower bound functionals* pour améliorer l'approximation de la fonction de recours. Cette nouvelle classe de fonctions peut aussi être utilisée pour des problèmes stochastiques plus généraux. Des résultats obtenus sur des instances tests probabilistes ainsi que sur des instances déterministes démontrent l'efficacité de cette approche.

Cette thèse se compose de six chapitres. Le chapitre 2 est une revue de la littérature portant sur différents problèmes de tournées de véhicules avec contraintes de chargement. L'emphase est mise sur les problèmes susceptibles d'être rencontrés en pratique et sur les méthodes utilisées pour les résoudre. Le chapitre 3 présente le premier travail où nous abordons avec une méthode exacte un problème de confection d'une seule tournée avec cueillettes et livraisons et contrainte de chargement de type “dernier entré, premier sorti”. Le chapitre 4 décrit ensuite un algorithme exact pour le problème de chargement de boîtes en deux dimensions. Enfin, le chapitre 5 traite du problème de tournées probabiliste où les dimensions exactes de certains objets ne sont pas connues avec certitude. Une conclusion est présentée au chapitre 6 qui résume les principaux résultats de cette thèse ainsi que les avenues futures de recherche.

CHAPITRE 2

REVUE DE LITTÉRATURE

L'étude des problèmes de tournées avec contraintes de chargement a commencé au début des années 2000 pour des items en deux ou trois dimensions. Certains cas particuliers furent étudiés par la suite. Une première revue de la littérature a déjà été présentée dans [88]. Nous irons toutefois un peu plus loin en examinant des travaux non encore publiés, réalisés par exemple dans le cadre de thèses, et nous discuterons également davantage en profondeur les méthodes de résolution proposées pour ces problèmes. Nous ferons enfin une brève revue des problèmes de tournées stochastiques.

Dans ce chapitre, les sections 2.1 et 2.2 se penchent sur des problèmes où les items à transporter sont en deux et trois dimensions, respectivement. Un cas particulier sera ensuite abordé dans la section 2.3, soit un problème d'empaquetage de boîtes en trois dimensions disposées sur des pallettes. La section 2.4 se concentre quant à elle sur des problèmes où l'espace de chargement des véhicules est divisé en piles. Les sections 2.5 et 2.6 portent sur des problèmes avec contraintes de chargement de type "dernier entré, premier sorti" et "premier entré, premier sorti", respectivement. La section 2.7 aborde un problème de transport de voitures. Enfin, la dernière section présente une revue des problèmes de tournées stochastiques.

2.1 Livraison de boîtes rectangulaires en deux dimensions

Le problème de livraison d'items rectangulaires en deux dimensions ("Two-dimensional Vehicle Routing and Loading Problem" ou $2L-CVRP$) est l'un des premiers problèmes de tournées avec chargement à apparaître dans la litérature. Dans ce problème, chaque client demande qu'un ensemble d'items rectangulaires lui soit livré. Un item est caractérisé par une hauteur (ou longueur), une largeur et un poids. Ces items sont transportés par une flotte de véhicules homogène où chaque véhicule a une aire de chargement également rectangulaire. Une route réalisable correspond à une séquence de clients visités telle que la somme des poids des items transportés est inférieure à la capacité du véhicule et telle que le chargement associé est réalisable. Certains problèmes classiques de chargement sont les suivants :

- “Two-Dimensional Bin Packing Problem” ou $2BPP$: Supposons un ensemble d’items rectangulaires en deux dimensions et une quantité infinie de contenants rectangulaires identiques pouvant accueillir les items. L’objectif est de placer les items à l’intérieur des contenants de telle sorte que le nombre de contenants utilisé soit minimal et qu’il n’y ait aucun chevauchement des items.
- “Two-Dimensional Strip Packing Problem” ou $2SPP$: Supposons un ensemble d’items rectangulaires en deux dimensions et une bande de hauteur infinie et de largeur fixe. L’objectif est de placer tous les items de telle façon qu’il n’y ait aucun chevauchement et que la hauteur de la bande utilisée soit minimale.
- “Two-Orthogonal Packing Problem” ou $2OPP$: Supposons un ensemble d’items rectangulaires en deux dimensions et un contenant également rectangulaire et en deux dimensions. Le but est ici de savoir si les items peuvent tous être tous placés à l’intérieur du contenant sans qu’il n’y ait de chevauchement. Il s’agit donc d’un problème de réalisabilité (ou décidabilité).

Ces trois problèmes sont NP-difficiles. On peut noter que le $2L-CVRP$ sans contrainte de capacité est un problème $2OPP$. De même, le $2OPP$ est un sous-problème de $2BPP$ et $2SPP$. Il est donc possible d’utiliser les méthodes de résolution pour le $2BPP$ et le $2SPP$ afin de résoudre le $2OPP$. Une très vaste litérature existe pour ces problèmes classiques de chargement et nous n’en ferons ici qu’un bref survol.

Les problèmes en deux dimensions correspondent à une extension naturelle des problèmes en une seule dimension, comme le problème de sac-à-dos ou “Knapsack Problem” et le “Bin Packing Problem”. Une foule de variantes de problèmes en une dimension sont présentées dans [118] incluant différentes bornes inférieures et méthodes exactes pour les résoudre. Les revues dans [40, 41] contiennent également plusieurs résultats intéressants portant sur ces problèmes.

Les premières méthodes exactes pour le 2*BPP* et le 2*SPP* sont rapportées dans [119] et [116], respectivement. Dans chacun des cas, un sous-problème 2*OPP* est résolu à l'aide d'un arbre de branchement où, à chaque niveau de l'arbre, un item est choisi et placé à une des coordonnées réalisables de la solution partielle courante afin d'effectuer un branchement. L'arbre généré devient rapidement énorme et même des problèmes avec à peine une vingtaine d'items rectangulaires se sont révélés difficiles à résoudre. Les auteurs dans [4, 22, 93] rapportent des méthodes de résolution similaires mais procèdent de manière un peu différente lors du branchement. De plus, des techniques d'élagage sont développées afin de limiter l'explosion de l'arbre. Une autre technique d'énumération est proposée dans [67] pour des problèmes en plusieurs dimensions. Ici, le branchement repose sur le fait que deux items peuvent seulement être l'un à côté de l'autre ou l'un par dessus l'autre, ce qui garantit un nombre polynomial de branchements. Les auteurs présentent plusieurs fondements théoriques à leur approche mais obtiennent des résultats mitigés. Cet algorithme fut ensuite amélioré par [13, 122]. La thèse de Joncour [92] présente également une foule de relaxations. En ce qui connecerne les approches heuristiques, nous référons le lecteur aux revues de l'état de l'art dans [138, 161].

La plupart des algorithmes d'empaquetage ou "packing" utilisés pour trouver un chargement réalisable dans les problèmes de tournées font appel à des variantes de l'heuristique "Bottom-Left" [9] ou "Bottom-left fill" [32]. Ces algorithmes commencent par trier les items en ordre non croissant de largeur (d'autres critères peuvent aussi être utilisés). L'item en première position est d'abord placé dans le coin inférieur gauche du contenant et les suivants sont ensuite placés le plus en bas et à gauche possible. Les heuristiques "Improved Bottom-Left" [110] et "Touching Perimeter" [109] fonctionnent de manière similaire, mais avec des critères de chargement un peu plus sophistiqués.

Dans certains cas, le chargement doit respecter une contrainte de type "dernier entré, premier sorti", menant à un chargement dit "séquentiel", dans lequel chaque item est directement disponible lors de sa livraison, sans avoir à déplacer les items des autres clients. La figure 2.1 présente en (a) un chargement séquentiel et en (b) un chargement classique où les items sont retirés par le haut. Les numéros sur les items indiquent leur ordre de livraison. Ainsi, la solution en (b) ne correspond pas à un chargement séquentiel car l'item 1 doit être livré en premier et il est bloqué par les items 4 et 6.

Nous pouvons retrouver quatre types de problèmes de tournées avec chargement dans la littérature :

- $2|OS|L$: chargement orienté et séquentiel en deux dimensions ;
- $2|ON|L$: chargement orienté en deux dimensions ;
- $2|NS|L$: chargement non-orienté et séquentiel en deux dimensions ;
- $2|NN|L$: chargement non-orienté en deux dimensions.

Un chargement est dit orienté si les items ont une orientation spécifique qui ne peut être modifié. Il faut noter qu'il existe des relations entre les solutions réalisables pour ces différents types de problèmes. Par exemple, un chargement réalisable pour le $2|OS|L$ est nécessairement réalisable pour les trois autres types, alors qu'un chargement réalisable pour le $2|NN|L$ n'est pas nécessairement réalisable pour les trois autres types. En particulier, toute borne inférieure pour le $2|NN|L$ est valide pour les trois autres types et toute solution optimale d'un $2L-CVRP$ en version $2|NN|L$ aura une valeur au plus égale à celle des autres types.

Les algorithmes de résolution pour les problèmes de tournées avec chargement séquentiel trient typiquement les items en ordre non croissant selon leur numéro de séquence et, pour un même numéro de séquence, en ordre non croissant de largeur.

La première étude sur les problèmes de tournées avec chargement se retrouve dans [89]. Ici, les auteurs se concentrent sur les versions $2|OS|L$ et $2|ON|L$ du problème et font appel à une méthode de séparation et évaluation avec plans sécants ("branch-and-cut") en utilisant le solveur CPLEX pour résoudre le problème de tournées. Les contraintes de capacité (sur le poids) et de chargement étant relaxées, des inégalités valides sont générées à chaque noeud de l'arbre de branchement pour ces contraintes. Essentiellement, on tente d'identifier des sous-ensembles de clients tels que la somme des poids ou la somme des aires des items qui leur sont livrés excèdent la capacité ou le volume de chargement du véhicule. Si de tels sous-ensembles existent, des coupes sont générées

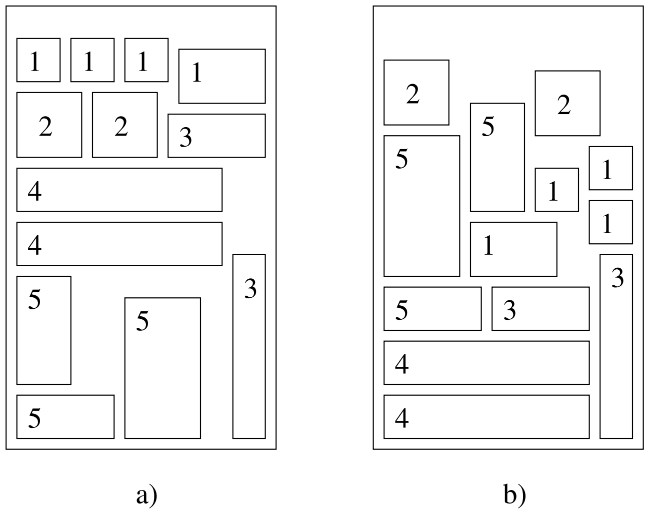

Figure 2.1 – Deux problèmes de chargement : a) séquentiel b) sans restriction

pour les éliminer. Autrement, la méthode de séparation est poursuivie jusqu'à l'obtention d'une solution entière.

Les routes sont alors extraites de la solution obtenue et les chargements sont validés à l'aide d'une méthode exacte de "branch-and-cut" qui s'inspire d'ailleurs fortement des travaux dans [117, 119]. La méthode de validation considère les items restants et les place, un à un, à toutes les positions possibles dans le chargement courant. Les bornes inférieures dans [119] sont d'abord appliquées à la racine de l'arbre. Si l'une de ces bornes indique qu'il faut plus d'un véhicule pour satisfaire les clients, la recherche se termine aussitôt. Sinon, l'heuristique dans [15] est utilisée pour essayer de trouver un chargement réalisable. Cette heuristique procède itérativement en plaçant les items toujours le plus en bas et à gauche tout en s'assurant qu'il n'y ait aucun chevauchement. Si aucun chargement réalisable n'est trouvé, alors un branchement est appliqué. De nouvelles bornes inférieures sont ensuite calculées à chacun des noeuds de l'arbre de branchement afin de favoriser l'élagage. Si la méthode ne peut identifier un chargement réalisable, alors une contrainte est ajoutée afin que cette route ne soit plus considérée par la méthode de "branch-and-cut". Les auteurs ont ainsi pu résoudre des problèmes ayant jusqu'à 35 clients et plus de 100 items en moins de 24 heures de calcul.

Les auteurs dans [76] ont proposé la première méta-heuristique pour résoudre les versions $2|OS|L$ et $2|ON|L$ du problème. Les auteurs y exploitent une recherche tabou qu'ils avaient précédemment conçue, appelée $TABUROUTE$ [74]. Ainsi, les déplacements de clients d'une route à une autre sont réalisés à l'aide d'insertions géneralisées de type GENI [73] qui correspondent à une insertion classique suivie d'une réoptimisation locale. Au cours de la recherche, les contraintes portant sur le poids maximal et le chargement sont relaxées afin de faciliter l'exploration de l'espace des solutions. Toute violation de ces contraintes induit une pénalité dans l'objectif qui est automatiquement ajustée au cours de l'algorithme. Ainsi, la pénalité associée à une contrainte augmente si elle est violée plus souvent, tandis que la pénalité diminue si elle est violée moins souvent. Il faut noter que la contrainte sur le poids maximal est facile à valider. Toutefois, la contrainte portant sur le chargement est beaucoup plus complexe. Pour ce faire, les auteurs utilisent l'heuristique "Touching Perimeter" de [109] qui a été appliquée au $2SPP$ dans [87]. Deux procédures d'intensification provenant de [89] sont utilisées pour forcer la réalisabilité de la solution courante, si elle ne l'est pas. La première est une version modifiée de leur heuristique de chargement et l'autre est une version tronquée de la méthode de séparation. Les instances tests générées contiennent entre 15 et 255 clients et entre 45 et 786 items. L'algorithme est comparé à la méthode exacte rapportée dans [89].

Une recherche tabou guidée [159] est introduite dans [165]. Trois voisinages classiques sont utilisés : repositionnement de clients, échange de paires de clients et mouvements 2-opt [53]. Le mécanisme de guidage se charge d'identifier et de retirer les séquences de mauvaise qualité en introduisant des pénalités dans l'objectif. Les variantes $2|OS|L$ et $2|ON|L$ sont résolues à l'aide de cinq heuristiques ayant toutes un comportement similaire. Elles démarrent avec un certain ordonnancement des items (par exemple, en ordre non croissant de largeur) et utilisent une liste de positions possibles ne contenant au départ que le coin inférieur gauche (0,0). À chaque itération, une position est choisie pour l'item courant en maximisant un critère donné : les deux premiers proviennent de l'heuristique "Bottom-Left Fill" [32], les deux autres reposent sur l'heuristique "Touching Perimeter" [109] tandis que la dernière est une nouvelle approche appelée “Min Area heuristic”. Dans ce dernier cas, les auteurs agrègent les positions possibles en surfaces rectangulaires non occupées. L’heuristique sélectionne alors la position permettant de minimiser la surface non utilisée du rectangle correspondant. Les différentes heuristiques sont appelées l’une après l’autre selon leur ordre de complexité, jusqu’à l’obtention d’une solution réalisable. La recherche locale pour optimiser les tournées est accélérée en éliminant les arcs les plus coûteux qui ont peu de chance de faire partie d’une solution optimale. Les tests furent réalisés sur les instances dans [76] et ont permis d’améliorer les résultats.

Une recherche tabou qui étend celle utilisée dans [165] est présentée dans [107]. Les auteurs modifient certains mécanismes de guidage tout en conservant les heuristiques de chargement rapportées dans [106]. Le mécanisme de guidage est modifié à la manière de la recherche locale guidée et étendue décrite dans [124] où un critère d’aspiration est spécialement défini pour les méthodes avec pénalités. Les auteurs prétendent que cette extension donne plus de robustesse à l’algorithme. Les résultats sont comparés seulement avec ceux dans [165] et apportent une certaine amélioration.

Dans [71], les auteurs proposent une métaheuristique basée sur les colonies de fourmis [137]. Les bornes inférieures dans [57, 119] sont utilisées pour les versions $2|ON|L$ et $2|NN|L$, respectivement. Les auteurs exploitent ensuite l’heuristique “Bottom-left fill” pour le chargement [23] afin de concevoir quatre heuristiques adaptées aux quatre types de problèmes de tournées avec chargement. L’heuristique “Touching Perimeter” [109] est également adaptée à cette fin. Si aucune des deux méthodes ne peut identifier de solution réalisable, alors la méthode de séparation tronquée dans [89] est utilisée. L’algorithme améliore les résultats rapportés dans [76] par un peu plus de 3%.

Un recuit simulé [95] est proposé dans [106] pour les variantes $2|ON|L$ et $2|NN|L$ du problème. Ce travail, très similaire à celui dans [165], utilise les mêmes voisinages au cours de la recherche locale mais ajoute une nouvelle heuristique de chargement aux cinq heuristiques originales. Dans ce dernier cas, l’emplacement le plus en bas et à gauche est d’abord choisi et, parmi tous les items restants, on choisit celui qui s’y insère le mieux. Les auteurs appliquent une à une les heuristiques de chargement, comme dans [165], afin de trouver un chargement réalisable. Les résultats obtenus sont supérieurs à ceux rapportés dans [71, 165].

Un travail intéressant est proposé dans la thèse de Bontoux [18]. Celui-ci présente deux heuristiques faisant appel à une méthode de génération de colonnes [58] pour résoudre la variante $2|OS|L$. Des routes initiales sont tout d'abord fournies au problème maître à des fins d'initialisation. À chaque itération, on tente ensuite d'identifier de nouvelles routes permettant d'améliorer l'objectif en résolvant des sous-problèmes de plus courts chemins. Dans le $2|OS|L$, il faut évidemment valider les contraintes de capacité et de chargement séquentiel. La première étant facile à valider, les auteurs proposent deux façons de générer des routes répondant à la contrainte de chargement. La première méthode, après avoir identifié de nouvelles routes suite à la résolution des sous-problèmes de plus courts chemins, valide la contrainte de chargement en appliquant différentes heuristiques. La seconde méthode génère des routes réalisables lors de la résolution même des sous-problèmes de plus courts chemins. Les heuristiques de chargement utilisées reposent sur les approches “Bottom-Left fill”[32], “Improved Bottom-Left” [110] et “Touching Perimeter” [109]. La solution optimale du problème maître est obtenue lorsqu'il n'est plus possible d'identifier de nouvelles routes réalisables permettant d'améliorer l'objectif. Les deux méthodes sont de nature heuristique, car il est possible que des routes véritablement réalisables ne puissent être validées comme telles par les heuristiques utilisées. Par ailleurs, plusieurs astuces sont mises en oeuvre afin d'améliorer la vitesse de calcul. Les résultats obtenus sont comparés à ceux dans [71, 76, 89] mais sont malheureusement de qualité inférieure.

Dans [150], les auteurs présentent une manière originale de savoir assez rapidement si une route est réalisable ou non. En effet, si une route r est réalisable, alors toutes les sous-routes r' de r sont réalisables. Par ailleurs, si r n'est pas réalisable, alors toute route r' ayant r comme sous-route ne peut être réalisable. Les auteurs proposent donc des structures de données permettant d'emmagasiner les routes afin de rechercher rapidement des sous-routes et des super-routes particulières. Une recherche à voisinage variable [125] développée dans [157] est utilisée pour la confection des tournées, tandis que des heuristiques et une méthode exacte sont utilisées pour résoudre les problèmes de chargement [71]. Les auteurs observent malheureusement que le temps requis pour identifier des sous-routes ou des super-routes demeure trop long même avec les structures de données proposées.

Une métaheuristique GRASP x ELS [135] est suggérée dans [155] pour les variantes $2|ON|L$ et $2|NN|L$. Les auteurs exploitent une relaxation du problème de chargement qu'ils appellent le “Resource Constrained Project Scheduling Problem” avec une seule ressource et sans contrainte de précédence. Il s'agit en fait d'une relaxation du $2SPP$ appelée “One-dimensional Contiguous Bin Packing Problem ($1CBP$)” [116]. Les auteurs résolvent le problème de tournées avec leur métaheuristique en s'assurant que les routes sastisfont la contrainte de chargement relaxée. Un tour géant est d'abord produit et une procédure de partitionnement de type “Split” [136] légèrement modifiée est ensuite appliquée. Ils tentent également de construire une solution pour la variante $2|ON|L$ (ou $2|NN|L$) à partir de la solution relaxée en faisant appel à une autre heuristique. L'algorithme final est comparé avec ceux dans [71, 76, 165] et produit les meilleurs résultats connus à ce jour sur les instances tests dans [76]. Les résultats pour la version $2|ON|L$ sont rapportés dans [62].

Une extension avec fenêtres de temps du problème $2L-CVRP$ est introduite dans [94] et est abordée avec une approche mémétique [126]. Il s'agit essentiellement d'un algorithme génétique couplé avec une méthode de recherche locale. L'algorithme fait appel à une version modifiée de la procédure “Split” [136] pour tenir compte des fenêtres de temps, telle que rapportée dans [96]. Les auteurs utilisent les voisinages Or-Opt, 2-Opt et 2-Opt* au sein de la recherche locale qui est appliquée aux routes obtenues à la suite du “Split”. Trois heuristiques valident le chargement pour une route donnée, soit l'heuristique de construction présentée dans [3], l'heuristique “Touching Perimeter” [109] et le “Shelf Heuristic Filling” [14]. Étant donné qu'aucune autre méthode de résolution n'est connue pour ce problème, les auteurs comparent différentes versions de leur algorithme. Ils remarquent en particulier que l'heuristique “Touching Perimeter” permet d'obtenir les meilleures solutions parmi les trois méthodes retenues pour résoudre le problème de chargement.

Les auteurs dans [105] considèrent une variante du $2L-CVRP$ pour une flotte de véhicules hétérogène. Ici, les véhicules ont des coûts d'utilisation fixes et variables ainsi qu'un espace de chargement qui diffère d'un véhicule à l'autre. La méthode de résolution proposée est fortement similaire à celle dans [106]. Les problèmes de chargement sont résolus à l'aide des heuristiques développées dans [165] et la confection des tournées est optimisée par recuit simulé. Les auteurs génèrent aussi de nouvelles instances tests à partir des instances originalement proposées dans [76].

Enfin, les auteurs dans [115] présentent une extension des problèmes en deux dimensions avec chargement séquentiel où l'on retrouve des cueillettes et des livraisons au sein d'une même route. Toutefois, seul un modèle de programmation par contraintes est présenté.

De façon générale, il est très difficile de comparer tous ces algorithmes entre eux. Le language utilisé pour l'implantation, la qualité de cette implantation ainsi que la machine utilisée ont tous un impact important sur les résultats rapportés. De plus, comme ces algorithmes abordent à la fois la confection des tournées et le chargement, il est difficile d'évaluer la contribution individuelle des méthodes conçues pour chacun de ces deux aspects du problème. Il se pourrait ainsi qu'une excellente heuristique de chargement soit intégrée à un algorithme qui paraît peu performant globalement à cause de la faiblesse de la composante qui prend en charge la confection des tournées. La revue de la littérature nous a également permis de constater qu'il existe encore de nombreuses opportunités de recherche pour les problèmes avec chargement séquentiel, qui se rapprochent souvent davantage des applications rencontrées en pratique.

2.2 Livraison de boîtes rectangulaires en trois dimensions

La livraison de boîtes en trois dimensions de forme rectangulaire ("Three-dimensional Vehicle Routing and Loading Problem") ou $3L-CVRP$, telle que présentée dans [75], est la variante qui se rapproche le plus de la réalité. Ici, chaque client désire qu'un ensemble d'items, ayant chacun la forme d'un parallépipède rectangle et que nous appellerons simplement boîtes dans la suite, lui soit livré par une flotte de véhicules homogène. Chaque véhicule possède un conteneur, également de forme rectangulaire. Les boîtes possèdent une largeur, une hauteur, une profondeur ainsi qu'un poids. L'objectif est de trouver un ensemble de routes de livraison qui minimise la somme des distances parcourues par les véhicules tout en s'assurant que chaque client est visité exactement une fois, que la somme des poids des boîtes dans chaque route ne dépasse pas la capacité du véhicule et que les contraintes de chargement sont satisfaites. Plusieurs types de contraintes de chargement sont rapportés dans la litérature, tels que :

- orientation verticale,
- contrainte liée à la fragilité d'un item,
- contrainte touchant à l'aire de support,
- contrainte de type “dernier entré, premier sorti”.

Ainsi, l'orientation verticale est requise par exemple dans le cas des réfrigérateurs afin que les gaz réfrigérants ne s'échappent pas. La deuxième contrainte empêche de déposer un item trop lourd sur un item plus fragile. La troisième contraite exige que la surface qui supporte la base d'une boîte ait une aire plus grande ou égale à un certain seuil (généralement exprimé comme un pourcentage de l'aire de la base de cette boîte). Enfin, La contrainte de type “dernier entré, premier sorti” demande qu'une boîte soit directement accessible au moment de sa livraison (on peut donc également parler de contrainte de déchargement).

Les problèmes touchant à la satisfaction de telles contraintes de chargement sont très similaires à ceux que l'on retrouve dans les problèmes classiques de “packing” en deux dimensions. Plus précisément :

- “Container Loading Problem” ou *CLP* : nous disposons d'un conteneur de largeur et de longueur finies mais de hauteur infinie. L'objectif est de placer les boîtes à l'intérieur du conteneur de telle sorte que la hauteur de l'espace occupé soit minimale.
- “Three-Dimensional Bin Packing Problem” ou 3*BPP* : nous disposons d'un ensemble de conteneurs possédant tous les mêmes dimensions et d'un ensemble de

boîtes à placer à l'intérieur de ces conteneurs. L'objectif est de placer toutes les boîtes de telle sorte que le nombre de conteneurs utilisés soit minimal.

- "Knapsack Loading Problem" ou *KLP* : nous disposons d'un ensemble de boîtes ayant chacune une valeur différente. L'objectif est de choisir et placer un sous-ensemble de ces boîtes dans un conteneur unique de dimension finie afin de maximiser la valeur totale des boîtes choisies. Si le valeur d'une boîte correspond à son volume, on minimise alors l'espace résiduel.

Comme les problèmes de chargement en trois dimensions constituent une extension naturelle des problèmes en deux dimensions, plusieurs techniques de résolution font appel à des adaptations d'algorithmes connus en deux dimensions ou réduisent le problème en sous-problèmes en deux dimensions. La littérature étant très abondante sur ce sujet, nous ne rapportons ici que les travaux les plus importants.

À notre connaissance, la première méthode exacte pour le $3BPP$ se trouve dans [117] où une approche en deux phases est présentée. Lors de la première phase, un arbre d'énumération est généré afin d'affecter les boîtes aux conteneurs. Dans la seconde phase, une méthode exacte est appelée pour résoudre les problèmes à un seul conteneur obtenus durant la première phase. Plusieurs bornes inférieures dérivées du problème en deux dimensions sont également proposées.

La première heuristique pour le problème en trois dimensions se trouve dans [79]. Cette heuristique insère les items dans un ordre prédéterminé en partant du fond du conteneur (par exemple, de l'item le plus volumineux au moins volumineux). Plusieurs des heuristiques rapportées dans la littérature par la suite font appel à cette approche, tels l'arbre d'énumération partielle dans [65], l'heuristique avec points extrêmes dans [52] ou le GRASP dans [130].

La méthode de résolution dans [75] est du même type que celle rapportée dans [76] pour le cas à deux dimensions. En effet, les auteurs utilisent ici aussi une variante de $TABUROUTE$ [74] pour confectionner des tournées. Les problèmes de chargement sont également résolus à l'aide d'une recherche tabou. Celle-ci exploite les heuristiques "Bottom-Left" [9] et "Touching Parameter" [109] qui sont adaptées au problème en trois dimensions. Essentiellement, la recherche tabou modifie l'ordre de présentation des items à ces deux heuristiques d'insertion afin d'obtenir différentes solutions. De nouvelles instances tests en trois dimensions sont générées à partir d'instances classiques en deux dimensions, ayant de 15 à 100 clients et de 32 à 198 boîtes.

Les auteurs dans [72] généralisent aussi certaines approches de résolution proposées pour le $2L-CVRP$. Ils utilisent entre autres les bornes inférieures dans [117] pour démontrer la non réalisabilité d'un chargement. De plus, ils exploitent les algorithmes gloutons proposés dans [75], sans toutefois les intégrer à une recherche tabou. Ils adaptent aussi l'algorithme de colonies de fourmis dans [137] pour le cas en trois dimensions. L'algorithme obtenu de cette façon se montre d'ailleurs supérieur à la recherche tabou rapportée dans [75].

Dans [160], les auteurs présentent une recherche tabou et deux nouvelles heuristiques pour valider le chargement, appelées "Deepest-Bottom-Left-Fill" (DBLF) et "Maximum Touching Area" (MTA). Les boîtes sont préalablement triées et insérées une à la fois. L'heuristique DBLF choisit la position réalisable "la plus en bas, la plus profonde et la plus à gauche". Deux versions de cette heuristique sont proposées pour tenir compte de différents contraintes. Dans le cas de MTA, la position choisie pour la boîte courante est celle qui maximise la surface qu'elle touche. L'algorithme de confection de tournées débute par la construction d'une solution initiale et se poursuit avec une recherche tabou. Celle-ci fait appel à cinq voisinages différents, ayant chacun une certaine probabilité d'être choisi à chaque itération.

Une implantation particulièrement efficace de la recherche tabou est présentée dans [21]. L'auteur utilise quatre types de mouvements : échange de deux clients dans deux routes différentes, déplacement d'un client vers une autre route, échange de deux clients dans la même route, déplacement d'un client à l'intérieur de la même route. Dans les travaux précédents des mêmes auteurs, ceux-ci évaluaient tous les mouvements possibles dans le voisinage tout en s'assurant également de valider le chargement. Le meilleur mouvement réalisable était alors appliqué à la solution courante. Dans [21], l'auteur trie d'abord tous les mouvements possibles, du meilleur au pire, pour le problème de tournées. Ensuite, il parcoure la liste de haut en bas afin de valider le chargement. Dès qu'un chargement réalisable est identifié, la solution correspondante est acceptée. Cette approche réduit grandement les temps de calcul. Un algorithme sophistiqué qui génère de manière récursive un arbre de recherche tronqué est utilisé afin de résoudre le problème de chargement [65]. Les résultats obtenus sont clairement supérieurs à ceux rapportés dans [72, 75, 152].

Dans [97], les auteurs proposent une approche issue de leur algorithme GRASP x ELS pour le problème en deux dimensions [62, 155], où seule la résolution du problème de chargement est adaptée au $3D$. De fait, les auteurs relaxent la contrainte sur la hauteur afin de résoudre un problème de chargement en deux dimensions. Un chevauchement des boîtes est toutefois permis dans l'espace en deux dimensions, en autant que la somme des hauteurs des boîtes qui se chevauchent soit plus petite ou égale à la hauteur du conteneur. La première phase tente donc de minimiser les chevauchements dans l'espace à deux dimensions. Une méthode de construction est appelée par la suite pour obtenir une solution en trois dimensions. Les résultats obtenus sont supérieurs par un peu plus de 1% à ceux rapportés dans [72].

Une idée originale est proposée dans [120] dans le cadre d'une méthode de génération de colonnes. Ici, les auteurs font appel à une heuristique de colonies de fourmis pour identifier des routes de coût réduit négatif dans le sous-problème ("pricing"). Le chargement est alors validé avec la méthode rapportée dans [21]. Les routes réalisables sont ensuite ajoutées au problème maître qui correspond à un problème de couverture d'ensemble ("set covering"). Avec cet algorithme, ils obtiennent de meilleurs résultats que ceux rapportés dans [72, 75, 152], mais avec des temps de calcul plus élevés.

L'étude dans [128] décrit une problématique de type $3L-CVRP$ avec fenêtres de temps. Les auteurs présentent une formulation mathématique en nombre entiers ainsi que deux heuristiques de construction qui sont suivies d'une méthode de recherche locale. Dans la première heuristique, certaines contraintes du problème de chargement sont relaxées (comme la contrainte "dernier arrivé, premier sorti") afin de simplifier la résolution. La seconde heuristique, par contre, considère toutes les contraintes. Le chargement est ensuite validé avec l'heuristique GRASP développée par les mêmes auteurs dans [127]. Celle-ci est spécialement conçue pour tenir compte d'une contrainte de stabilité des boîtes. Les algorithmes sont testés sur des instances qui sont dérivées d'instances classiques de tournées avec fenêtres de temps.

L'approche proposée dans [152] pour une variante du $3L-CVRP$, appelée $M3L-CVRP$, permet de mieux modéliser certaines situations rencontrées en pratique. Si par exemple une partie de la base d'une boîte B se trouve au-dessus d'une boîte A, mais qu'il y a un espace vide entre les deux boîtes qui permet de déplacer A pour la sortir du camion sans affecter B ni aucune autre boîte, alors le chargement est autorisé, même dans le cas où A doit être livrée avant B dans la tournée. Dans le $3L-CVRP$ classique, un tel chargement serait interdit parce que A se trouve (totalement ou partiellement) sous B. Les auteurs utilisent la recherche tabou avec guidage [165] pour résoudre ce problème. Six algorithmes sont définis faisant appel à diverses heuristiques qui sont des variantes de "Bottom-Left" [9] et de "Touching Parameter" [109] adaptées pour le $3L-CVRP$. Les expériences numériques améliorent de 3% environ les résultats rapportés dans [75].

Enfin, une véritable application industrielle est présentée dans [30]. On y retrouve des contraintes très particulières comme une contrainte de poids associée à certaines zones du conteneur. L'objectif est de minimiser une fonction de coût assez complexe qui correspond à la réalité de la compagnie. Ces coûts incluent le nombre de boîtes non livrées, le nombre de conteneurs utilisés et l'espace résiduel. Les auteurs font appel à différents opérateurs qui sont intégrés à un recuit simulé et une recherche tabou. Les expérimentations furent réalisées sur des instances tests provenant de la compagnie partenaire ainsi que sur des instances classiques.

2.3 Pallet Packing Vehicle Routing Problem

Un problème proposé récemment dans dans la litérature s'intéresse à la confection de tournées pour une flotte de véhicules livrant des items en trois dimensions de petite taille [166]. Ces derniers ne sont pas chargés directement dans les véhicules, mais plutôt empilés sur des palettes qui sont par la suite placés dans les véhicules à l'aide de chariots élévateurs. Cette manière de procéder est observée dans de nombreuses applications réelles. Ainsi, les magasins d'électronique emmagasinent typiquement les ordinateurs et autres équipements sur des étagères dans un entrepôt. Les travailleurs prennent alors ces items et les déposent sur des palettes qui sont ensuite placées dans les véhicules. Tous les items pour un même client doivent être déposés sur la même palette. Les auteurs n'imposent pas de contraintes de type “dernier entré, premier sorti” car ils présument que les palettes sont toujours directement accessibles. Le problème de chargement de palettes en trois dimensions se rapproche alors fortement du problème $3L-CVRP$.

La méthodologie proposée est une recherche locale où on fait appel à trois opérateurs comme dans [152], soit le repositionnement de clients, l'échange de paires de clients et les mouvements 2-opt. Les chargements sont validés à l'aide d'heuristiques semblables à celles proposées dans [152] pour les problèmes en deux dimensions (voir la section 2.2), en l'occurrence les heuristiques “Bottom-Left Fill” [32], “Touching Perimeter” [109] et “Min Area heuristic” [165] qui sont adaptées aux problèmes en trois dimensions. Les auteurs trient les boîtes de six façons différentes et emploient 24 critères différents pour identifier la position de l'item courant au sein du conteneur, ce qui donne 144 combinaisons possibles. De nouvelles instances sont générées à partir de celles dans [34] où l'on retrouve entre 50 et 199 clients et un nombre d'items entre 444 et 1874. Ils générent aussi des instances avec fenêtres de temps qui s'inspirent de celles dans [149]. Les auteurs comparent leur méthodologie sur des $3L-CVRPs$ et montrent qu'il peuvent améliorer les résultats dans [152].

2.4 Problèmes multi-piles

Une problématique rencontrée dans la livraison de planches de bois, appelée “multi-pile vehicle routing problem” ou $MP-VRP$, est rapportée dans [59]. L'aire de chargement en deux dimensions de chaque véhicule est divisée en un certain nombre de piles de largeur unitaire (qui peuvent donc être considérées comme unidimensionnelles). Les auteurs considèrent un cas particulier, appelé $1VLP$, où une palette contient des planches qui occupent une largeur couvrant soit une, deux ou trois piles. Les palettes doivent aussi être disposées à l'intérieur du véhicule de façon à respecter l'ordre des livraisons. Il est donc impossible de déplacer des palettes pour atteindre celles qui doivent être livrées.

La figure 2.2 présente un exemple de chargement. Il faut remarquer que les planches et l'aire de chargement du $MP-VRP$ correspondent à un cas particulier du $2L-CVRP$ avec chargement séquentiel.

Les auteurs présentent deux algorithmes spécialisés pour leur problème multi-piles. Le premier est un algorithme approximé pour évaluer la longueur totale requise pour livrer des palettes dans une tournée. Le second est un algorithme de programmation dynamique qui résout le problème de manière exacte. Ces deux algorithmes sont intégrés à deux métaheuristiques, soit une recherche tabou et une méthode d'optimisation par colonies de fourmis. Les méthodes proposées sont très similaires à celles dans [71] qui a été décrite précédemment. Les auteurs précisent que l'utilisation de la méthode exacte pour résoudre le problème de chargement n'apporte que très peu d'améliorations au niveau de la qualité des solutions et entraîne une forte augmentation des temps de calcul.

Un second travail sur ce problème est présenté dans [157] où les auteurs proposent plusieurs approches permettant de résoudre le $1VLP$ de façon plus efficace que dans [59]. Ils notent qu'en retirant certaines contraintes du $1VLP$, des bornes inférieures peuvent être dérivées à partir de celles proposées dans [56, 86] pour le "Identical Parallel Machine Scheduling Problem". Une heuristique est décrite pour le $1VLP$ où un ensemble de planches est choisi à chaque itération de façon à minimiser la perte d'espace. Ils améliorent aussi l'algorithme de programmation dynamique dans [59] en ajoutant des critères de dominance, ce qui permet de rendre l'algorithme plus efficace. Pour résoudre le problème de tournées, ils emploient une recherche à voisinage variable [125] avec mouvements 2-opt [53]. Une méthode exacte basée sur celle dans [89] est également pré-

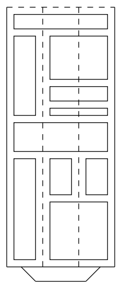

Figure 2.2 – Tournées de véhicules avec plusieurs piles

sentée. Au niveau du problème de chargement, les auteurs ont simplement remplacé les méthodes de résolution pour le chargement dans le $2L-CVRP$ par des méthodes adaptées au $MP-VRP$. Leurs algorithmes sont comparés avec ceux dans [59] et montrent des améliorations.

2.5 Cueillettes et livraisons avec contrainte “dernier entré, premier sorti”

Nous avons récemment étudié un problème de cueillettes et livraisons avec contraintes de chargement [48, 49]. Nous avons considéré un seul véhicule dont le conteneur est divisé en un certain nombre de piles de longueur finie, où des palettes de différentes longueurs peuvent y être déposées. Celles-ci entrent par l’arrière du véhicule et la manière dont celles-ci sont chargées et déchargées doit respecter la règle “dernier entré, premier sorti”. Encore une fois, si la palette A est recueillie avant B et que les deux palettes sont déposées dans la même pile, alors A doit nécessairement être livrée après B. La figure 2.3 présente une route réalisable pour une seule pile, où $i+$ et $i-$ correspondent aux points de cueillette et de livraison du client i. La figure 2.4 présente ensuite un exemple pour deux piles.

Ce problème, que nous appelons le “Single Vehicle Pickup and Delivery Problem with Multiple Loading Stacks” ou $1PDMS$, est résolu à l’aide d’une recherche à grand voisinage [142, 145, 148]. Cette approche heuristique applique un opérateur de destruction suivi d’un opérateur de reconstruction à la solution courante afin de produire une nouvelle solution. L’opérateur de destruction retire d’abord un certain nombre de clients de la solution courante qui sont ensuite réintroduits à l’aide de l’opérateur de re-

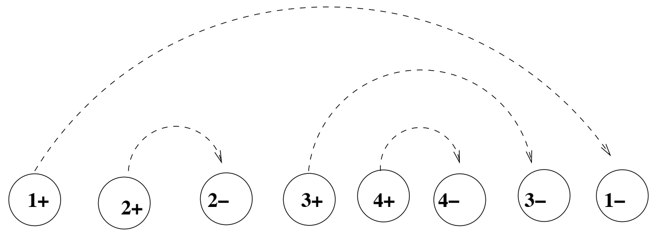

Figure 2.3 – Contrainte dernier entré premier sorti avec une seule pile

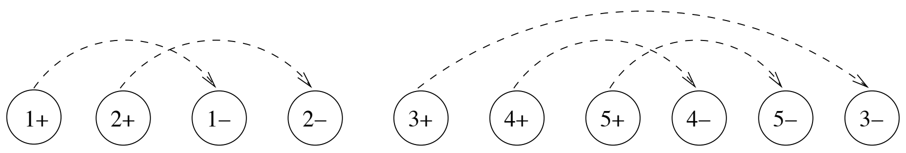

Figure 2.4 – Contrainte dernier entré premier sorti avec deux piles

construction. Comme un ensemble d'opérateurs de destruction et de reconstruction sont disponibles, on choisit de façon aléatoire les deux opérateurs qui sont appliqués à chaque itération. Les auteurs proposent aussi un algorithme de programmation dynamique qui permet d'obtenir la séquence optimale de cueillettes et livraisons pour un chargement donné. Des instances tests sont produites à partir de celles dans [29] où le nombre de sommets varie entre 25 et 750. Les auteurs rapportent des résultats compétitifs avec ceux rapportés dans [29, 68, 131] pour des cas particuliers du $1PDMS$, appelés $DTPSMS$ et $TSPPDL$ (voir plus bas).

Le $1PDMS$ généralise certains problèmes déjà rencontrés dans la littérature. Par exemple, un cas particulier étudié dans [131] s'intéresse au chargement d'un conteneur avec des palettes de longueur unitaire, où le conteneur est divisé en plusieurs piles de longueur finie. Les auteurs appellent ce problème le "Double Traveling Salesman Problem with Multiple Stacks" ou $DTPSMS$. Il faut ici construire deux tournées : une pour les cueillettes et l'autre pour les livraisons. Ainsi, le véhicule part à vide et recueille toutes les palettes avant de revenir à son point de départ. Les livraisons doivent ensuite être effectuées dans l'ordre "dernier entré, premier sorti", ce qui signifie que la palette d'un client doit être sur le dessus d'une pile pour être livrée. Dans [132], on note que le problème de chargement associé au $DTSPMS$ correspond à un problème de coloriage NP-difficile. Une analyse de la complexité de ce problème de chargement est aussi offerte dans [17, 31, 154].

Une approche originale pour résoudre le $DTSPMS$ est proposée dans [111]. Ici, les k meilleures routes du problème du voyageur de commerce pour les cueillettes seulement et les k meilleures routes pour les livraisons seulement sont générées. Ensuite, pour chaque paire de routes cueillette/livraison, le problème de chargement associé est résolu à l'aide d'un solveur en nombre entiers. Une solution optimale est alors obtenue dès qu'un chargement réalisable est identifié.

Les auteurs dans [28] emploient une méthodologie fort similaire à celle proposée dans [27] alors qu'ils font appel à différents critères permettant d'identifier des arcs qui ne peuvent mener à une solution au problème. L'application de tels critères mène ainsi au retrait d'un très grand nombre d'arcs pour les problèmes à deux piles. Par contre, la performance diminue fortement lorsqu'il y a trois piles et plus. C'est pourquoi les résultats sont meilleurs que ceux dans [111, 131] pour les instances à deux piles seulement.

Le travail dans [132] propose une méthode exacte par séparation et évaluation avec plans sécants pour le $DTSPMS$. Trois formulations sont proposées pour la contrainte de chargement. À chaque noeud de l'arbre de branchement, la contrainte qui permet d'éliminer les sous-tours est d'abord vérifiée. Dès qu'une solution entière et sans sous-tours est trouvée, le problème de chargement est résolu à l'aide d'un solveur en nombre entiers. Si la solution n'est pas réalisable, on tente d'identifier la plus courte sous-séquence non réalisable. Une contrainte est alors ajoutée au modèle afin d'interdire cette sous-séquence. Les auteurs résolvent ainsi des instances qui contiennent jusqu'à 42 sommets et 4 piles.

Une autre méthode de séparation et évaluation avec plans sécants a été récemment proposée dans [2] pour le même problème. À chaque noeud de l'arbre de branchement, l'algorithme recherche à l'aide d'heuristiques des chemins ou des sous-chemins ne respectant pas les contraintes de capacité ou les contraintes de type “dernier entré, premier sorti”. Lorsqu'un tel chemin est identifié, celui-ci est alors interdit. Les résultats rapportés sont supérieurs à ceux dans [28, 111, 132], alors que les auteurs peuvent résoudre des instances qui contiennent jusqu'à 56 sommets et 4 piles.

Dans [131], les auteurs présentent quatre métaheuristiques pour le $DTSPMS$ ainsi qu'une formulation mathématique. Plusieurs opérateurs classiques du TSP sont adaptés et intégrés aux quatre métaheuristiques, soit une recherche locale itérée, une recherche tabou[80], un recuit simulé [95] et une recherche à voisinage large [145]. Les auteurs résolvent des instances contenant jusqu'à 12 sommets à l'aide de leur formulation mathématique. Par ailleurs, des instances contenant jusqu'à 66 sommets sont résolues avec les différentes métaheuristiques et c'est la recherche à voisinage large qui obtient les meilleurs résultats. Deux métaheuristiques de recherche avec voisinages variables sont aussi présentées dans [68] exploitant six voisinages différents qui tiennent compte de la structure particulière du problème.

Dans [69], où les auteurs améliorent leur précédent travail [68], une recherche dans l'espace des solutions non réalisables est introduite qui permet de dépasser légèrement la contrainte de longueur maximale des piles. Lorsqu'une solution non réalisable est acceptée, les auteurs utilisent alors un opérateur particulier qui tente de reconstruire le chargement de façon à ce que la solution devienne réalisable.

Un autre cas particulier du $1PDMS$ est le problème du voyageur de commerce avec cueillettes et livraisons et contrainte de chargement de type "dernier entré, premier sorti" ou $TSPPDL$ [98]. Dans ce problème, le véhicule possède une seule pile de longueur infinie.

Une première méthode exacte par séparation et évaluation ("branch-and-bound") pour ce problème est présentée dans [27]. Cette approche est basée sur le "additive branch-and-bound", précédemment appliqué au problème du voyageur de commerce avec contraintes de précédence ou $TSP-PC$ [70]. L'algorithme calcule des bornes inférieures pour le $TSP-PC$ en considérant des relaxations de type affectation et arbre de recouvrement minimal. Les auteurs dans [27] ajoutent des critères pour retirer certains arcs ne pouvant mener à des solutions réalisables afin d'améliorer les bornes inférieures, ce qui leur permet de résoudre des instances ayant jusqu'à 43 sommets.

Une méthode de séparation et évaluation avec plans sécants ("branch-and-cut") est proposée dans [45], toujours pour le $TSPPDL$. Les auteurs présentent plusieurs formulations mathématiques pour la contrainte "dernier entré, premier sorti". De plus, ils dérivent plusieurs inégalités valides qui tiennent compte de la structure du problème. Les résultats améliorent ceux dans [27] et des instances de plus grande taille sont résolues.

Les auteurs dans [29] ont récemment abordé le $TSPPDL$ avec une recherche à voisinage variable [125]. Ils présentent plusieurs opérateurs qui tiennent compte de la structure du problème tout en préservant la réalisabilité des solutions. Ils rapportent des résultats sur plusieurs instances tests dérivées d'instances classiques du problème du voyageur de commerce.

Dans [108], les auteurs présentent une structure de données arborescente pour tenir compte de l'ordre “dernier entré, premier sorti”. Ils décrivent ensuite plusieurs adaptations d'opérateurs classiques pour les problèmes de tournées de véhicules qui sont intégrées à l'intérieur d'une recherche à voisinages variables.

Une problématique quelque peu différente est rapportée dans [11]. Ici, les auteurs se concentrent sur une variante du problème du voyageur de commerce avec contrainte de chargement de type “dernier entré, dernier sorti” où deux types de marchandises doivent être recueillis ou livrés à des clients. Les deux types de marchandises sont chargés sur une seule pile. Il est toutefois possible de réamménager les marchandises dans la pile afin de minimiser le temps de parcours. L'objectif du problème est donc de desservir tous les clients tout en minimisant le temps total de parcours plus le temps nécessaire pour les réaménagements. Les auteurs proposent une formulation en nombre entiers qu'ils résolvent à l'aide d'une méthode de séparation et évaluation avec plans sécants. Des instances contenant jusqu'à 25 sommets sont ainsi résolues.

2.6 Cueillettes et livraisons avec contrainte “premier entré, premier sorti”

Une variante naturelle des problèmes de cueillettes et livraisons avec contrainte de type “dernier entré, premier sorti” est la variante avec contrainte de type “premier entré, premier sorti”. Ainsi, c'est l'item le plus ancien ayant été recueilli qui doit être livré en premier. Un tel mode d'opération se rencontre dans le transport de véhicules par bateau (les traversiers, par exemple) où les véhicules entrent d'un côté du bateau et en ressortent de l'autre [64]. La figure 2.5 présente une tournée avec cinq clients qui est valide pour une contrainte de type “premier entré, premier sorti”.

Cette problématique fut introduite dans [64], où les auteurs fournissent une formulation mathématique en nombre entiers ainsi que deux métaheuristiques, soit une recherche tabou probabiliste et une recherche locale itérée. Plusieurs opérateurs spécialisés préservant la réalisabilité de la solution sont employés à l'intérieur de ces métaheuristiques, tels que l'échange de paires de clients, l'échange de séquences, le déplacement d'une

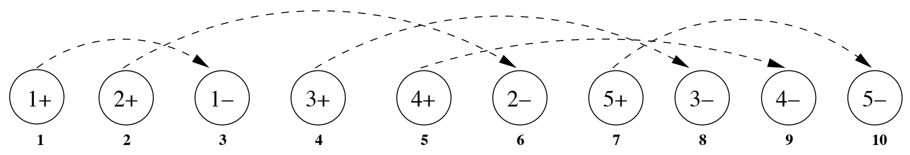

Figure 2.5 – Tournée pour un véhicule avec contrainte “premier entré, premier sorti”

paire de clients, le déplacement d’une séquence et un dernier opérateur spécialisé faisant appel à un algorithme de programmation dynamique. Des instances contenant jusqu’à 24 sommets sont résolues avec la formulation exacte, tandis que des tests sur des instances ayant jusqu’à 750 sommets sont réalisés avec les approches métaheuristiques.

Une approche exacte est proposée dans [27] faisant appel à une méthode de séparation. Les auteurs utilisent l’algorithme précédemment développé pour le $TSPPDL$ mais définissent des critères de branchement différents tenant compte de la structure du problème. Ils peuvent ainsi résoudre des instances contenant jusqu’à 38 sommets.

Une méthode de séparation et évaluation avec plans sécants est aussi rapportée dans [43]. De nombreuses inégalités valides ainsi que des règles de branchement sont déduites de la structure “premier entré, premier sorti”, ce qui permet d’accélérer la résolution. Les auteurs obtiennent des résultats supérieurs à ceux dans [27] et peuvent résoudre des instances contenant jusqu’à 50 sommets.

2.7 Transport de voitures

Une problématique particulièrement intéressante est celle du transport des voitures neuves aux concessionnaires. Elle consiste à charger des véhicules sur des remorques à flancs porteurs afin de les livrer aux concessionaires. La figure 2.6 présente ainsi une remorque chargée de 8 véhicules. La complexité du problème vient de ce qu’une remorque peut effectuer plusieurs livraisons à des concessionnaires différents et que chaque véhicule doit pouvoir être livré sans déplacer les autres. Cette contrainte de type “dernier entré, premier sorti” a une importance capitale car la manipulation des véhicules peut engendrer des bris qui font grandement augmenter les coûts de livraison. Le problème de chargement est ici particulièrement difficile car il implique la manipulation de formes géométriques non convexes. Toutefois, plusieurs auteurs relaxent le problème pour qu'il soit plus facile à résoudre. Dans [1], les auteurs se concentrent spécifiquement sur le problème de chargement, tandis que dans [151], les auteurs proposent une heuristique basée sur une formulation mathématique en nombre entiers où la contrainte de chargement est relaxée et transformée en une contrainte de capacité standard. Récemment, le problème de chargement a été relaxé en un problème de "Bin Packing" avec contraintes de précédence [55].

2.8 Tournées stochastiques

La classe des problèmes de tournées de véhicules stochastiques (SVRP) se caractérise par un aspect aléatoire associée à une partie des données, comme les demandes des clients, les temps de parcours ou les temps de service qui ne sont pas connus avec certitude mais plutôt caractérisés par une distribution de probabilités. Parfois, c'est l'ensemble des clients à visiter qui est sujet à une distribution de probabilités.

L'aspect probabiliste de ces problèmes les rend intrinsèquement différents de leur contre-partie déterministe. Dans la plupart des cas, des hypothèses sont émises sur les aspects aléatoires pour en faciliter la résolution. Par exemple, on assume souvent que la demande est une variable aéatoire qui obéit à une loi de Poisson. Même avec de telles hypothèses, certaines propriétés fondamentales de la contre-partie déterministe ne sont malheureusement plus vérifiées ce qui rend les problèmes de nature probabiliste beaucoup plus difficiles à résoudre.

La programmation stochastique à deux étapes est une manière naturelle d'aborder

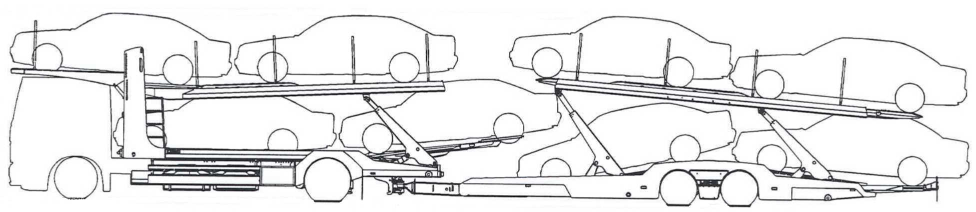

Figure 2.6 – Problème de chargement de voitures

le problèmes de tournées stochastiques. Au premier niveau, une solution planifiée est construite “a priori” [16], sans tenir compte des aspects aléatoires. Par la suite, lorsque la solution est exécutée au second niveau et que les valeurs des variables aléatoires sont observées, un *recours* est calculé afin de tenir compte du coût des actions correctrices qui doivent être entreprises. L’objectif est donc une somme de coûts déterministes et de coûts espérés (recours).

Pour le problème de tournées avec demandes stochastiques, par exemple, une solution à la première étape est d’abord construite permettant de desservir tous les clients. Lorsque les demandes sont révélées à la seconde étape, il est alors possible que la demande totale sur certaines routes excède la capacité du véhicule. Un recours possible serait alors d’exécuter les routes jusqu’au point de rupture. À ce moment, les véhicules doivent retourner au dépôt pour décharger leur marchandise avant de poursuivre leur route à partir du point de rupture. Cette action correctrice ou recours entraîne évidemment des coûts additionnels.

Il existe une multitude de problèmes de tournées stochastiques et nous référons le lecteur à la revue de littérature dans [78]. Le problème auquel nous nous intéressons dans cette thèse se rapporte au cas où la demande est stochastique (VRPSD) qui est l’un des plus étudiés. Il a été introduit dans [153] où les auteurs proposent un algorithme basé sur l’algorithme classique de *savings* de Clarke et Wright [36].

Les méthodes exactes récentes pour résoudre ce problème sont basées sur l’algorithme L-Shaped en nombre entiers [100]. Celui-ci repose lui-même sur la méthode rapportée dans [158] pour la résolution de programmes stochastiques continus. L’idée de base est de relaxer la composante aléatoire dans l’objectif du modèle et d’y introduire à la place une variable qui représente le coût espéré du recours. Le modèle est ensuite résolu par une méthode de type séparation et évaluation avec plans sécants (“branch-and-cut”). En particulier, des coupes dites d’optimalité sont générées afin de borner inférieurement le recours. L’addition de “Lower Bounding Functionals” [85], qui bornent également le recours, améliore encore davantage le comportement général de la méthode.

Cet algorithme fut appliqué pour la première fois sur un VRP avec clients et demandes stochastiques dans la thèse de Séguin [146] et ensuite dans [77]. Certaines instances avec seulement 2 véhicules et 9 sommets se sont révélées difficiles à résoudre car la présence de clients stochastiques complexifie beaucoup le problème. En présence seulement de demandes stochastiques, les auteurs réussissent toutefois à résoudre des instances ayant jusqu'à 70 sommets et 2 véhicules.

L'algorithme L-Shape en nombre entiers fut ensuite amélioré dans [85, 90, 101]. Dans [85], le concept de routes partielles est introduit, ce qui permet de produire un nouveau type d'inégalités. Les auteurs appliquent leur algorithme au problème à un seul véhicule et sont capables de résoudre des instances ayant jusqu'à 90 sommets. L'idée des routes partielles est généralisée dans [101] pour plusieurs véhicules et on y présente des résultats pour des demandes obéissant à des lois de Poisson et des lois Normale. Finalement, les auteurs dans [90] introduisent trois nouveaux types de "Lower Bounding Functionals" en exploitant et étendant les idées présentées dans [101].

CHAPITRE 3

A BRANCH-AND-CUT ALGORITHM FOR THE PICKUP AND DELIVERY TRAVELING SALESMAN PROBLEM WITH MULTIPLE STACKS

Ce chapitre se penche sur l'élaboration d'une méthode de séparation et évaluation avec plans sécants ("branch-and-cut") pour le problème de confection d'une seule tournée avec cueillettes et livraisons pour un véhicule dont le conteneur est divisé en piles de hauteur (ou longueur) fixe. Une contrainte de type "dernier entré, premier sorti" est imposée sur l'ordre des livraisons, de telle sorte qu'un item ne peut être livré que s'il se trouve sur le dessus d'une pile. Plusieurs formulations et inégalités valides pour ce problème sont présentées dans l'étude.

Ce travail fut réalisé dans le cadre d'une bourse de recherche à l'étranger que j'ai obtenue pour un stage en Italie qui s'est tenu de février 2010 à juin 2011 et qui m'a permis de collaborer avec Claudia Archetti et Maria Grazia Speranza de l'Université de Brescia. L'article fut publié dans le journal *Networks* :

Côté J.-F., Archetti C., Speranza M. G., Gendreau M., Potvin J.-Y., *A Branch-And-Cut Algorithm for the Pickup and Delivery Traveling Salesman Problem with Multiple Stacks*, Networks 60, pages 212-226, 2012.

A Branch-And-Cut Algorithm for the Pickup and Delivery Traveling Salesman Problem with Multiple Stacks

Jean-François Côté †§, Claudia Archetti, Maria Grazia Speranza,
Michel Gendreau ‡§, Jean-Yves Potvin †§

†Département d'informatique et de recherche opérationnelle, Université de Montréal,
C.P. 6128, succ. Centre-Ville, Montréal, Québec, Canada H3C 3J7.

‡Département de mathématiques et de génie industriel, École Polytechnique de Montréal
C.P. 6079, succ. Centre-Ville, Montréal, Québec, Canada H3C 3A7.

§CIRRELT, Université de Montréal,
C.P. 6128, succ. Centre-Ville, Montréal, Québec, Canada H3C 3J7.

Dipartimento Metodi Quantitativi, Università degli Studi di Brescia
Contrada Santa Chiara 50, 25122 Brescia, Italia

Abstract

This paper studies the pickup and delivery traveling salesman problem with multiple stacks. The vehicle contains a number of (horizontal) stacks of finite capacity for loading items from the rear of the vehicle. Each stack must satisfy the last-in-first-out constraint which states that any new item must be loaded on top of a stack and any unloaded item must be on top of its stack. A branch-and-cut algorithm is proposed for solving this problem. Computational results are reported on different types of randomly generated instances, as well as on classical instances for some well-known special cases of the problem.

Keywords : Traveling salesman, pickup, delivery, loading, multiple stacks.

3.1 Introduction

While the vehicle routing problem has been studied for a long time and hundreds of papers have dealt with solution algorithms for different variants of this problem, much less attention has been paid to problems with both routing and loading aspects. As vehicle routing problems are hard problems, integrating routing and loading is a challenging task. However, this is the kind of problems that carriers must face on a daily basis. By solving the routing and loading problems separately, grossly sub-optimal solutions are obtained. For example, if the loading problem is solved first, the optimization of the loading space is likely to lead to severe constraints on the routing aspect of the problem and to time consuming unloading and reloading operations along the route.

In this paper, we study the pickup and delivery traveling salesman problem with multiple stacks (*PDTSPMS*), where a single vehicle is available to serve a set of customer requests. Each request consists in picking up an item at a given location and delivering it to a different location. The vehicle contains a number of independent horizontal stacks (rows) of finite length (capacity) for loading items from the rear. Each stack must satisfy the last-in-first-out (LIFO) property which states that any new item must be loaded on top of a stack and any unloaded item must be on top of its stack. The objective is to serve all requests at minimum cost, where the cost corresponds to the traveling distance.

The *PDTSPMS* is a generalization of the double traveling salesman problem with multiple stacks (*DTSPMS*) introduced in [131]. In the *DTSPMS*, a number of LIFO stacks are available to stock items of equal length. In a valid solution, the vehicle must collect all pickups and return to the depot before performing the deliveries. In [131], the authors propose a mathematical model and some metaheuristics to solve the *DTSPMS*, while new neighborhood structures are considered in [68]. A large neighborhood search heuristic developed primarily for the *PDTSPMS* is also successfully applied to the *DTSPMS* in [49]. Exact solution methods are reported in [111, 132]. The first work is based on a branch-and-cut procedure, while the second one matches the *k* best tours for the pickups and the *k* best tours for the deliveries. An additive branch-and-bound method is proposed in [28] for a special case with only two stacks. Finally, some theoretical properties of the DTSPMS are derived in [31] and a heuristic algorithm is developed to exploit these properties.

The *PDTSPMS* is also a generalization of the pickup and delivery TSP with LIFO loading constraints (*PDTSPL*), where the vehicle contains a single LIFO stack of infinite capacity. This problem is solved with either heuristic or exact methods in [27, 29, 45]. A polyhedral study and a branch-and-cut algorithm are also described in [63]. Other variants of pickup and delivery traveling salesman problems with different types of loading policies, like first-in-first-out (FIFO), are reported in [11, 27, 43, 64].

Many types of combined vehicle routing and loading problems can be found in the scientific literature when the routing problem is made of pickup- or (exclusive) delivery-only operations. A tabu search and an ant colony heuristic are used to solve a problem with two dimensional loading constraints in [71, 76], while an exact approach is proposed in [89]. The problem is extended to three dimensional loading constraints and is solved with metaheuristics in [72]. Other types of loading constraints have also been considered in [59, 75, 157]. For an overview of this line of research, the reader is referred to [88].

In this paper, several classes of valid inequalities for the *PDTSPMS* are derived and exploited within a branch-and-cut algorithm. The computational results show that the size of instances that can be solved with this algorithm strongly depends on the specific type of instance considered. While instances with up to 43 nodes can be solved in some test sets, instances with 27 nodes cannot be solved exactly in some other sets. Since the branch-and-cut algorithm proposed in this work is the first exact algorithm for the *PDTSPMS*, *DTSPMS* instances have been used to compare its performance with other exact methods, namely the algorithms of Lusby et al. [132] and Petersen et al. [111], for a variable number of stacks, and the algorithm of Carrabs et al. [28] for two stacks. The results show that our algorithm outperforms these previous methods, even if it was not specifically designed for the *DTSPMS*.

The paper is organized as follows. In Section 3.2, three different formulations for the *PDTSPMS* are proposed. Then, several classes of valid inequalities are presented in Section 3.3 and separation procedures are described in Section 3.4. Section 3.5 is devoted to the branch-and-cut algorithm and computational results are reported in Section 3.6.

3.2 Problem formulation

The *PDTSPMS* can be formally stated as follows. Let $G = (V, A)$ be a complete directed graph where $V = \{0, 1, ..., 2n+1\}$ is the node set and A is the arc set. Nodes 0 and $2n+1$ denote the depot at the start and at the end of the tour, respectively, while nodes i and $n+i$ are the pickup and delivery locations of customer i, $1 \leq i \leq n$. Each request implies to pickup an item at location i and to deliver it at location $n+i$. We denote $P = \{1, ..., n\}$ and $D = \{n+1, ..., 2n\}$ the set of pickup and delivery locations, respectively. An item of length d_i is associated with pickup location $i \in P$. For simplicity, we will refer in the following to the item picked up at location i as item i. An item of length $d_0 = d_{2n+1} = 0$ is associated with the depot and an item of length $-d_i$ with delivery location $n+i \in D$. We also have a cost c_{ij} associated with each arc $(i, j) \in A$. A single vehicle is available to serve all requests. This vehicle contains a set $M = \{1, 2, ..., m\}$ of LIFO stacks, each of capacity Q, to transport the items between pickup and delivery locations. The goal is to serve all customers with a least-cost route, starting at node 0 and ending at node $2n+1$, while satisfying the side constraints.

To model the problem as a mathematical program, we introduce the following notation :

- $\bar{S} = V \backslash S$, $S \subseteq V$;
- $x(S) = \sum_{i,j \in S} x_{ij}$, $S \subseteq V$;
- $x(\delta^+(S)) = \sum_{i \in S, j \notin S} x_{ij}$, $S \subseteq V$;
- $x(\delta^-(S)) = \sum_{i \notin S, j \in S} x_{ij}$, $S \subseteq V$;
- $x(\delta(S)) = x(\delta^+(S)) + x(\delta^-(S))$, $S \subseteq V$;
- $x(i, S) = \sum_{j \in S} x_{ij}$, $i \in V$, $S \subseteq V$;
- $x(S, i) = \sum_{j \in S} x_{ji}$, $i \in V$, $S \subseteq V$;

- $\pi(j) = j,\ j \in P$;
- $\pi(n+j) = j,\ n+j \in D$;
- $\pi(S) = \{i \in P | n+i \in S\}$;
- $\sigma(S) = \{n+i \in D | i \in S\}$;
- $S_j = \{S' \subset P \cup D | j \in S'$ and $n+j \notin S'\}$;
- $S_{n+j} = \{S' \subset P \cup D | j \notin S'$ and $n+j \in S'\}$;
- Ω is a collection of subsets $S \subset V$ such that, for each subset S, we have $0 \in S, 2n+1 \notin S$ and there exists a pickup $i \notin S$ for which $n+i \in S$;

We also define the following decision variables :

- x_{ij} is 1 if node j is visited immediately after node i, 0 otherwise, $i, j \in V$, $i \neq j$;
- y_{ik} is 1 if item i is loaded in stack k, 0 otherwise, $i \in P$, $k \in M$;
- $0 \leq s_{ik} \leq Q$ is the load of stack k upon leaving node i, $i \in V$, $k \in M$ (with $s_{0k} = 0$, $k \in M$).

The *PDTSPMS* can now be formulated as follows :

$$\min \sum_{(i,j) \in A} c_{ij} x_{ij} \tag{3.1}$$

subject to

$$\sum_{j \in V} x_{ij} = 1,\ \ i \in P \cup D \cup \{0\} \tag{3.2}$$

$$\sum_{j \in V} x_{ji} = 1,\ \ i \in P \cup D \cup \{2n+1\} \tag{3.3}$$

$$x(S) \leq |S| - 1,\ \ S \subset P \cup D, |S| \geq 2 \tag{3.4}$$

$$x(S) \leq |S| - 2,\ \ S \in \Omega \tag{3.5}$$

$$\sum_{k \in M} y_{ik} = 1, \quad i \in P \tag{3.6}$$

$$s_{jk} \geq s_{ik} + d_j y_{\pi(j)k} - Q(1 - x_{ij}), \quad i \in V,\ j \in P \cup D,\ k \in M \tag{3.7}$$

$$s_{jk} \leq s_{ik} + d_j y_{\pi(j)k} + Q(1 - x_{ij}), \quad i \in V,\ j \in P \cup D,\ k \in M \tag{3.8}$$

$$s_{(n+j)k} \geq s_{jk} - d_j y_{jk} - Q(1 - y_{jk}), \quad j \in P,\ k \in M \tag{3.9}$$

$$s_{(n+j)k} \leq s_{jk} - d_j y_{jk} + Q(1 - y_{jk}), \quad j \in P,\ k \in M \tag{3.10}$$

$$s_{0k} = 0, \quad k \in M \tag{3.11}$$

$$0 \leq s_{ik} \leq Q, \quad i \in V,\ k \in M \tag{3.12}$$

$$x_{ij} \in \{0, 1\}, \quad i, j \in V. \tag{3.13}$$

This model will be referred to as the *network formulation*. In this formulation, the objective function (3.1) is aimed at minimizing the total cost which corresponds to the distance traveled by the vehicle. Each node is visited exactly once through constraints (3.2) and (3.3). Constraints (3.4) impose connectivity on the route. The precedence constraints between the pickup and delivery locations (3.5) are taken from [139]. Constraints (3.6) state that each item is loaded in exactly one stack. Constraints (3.7) and (3.8) define the status of the stacks after each pickup and delivery. The LIFO loading constraints are stated in (3.9) and (3.10). Finally, constraints (3.11) and (3.12) take into account the capacity of each stack.

In the next sections, alternative formulations are considered to avoid the use of (3.7), (3.8), (3.9) and (3.10) which are similar to the Miller, Tucker and Zemlin constraints [123]. These constraints are known to generate poor linear relaxations.

3.2.1 Flow formulation

We can extend the flow formulation proposed in [45] for the *PDTSPL* by adding a stack index to the variables. Let f_{ij}^k be the flow circulating on arc (i, j) for stack k. A pickup operation increases the flow whereas a delivery operation decreases it. Thus, the flow on arc (i, j) for stack k increases by d_i if item i is loaded on stack k, as indicated in constraints (3.14) below. When item i is not loaded on stack k, the flow remains unchanged.

To satisfy the LIFO constraints, the load on stack k before serving $i \in P$ and after serving $n+i \in D$ must be the same. Constraints (3.15) and (3.16) guarantee the LIFO policy only for items loaded on the same stack. If item i is on stack k, inequalities (3.15) and (3.16) state that the flow to $i \in P$ and the flow from $n+i \in D$ must be the same. On the other hand, if item i is not on stack k, constraints (3.15) and (3.16) are not binding. The flow formulation is then given by (3.1)-(3.6), (3.13) and (3.14)-(3.17).

$$\sum_{j \in N} f_{ji}^k - \sum_{j \in N} f_{ij}^k = d_i y_{\pi(i)k} \quad i \in P \cup D \tag{3.14}$$

$$\sum_{j \in N} f_{ji}^k - \sum_{j \in N} f_{(n+i)j}^k \leq Q(1 - y_{ik}) \quad i \in P,\ k \in M \tag{3.15}$$

$$\sum_{j \in N} f_{ji}^k - \sum_{j \in N} f_{(n+i)j}^k \geq -Q(1 - y_{ik}) \quad i \in P,\ k \in M \tag{3.16}$$

$$0 \leq f_{ji}^k \leq Q \quad (i,j) \in A,\ k \in M. \tag{3.17}$$

3.2.2 Infeasible path formulation

Let p be a path in G. The formulation presented here is inspired by the work on the DTSPMS in [132] and requires no additional variables with regard to the network formulation. The infeasible path formulation is given by (3.1)-(3.6), (3.13) plus a set of constraints for eliminating all paths which are infeasible with regard to the LIFO constraints (3.9)-(3.10) or capacity constraints (3.12). These so-called *path inequalities* are of the form :

$$x(p) \leq |p| - 1 \quad p \in \Phi \tag{3.18}$$

where $x(p)$ is the sum, over all arcs in path p, of the x variables associated with these arcs, $|p|$ is the number of arcs in path p and Φ denotes the set of all LIFO or capacity infeasible paths.

As the number of LIFO or capacity infeasible paths can be huge, inserting a cut for each infeasible path is inefficient. Thus, we relax these constraints and check for possible violations to the LIFO or capacity constraints by solving a packing problem (see below) whenever a valid pickup and delivery traveling salesman tour is found. If the packing problem is infeasible, we add a constraint of type (3.18) to remove this path from the solution space. Conversely, if a feasible packing is found, we have a feasible solution to the *PDTSPMS*.

3.2.2.1 Packing problem

Let p be a path. We want to check whether p is LIFO or capacity infeasible. From p it is possible to compute a set I of incompatible pairs of items (i, j), namely those items which cannot be packed on the same stack due to the LIFO constraints. This computation can be done in $O(n^2)$ by looking at each pair of nodes $i, j \in P$ visited in p, for which $i < j < n+i < n+j$ or $j < i < n+j < n+i$, where $i < j$ means that i is visited before j in p. For a set of items $\{i_1, i_2, ..., i_k\}$ for which $i_1 < i_2 < ... < i_k < n+i_1 < n+i_2 < ... < n+i_k$, we say that items $i_1, i_2, ..., i_k$ cross each other. We denote by O the list of nodes in the path sequenced according to their visit order. Let also a_{io} be equal to 1 if item i is in the vehicle when node $o \in O$ is visited, 0 otherwise. Hence, $a_{io} = 1$ if item i is picked up before o and delivered after o (i.e., $i < o < n+i$).

In Figure 3.1, an example for a path with 5 pickups and 5 deliveries is depicted. The list O is equal to $\{1, 2, n+1, n+2, 3, 4, 5, n+4, n+5, n+3\}$. With regard to incompatibilities, items 1 and 2 cross each other and cannot be on the same stack. The same applies to items 4 and 5. The set of incompatible pairs is then $I = \{(1,2), (4,5)\}$.

The objective is to find a feasible packing for the route, that is, an assignment of each item to a stack such that the LIFO and capacity constraints are satisfied. Let z_{ik} be

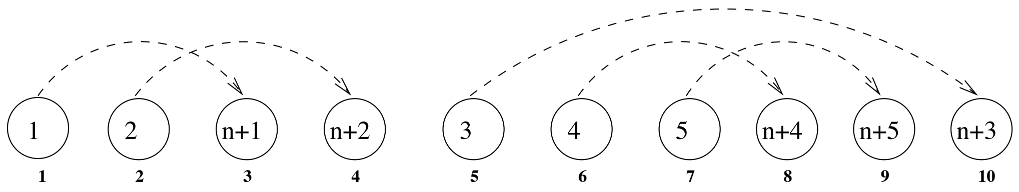

Figure 3.1 – Example of a pickup and delivery sequence

a binary variable equal to 1 if item *i* is loaded on stack *k*. The packing problem is then :

$$\sum_{k \in M} z_{ik} = 1 \quad i \in P \tag{3.19}$$

$$z_{ik} + z_{jk} \leq 1 \quad (i,j) \in I \tag{3.20}$$

$$\sum_{i \in P} a_{io} d_i z_{ik} \leq Q \quad k \in M,\ o \in O \tag{3.21}$$

$$z_{ik} \in \{0,1\} \quad i \in P, k \in M \tag{3.22}$$

In this formulation, constraints (3.19) associate a single stack with each item. Constraints (3.20) prohibit items on the same stack to cross each other. Capacity constraints (3.21) state that the sum of the lengths of items which are on the same stack at the same time must be less than or equal to the capacity of the stack.

We conclude this section by noting that a similar problem arises in the *DTSPMS* (where deliveries can only be done once all items have been picked up). The authors in [31, 154] note that the packing problem in the *DTSPMS* is NP-hard, as it is equivalent to a *Q*-bounded graph coloring problem. However, they showed that the problem is polynomially solvable when there is no capacity constraint. Since our packing problem is much more complex, due to deliveries which can precede pickups, it is very unlikely that a polynomial time algorithm exists.

3.3 Valid inequalities

This section presents valid inequalities for the *PDTSPMS*. We first present the inequalities that are inherited from the classical pickup and delivery traveling salesman problem (*PDTSP*), where the loading aspect is not considered. Then, we introduce new inequalities obtained by adapting some inequalities initially proposed for the vehicle routing problem (*VRP*) and the *PDTSPL* .

3.3.1 Inequalities for the PDTSPL

Given that the *PDTSPMS* extends the *PDTSP*, all known inequalities for this problem can be used. Relevant work for the *PDTSP* can be found in [42, 63, 140, 141]. Here, we use the set of inequalities in [45], as described below.

(a) A first class of inequalities is obtained through the predecessor and successor inequalities for the precedence-constrained $ATSP$ [8] :

$$x(S)+\sum_{i\in S}\sum_{j\in \bar{S}\cap\pi(S)} x_{ij}+\sum_{i\in S\cap\pi(S)}\sum_{j\in\bar{S}\setminus\pi(S)} x_{ij}\leq |S|-1 \quad S\subseteq P\cup D \tag{3.23}$$

$$x(S)+\sum_{i\in \bar{S}\cap\sigma(S)}\sum_{j\in S} x_{ij}+\sum_{i\in\bar{S}\setminus\sigma(S)}\sum_{j\in S\cap\sigma(S)} x_{ij}\leq |S|-1 \quad S\subseteq P\cup D \tag{3.24}$$

(b) For a given ordered set $S=\{i_1,i_2,...,i_k\}\subseteq V$ with $k\geq 3$, the following cycle inequalities for the $ATSP$ have been proposed in [83] :

$$\sum_{h=1}^{k-1} x_{i_h,i_{h+1}}+x_{i_k,i_1}+2\sum_{h=2}^{k-1} x_{i_h,i_1}+\sum_{h=3}^{k-1}\sum_{l=2}^{h-1} x_{i_h,i_l}\leq k-1 \tag{3.25}$$

$$\sum_{h=1}^{k-1} x_{i_h,i_{h+1}}+x_{i_k,i_1}+2\sum_{h=3}^{k} x_{i_1,i_h}+\sum_{h=4}^{k}\sum_{l=3}^{h-1} x_{i_h,i_l}\leq k-1 \tag{3.26}$$

The previous inequalities can be strengthened by considering sets $\pi(S)$ and $\sigma(S)$ to obtain the cycle inequalities [42] :

$$\begin{aligned}\sum_{h=1}^{k-1} x_{i_h,i_{h+1}}+x_{i_k,i_1}+2\sum_{h=2}^{k-1} x_{i_h,i_1}+\sum_{h=3}^{k-1}\sum_{l=2}^{h-1} x_{i_h,i_l}+&\\ \sum_{n+i_p\in\bar{S}\cap\sigma(S)} x_{n+i_p,i_1}\leq k-1&\end{aligned} \tag{3.27}$$

$$\sum_{h=1}^{k-1} x_{i_h,i_{h+1}} + x_{i_k,i_1} + 2\sum_{h=3}^{k} x_{i_1,i_h} + \sum_{h=4}^{k}\sum_{l=3}^{h-1} x_{i_h,i_l} + \sum_{i_p \in \bar{S} \cap \pi(S)} x_{i_1,i_p} \leq k-1 \tag{3.28}$$

(c) Finally, let $U_1,...,U_k \subset P \cup D$ be mutually disjoint subsets such that $i_1,...,i_k \in P$ are customer requests for which $i_l, n+i_{l+1} \in U_l$ for $l = 1,...,k$ (where $i_{k+1} = i_1$). The precedence cycle breaking inequalities, introduced for the precedence-constrained *TSP* in [8], are also valid for the *PDTSPMS* :

$$\sum_{l=1}^{k} x(U_l) \leq \sum_{l=1}^{k} |U_l| - k - 1 \tag{3.29}$$

3.3.2 LIFO inequalities

In [45], the authors present a class of inequalities for the *PDTSPL* that satisfies the LIFO constraints. Note again that this problem corresponds to a *PDTSPMS* with a single stack of infinite capacity. In the *PDTSPL*, items are not allowed to cross each other. To impose a LIFO policy, the set Γ is defined as a collection of sets $S \subset P \cup D$ such that there exist $j \in S$ and $n+j \notin S$ or $j \notin S$ and $n+j \in S$. In [45] the authors proved that inequalities of type (3.30) are sufficient to prohibit all LIFO infeasible solutions. A graphical representation is shown in Figure 3.2.

$$x(i,S) + x(S) + x(S,n+i) \leq |S| \quad S \in \Gamma, i, n+i \notin S, i \in P. \tag{3.30}$$

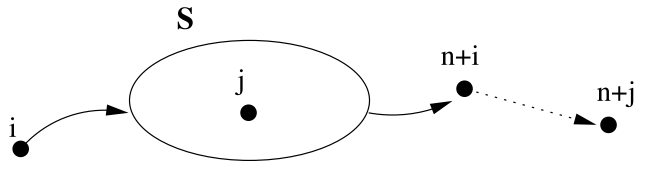

Figure 3.2 – Forbidden pattern for LIFO constraint with one stack

Note that inequalities (3.30) are not valid for the *PDTSPMS* because multiple stacks are available. However, they can be adapted to the *PDTSPMS*. We first show how inequalities (3.30) can be extended to the *PDTSPMS* with two stacks. Then we derive a formulation for an arbitrary number of stacks.

In the *PDTSPMS*, items can be allowed to cross each other depending on the number of stacks. For example, consider a vehicle with two stacks. A situation where two items cross each other could be acceptable because the first item can be loaded on the first stack and the other on the second stack. However, a situation where three items cross each other is forbidden because three stacks would then be required. We conclude that for a vehicle with k stacks, a set of $k+1$ or more items cannot cross each other, otherwise an infeasible solution is obtained.

Let us show how constraints (3.30) can be extended to two stacks. We suppose that the LIFO constraint is violated and that the solution is feasible for the *PDTSP*. It means that we can find a path where three items cross each other. In other words, there exist $i, j, k \in P$ such that $i < j < k < n+i < n+j < n+k$. We can exclude this path by using a constraint of type (3.18). However, this approach is rather weak because inequality (3.18) cuts a single path. A different inequality is proposed here based on an extension of (3.30). Let us define two subsets $S_j, S_k \subset P \cup D$ such that $j, k \in S_j; i, n+i, n+j, n+k \notin S_j$ and $k, n+i \in S_k; i, j, n+j, n+k \notin S_k$. A prohibited pattern is shown in Figure 3.3. By extending (3.30) to this pattern we obtain :

$$\begin{aligned} x(i,S_j)+x(S_j)+x(S_j,n+i)+ \\ x(j,S_k)+x(S_k)+x(S_k,n+j) \leq \\ |S_j|+|S_k|+1. \end{aligned} \tag{3.31}$$

Inequality (3.31) forbids all paths from i to $n+i$ going through j and k (but not through $n+j$ and $n+k$) and, at the same time, forbids all paths from j to $n+j$ going through k and $n+i$ (but not through i and $n+k$).

Constraint (3.31) can be extended to impose LIFO constraints in the general case

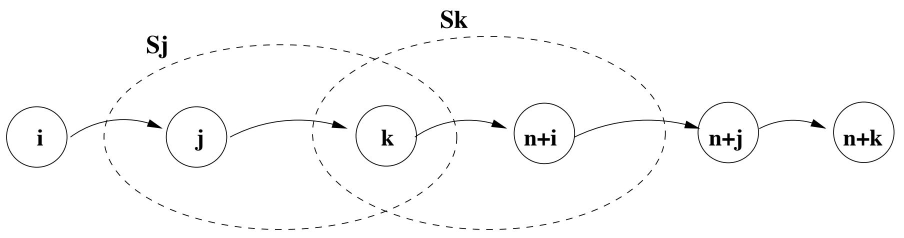

Figure 3.3 – Forbidden pattern for LIFO constraint with two stacks

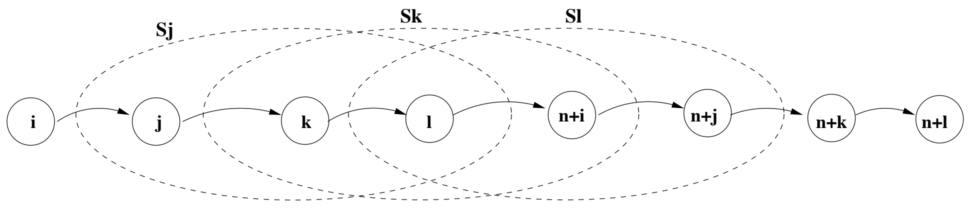

Figure 3.4 – Forbidden pattern for LIFO constraint with three stacks

with an arbitrary number of stacks M. Let $\mathscr{L} = \{i_1, i_2, ..., i_{M+1} \in P, i_j \neq i_k\}$ be a set of pickup locations where the associated items mutually cross each other (i.e., each item in $\mathscr{L}$ crosses all other items in $\mathscr{L}$). This situation induces the following delivery pattern : $i_1 < i_2 < ... < i_{M+1} < n+i_1 < ... < n+i_{M+1}$. Also, let S_{i_h}, $2 \leq h \leq M+1$, denote a set with nodes i_h to i_{M+1} and $n+i_1$ to $n+i_{h-2}$, but without nodes i_1 to i_{h-1} and $n+i_{h-1}$ to $n+i_{M+1}$. Note that S_{i_2} does not contain the delivery nodes $n+i_1$ to $n+i_{M+1}$. An example with three stacks is shown in Figure 3.4. Since all these requests cross each other, none of them can be on the same stack and at least $M+1$ different stacks are required. Thus, we can exclude all paths with this delivery pattern by extending inequality (3.31) in the following way :

$$\sum_{h=1}^{M} [x(i_h, S_{i_{h+1}}) + x(S_{i_{h+1}}) + x(S_{i_{h+1}}, n+i_h)] \leq \sum_{h=2}^{M+1} |S_{i_h}| + M - 1. \tag{3.32}$$

Using the same line of reasoning for inequality (3.31), inequality (3.32) forbids all paths from i_h to $n+i_h$ going through nodes i_{h+1} to i_{M+1} and $n+i_1$ to $n+i_{h-1}$, for $h=1,...,M+1$.

3.3.3 Capacity inequalities

In the classical capacitated *VRP*, the capacity constraints can be imposed as follow :

$$x(\delta(S)) \geq 2r(S) \;\; S \subset P \cup D, \tag{3.33}$$

where $r(S)$ is the minimum number of vehicles required to serve all customers in S. However, computing $r(S)$ is difficult as it involves solving a bin packing problem. By replacing $r(S)$ with the lower bound $\lceil|\sum_{i\in S} d_i/Q|\rceil$, we obtain the *rounded capacity inequality*

$$x(\delta(S)) \geq \lceil|\sum_{i\in S} d_i|/Q\rceil. \tag{3.34}$$

Inequality (3.34) can be easily adapted to the *PDTSPMS* by ignoring the presence of multiple stacks. The whole loading area would then become available for a total capacity of MQ. The rounded capacity inequality becomes :

$$x(\delta(S)) \geq 2\lceil|\sum_{i\in S} d_i|/(MQ)\rceil. \tag{3.35}$$

Note that we have $|\sum_{i\in S} d_i|$ in (3.35) because $d_i < 0$ for delivery locations. Unfortunately, these inequalities are not sufficient to cut all solutions that violate the capacity constraints. Consider the example shown in Figure 3.5 with two stacks and $Q=2$. The total available capacity is equal to 4 and the sum of the lengths of the items in S is also equal to 4. Thus, constraint (3.35) is satisfied. However, this solution is clearly infeasible because i and j cross each other. Thus, j and k must be in the same stack which is infeasible.

However, it is possible to derive a new set of inequalities for the capacity constraints. In the previous example, all items in conflict with i must fit in the remaining available capacity of the vehicle, which is $Q(M-1)$. By summing up the lengths of those items,

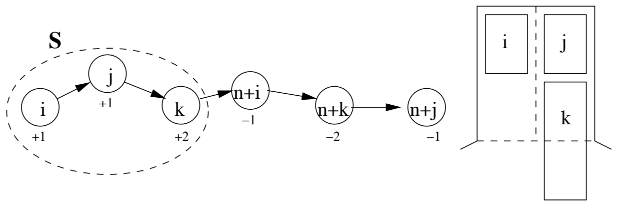

Figure 3.5 – Violated capacity constraint

we obtain a value of 3 which is larger than $Q(M-1)$, thus indicating that the solution is infeasible. The basic idea is presented in Figure 3.6, where D_1 is the sum of the lengths of items that are picked up before i and delivered before $n+i$, while P_1 is the sum of the lengths of items picked up between i and $n+i$ and delivered after $n+i$. Items corresponding to D_1 and P_1 cross i and cannot be on the same stack than i, which imply that they must fit in the $M-1$ remaining stacks.

To write this type of inequality, we define $q(S) = \sum_{i \in S} d_i$ as the sum of the lengths of items in S and $z(S)$ as the maximum between the sum of the lengths of pickup and delivery items in S :

$$z(S) = \max\{q(\pi(S)\backslash S), -q(\sigma(S)\backslash S)\}. \tag{3.36}$$

Note that $\pi(S)\backslash S$ is the set of pickup nodes not in S and for which the corresponding delivery is in S, while $\sigma(S)\backslash S$ is the set of delivery nodes not in S and for which the corresponding pickup is in S.

When $z(S)$ is greater than $Q(M-1)$, then inequality (3.37) can be added to forbid all paths from i to $n+i$ going through all nodes in S. We refer to this new type of inequalities as *conflict capacity inequalities*

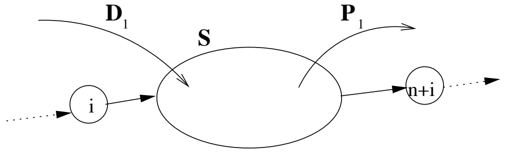

Figure 3.6 – Conflict capacity constraint

$$x(i,S)+x(S)+x(S,n+i) \leq |S| \quad S \in P \cup D,\ i,n+i \notin S,\ z(S) > Q(M-1) \tag{3.37}$$

The previous inequality can be extended by considering more items. Let $\mathscr{L} = \{i_1, i_2, ..., i_k \in P, 2 \leq k \leq M-1, i_j \neq i_k\}$ be a set where each item crosses all other items in the set. This situation induces the delivery pattern $i_1 < i_2 < ... < i_k < n+i_1 < ... < n+i_k$. Also, let S_{i_h}, $2 \leq h \leq k$, denote a set with nodes i_h to i_k and $n+i_1$ to $n+i_{h-2}$, but without nodes i_1 to i_{h-1} and $n+i_{h-1}$ to $n+i_k$.

Note that S_{i_2} does not contain the delivery nodes $n+i_1$ to $n+i_{k+1}$. Since all these items cross each other, none of them can be on the same stack and at least k different stacks are required. Now, all the other items that cross items in $\mathscr{L}$ can only fit in the $M-k+1$ remaining stacks. These items are picked up before i_1 and delivered between i_k and $n+i_1$ or they are picked up between i_k and $n+i_1$ and delivered after $n+i_k$. The former case means that the deliveries associated with these items are in the intersection $S_{i_2} \cap ... \cap S_{i_k}$, while the pickups are not in the union $S_{i_2} \cup ... \cup S_{i_k}$. In the latter case, the pickups associated with these items are rather in the intersection of these sets while the corresponding deliveries are not in the union of these sets. Denoting by $z(S_{i_2}, ..., S_{i_k})$ the maximum between the sum of the lengths of the items delivered and picked up in $S_{i_2} \cap ... \cap S_{i_k}$, we have :

$$\begin{aligned} z(S_{i_2}, ..., S_{i_k}) = \max\{&q(\pi(S_{i_2} \cap ... \cap S_{i_k})/(S_{i_2} \cup ... \cup S_{i_k})), \\ &-q(\sigma(S_{i_2} \cap ... \cap S_{i_k})/(S_{i_2} \cup ... \cup S_{i_k}))\}. \end{aligned} \tag{3.38}$$

If $z(S_{i_2}, ..., S_{i_k})$ exceeds the capacity $Q(M-k+1)$ of the remaining stacks, we have an infeasible solution which can be excluded through the following inequality :

$$\sum_{h=1}^{k}[x(i_h, S_{i_{h+1}}) + x(S_{i_{h+1}}) + x(S_{i_{h+1}}, n+i_h)] \leq \sum_{h=2}^{k}|S_{i_h}| + k - 1, \qquad \text{where } z(S_{i_2},...,S_{i_k}) > Q(M-k+1). \tag{3.39}$$

Inequality (3.39) forbids all paths from i_h to $S_{i_{h+1}}$ and from $S_{i_{h+1}}$ to $n+i_h$ going through all nodes in $S_{i_{h+1}}$, $1 \leq h \leq k-1$. It is important to observe that it does not exclude all capacity constraint violations, because the sequence of pickups and deliveries in $S_{i_2} \cap ... \cap S_{i_k}$ is not known.

We can extend inequality (3.39) for the *DTSPMS* to cover a broader range of capacity constraint violations. Given the particular structure of *DTSPMS* solutions, where all pickups are done before any delivery, we know that pick up items are served before delivery items in $S_{i_2} \cap ... \cap S_{i_k}$. Thus, all these items are in the vehicle at the same time. By summing up the demand of all pickup and delivery items in the intersection, we get :

$$\begin{aligned} z(S_{i_2},...,S_{i_k}) = q(\pi(S_{i_2} \cap ... \cap S_{i_k})/(S_{i_2} \cup ... \cup S_{i_k})) + \\ q(\sigma(S_{i_2} \cap ... \cap S_{i_k})/(S_{i_2} \cup ... \cup S_{i_k})). \end{aligned} \tag{3.40}$$

3.4 Separation procedures

3.4.1 *TSPPD* inequalities

Inequalities (3.23)-(3.29) are introduced and identified through the separation algorithms described in [45]. The reader is referred to this work for details.

3.4.2 LIFO inequalities

To separate the LIFO inequalities (3.32) we proceed as follows. For each pair $i, j \in P$, we check whether there exists a set S such that $i, n+i, n+j \notin S$, $j \in S$ and there is one unit of flow going from i to S and one unit of flow from S to $n+i$. It means that i is visited before S while $n+i$ is visited after S leading to the following delivery pattern :

$i < j < n+i < n+j$. To find this set, we use the separation procedure for (3.30) proposed in [45]. For each pair $i, j \in P$, we record the 3-tuple (i, j, S), if we can find such a set. Now, since $M+1$ requests need to cross each other to violate an inequality, we check whether the corresponding items cross each other for each ordered sequence in $\mathscr{L} = \{i_1, i_2, ..., i_{M+1} \in P, i_j \neq i_k\}$ (so that inequality (3.32) is violated).

The complexity of this separation procedure clearly depends on the complexity of the algorithm used to find the set S for each pair of nodes $i, j \in P$ and on the number of stacks M (as we need to check tuples of $M+1$ nodes). To find set S, for a given pair of nodes i and j, we have to solve a max-flow problem. To this end, we use the procedure proposed in [112] for which the worst-case performance is $O(n^2\sqrt{m})$ on a graph with n nodes and m arcs. However, in practice, the average performance is $O(n^{1.5})$. As there are n^2 pairs of pickup nodes, we obtain a complexity $O(n^2 f(n))$, where $O(f(n))$ is the complexity of the max-flow algorithm. Checking all tuples can be done in $O(n^{M+1})$. Hence, the total complexity is $O(n^{M+1} + n^2 f(n))$.

3.4.3 Conflict capacity inequalities

The separation of the conflict capacity inequalities (3.39) is similar to the LIFO inequalities, with the exception that the intersection of all sets must be determined to calculate the value of $z(S_{i_2}, ..., S_{i_k})$. This additional check increases the complexity of the separation procedure, since the sets contain $O(n)$ nodes. To reduce the computation time, this procedure is only used with $k = 2$ or 3.

The procedure finds, for each pair $(i, j) \in P$, a set S such that $i, n+i, n+j \notin S$ and $j \in S$. Then, it checks all ordered sequences in $\mathscr{L} = \{i_1, i_2, ..., i_k \in P, 2 \leq k \leq M-1, i_j \neq i_k\}$ to find k requests that cross each other. If such a sequence is found, the value $z(S_{i_2}, ..., S_{i_k})$ is calculated by scanning the nodes in the intersection $S_{i_2} \cap ... \cap S_{i_k}$, which requires at most $O(n)$ operations. The final complexity of the procedure is $O(n^{k+1} + n^2 f(n))$.

3.4.4 Rounded capacity inequalities

We use the heuristic procedure described in [140] to separate inequalities (3.35). This algorithm iteratively builds set S by first choosing a single node to initialize the set and by adding at each iteration the node which maximizes the following function $f(S)$:

$$\begin{aligned} f(S) = \lambda_1(\max\{q(\pi(S)\backslash S), -q(\sigma(S)\backslash S)\} - Qx(\delta^+(S))) + \\ \lambda_2 Q(\max\{\lceil \frac{q(\pi(S)\backslash S}{Q} \rceil, \lceil \frac{-q(\sigma(S)\backslash S)}{Q} \rceil\} - x(\delta^+(S))) + \\ \lambda_3(\min\{q(\pi(S)\backslash S), -q(\sigma(S)\backslash S)\} - Qx(\delta^+(S))). \end{aligned} \tag{3.41}$$

The algorithm is stopped when an inequality of type (3.35) is violated or when it is unlikely that such a violation will be found, as it is done in [140]. The function $f(S)$ contains the three parameters λ_1, λ_2 and λ_3. The first two are randomly set to a value in the interval $[1,5]$ and the last one in the interval $[0,1]$. These values are chosen at each restart of the heuristic. In [140], the heuristic is run several times, using each node as a starting point. However, we observed that it is relatively easy to find sets S that violate the inequality in our problem. To avoid generating the same inequality many times, we only run the heuristic five times and randomly choose the starting node each time. We refer the reader to [140] for details.

3.5 Branch-and-cut algorithm

Different branching strategies proposed in [43] were tried within our branch-and-cut algorithm and the most effective one was the strong branching. We also implemented local pools of cuts, as suggested in [43] and [113], but we observed that they were not really effective. Instead, we separated all *TSPPD* inequalities (3.23)-(3.29) and rounded capacity inequalities (3.35) at each node of the branch-and-bound tree and added them locally. If none of the previous cuts were generated, we separated the LIFO inequalities (3.32) and the conflict capacity inequalities (3.39) and added them locally. Each time a feasible *PDTSP* tour was found, we called the packing procedure to find a feasible packing. This implementation proved to be the most effective one. Unfortunately, adding cuts locally caused the separation procedures to generate the same cuts several times at different branches of the tree. For example, on the *DTSPMS* instances with 2 stacks and a capacity of 7 for each stack, an average of 110 000 cuts were generated. By adding them globally, only 4 000 cuts were generated on average. However, from a computational point of view, it was much better to add the cuts locally.

The upper bound was provided by the large neighborhood search heuristic in [49], which was run 10 times for a total of 25 000 iterations. We do not report the exact computation times for this heuristic, but these times stand between 3 and 30 seconds, depending on the instance.

3.6 Computational results

The branch-and-cut algorithm was coded in C++ and used Cplex 12 as the integer programming solver. All tests were performed on a 2.2 Ghz AMD Opteron 275 processor running Linux and the maximum computation time was set to one hour (except for some experiments in Section 3.6.3.1). In the following, the test instances are first described, then a comparison of the three models proposed in Section 3.2 is presented. Finally, the results obtained with the best model on *PDTSPMS* and *DTSPMS* instances are reported.

3.6.1 Instances

To test the branch-and-cut algorithm, new *PDTSPMS* instances were generated based on the *PDTSPL* benchmark instances in [27, 29, 45] with 23, 27, 31, 35, 39 and 43 nodes. We generated two classes of instances. In the first class *C1*, the demand of each pickup is one unit, the number of stacks is a random number between 2 and 4 and the capacity of each stack is a random number between 1 and 3. In the second class *C2*, the demand of each pickup is a random number between 1 and 10, the number of stacks is a random number between 2 and 4 and the capacity is a random number between 10 and 15. Note that the capacity value is tight to get difficult instances and obtain solutions that are different from optimal solutions to the standard *PDTSP*. We have a total of 54 instances in each class (i.e., 9 instances of each size).

3.6.2 Model comparison

We first report some results aimed at evaluating the effectiveness of the three models proposed in this work. These results are summarized in Table 3.I. M_1 refers to model (3.1)-(3.13) and (3.23)-(3.29). M_2 is given by (3.1)-(3.6), (3.13), (3.14)-(3.17) and (3.23)-(3.29), while M_3 is made of (3.1)-(3.6), (3.13), (3.18) and (3.23)-(3.29). The first group of three lines refer to the plain models. In the following three groups, the rounded capacity constraints (3.35), LIFO inequalities (3.32) and conflict capacity inequalities (3.39) are integrated, respectively. In the last group of three lines all inequalities are integrated. For both classes of instances we report the number of instances solved over a total of 54 instances (*Solved*), the average final gap on unsolved instances (*Gap*), the average root node gap (*Root*) on all instances, the average CPU time in seconds (*Time*) and the average number of path inequalities (*Path*) generated through the packing procedure in formulation M_3. Gaps are measured with regard to the optimal solution, when available, otherwise they are measured with regard to the heuristic solution found in [49].

The results show that the LIFO and rounded capacity inequalities improve slightly the gaps when compared with the conflict capacity inequalities. Moreover, the conflict capacity inequalities seem to cut off most of the LIFO- and capacity-infeasible paths. Using all inequalities brings the largest gain in terms of number of solved instances and gap. Overall, model M_3 is superior to both M_1 and M_2 when all inequalities are considered. Accordingly, formulation M_3 will be used in the following.

3.6.3 PDTSPMS results

Tables 3.II and 3.III report detailed results for formulation M_3 on all instances of both classes. The tables present the instance name (*Instance*), the number of nodes ($2n+1$), the number of stacks (M), the capacity of each stack (Q), the heuristic upper bound (*UB*), the optimal solution (*Optimal*), the gap between the final lower bound and the heuristic upper bound (*Gap*), the root node gap (*Root*), the CPU time in seconds (*Time*), the

	Class 1					*Class 2*				
Model	*Solved*	*Gap*	*Root*	*Time*	*Path*	*Solved*	*Gap*	*Root*	*Time*	*Path*
M1	21	16.1%	9.9%	192.5		15	27.3%	11.1%	355.8	
M2	24	14.3%	9.8%	273.0		17	24.4%	11.1%	457.8	
M3	20	17.8%	9.9%	279.0	2503.4	13	32.6%	11.0%	109.3	3337.5
M1 + LIFO	22	15.4%	9.8%	314.8		16	25.6%	11.1%	429.8	
M2 + LIFO	24	14.3%	9.8%	261.2		16	25.8%	11.1%	220.5	
M3 + LIFO	21	15.9%	9.8%	276.0	760.9	16	25.0%	11.0%	440.6	1180.1
M1 + Conflict	24	13.8%	9.6%	312.9		16	24.4%	11.1%	332.1	
M2 + Conflict	24	14.3%	9.8%	270.2		17	24.1%	11.1%	373.7	
M3 + Conflict	24	13.3%	9.6%	466.1	715.4	18	20.9%	11.0%	333.2	727.1
M1 + Capacity	29	5.8%	6.1%	298.0		17	15.0%	7.9%	337.9	
M2 + Capacity	28	7.1%	6.6%	319.3		16	17.4%	8.2%	158.6	
M3 + Capacity	31	5.1%	6.3%	266.7	95.4	22	10.1%	7.8%	501.1	767.1
M1 + All	30	5.5%	5.9%	261.2		19	12.9%	7.8%	436.1	
M2 + All	28	7.3%	6.5%	324.2		18	15.0%	8.0%	355.1	
M3 + All	34	4.2%	6.1%	381.9	8.3	24	8.7%	7.8%	312.2	186.5

Tableau 3.I – Model comparison

number of cuts of type (3.32), (3.34) and (3.39) (*Cuts*), the number of path inequalities (3.18) generated through the packing procedure (*Path*) and the number of nodes in the branch-and-bound tree (*Nodes*).

More instances in class *C1* are solved when compared to class *C2*, which means that class *C1* contains easier instances. We also observe that the number of added cuts for both classes of instances is quite large, although path inequalities are seldom violated. A large number of path inequalities is observed only in the few cases where the LIFO and conflict capacity inequalities prove to be inefficient.

The sets of instances *brd14051* and *nrw1379* are very difficult to solve. In fact, these instances are also very difficult to solve in their *PDTSP* version. It seems that when some *PDTSP* instance cannot be solved, the same is true for the corresponding *PDTSPMS* instance. The comparison between the heuristic and optimal solutions show that the heuristic algorithm often finds the optimum or is close to the optimum. There are, however, a few cases where the heuristic solution is far from the optimum.

3.6.3.1 DTSPMS results

Our branch-and-cut algorithm was also applied to the *DTSPMS* test instances in [132], using a CPU time limit of 3 hours. Table 3.IV summarizes the results, where each row refers to a set of 20 instances. Here, we show the number of stacks (M), the

Instance	$2n+1$	*M*	*Q*	*UB*	*Optimal*	*Gap*	*Root*	*Time*	*Cuts*	*Path*	*Nodes*
a280	23	2	2	449	449		3.79%	2.0	192	0	42
	27			488	468		3.46%	6.9	168	0	41
	31			613	542		4.59%	105.1	679	0	320
	35			633	624		5.54%	181.3	1 139	0	631
	39			709	669		5.81%	2202.1	3 434	0	1 982
	43			773	n.a.	10.35%	13.98%	3 600	3 702	0	1 923
att532	23	3	3	4 177	4 177		0.93%	0.2	10	0	2
	27			4 937	4 937		1.31%	0.7	7	0	6
	31			5 151	5 151		2.30%	2.9	24	0	17
	35			5 294	5 294		1.51%	2.8	19	0	10
	39			5 587	5 587		2.35%	13.8	30	0	25
	43			9 266	9 266		3.60%	2034.2	3 110	32	4 020
brd14051	23	2	2	4 396	4396		3.69%	8.7	314	0	84
	27			4 439	4 439		4.46%	30.4	642	3	164
	31			4 809	n.a.	3.41%	8.38%	3 600	31 596	0	4 894
	35			4 945	n.a.	4.65%	6.87%	3 600	20 525	0	3 267
	39			6 704	n.a.	4.72%	8.87%	3 600	9 645	0	1 905
	43			6 923	n.a.	4.40%	4.60%	3 600	7 024	0	1 305
d15112	23	3	2	74 603	74 603		6.32%	2.9	98	2	66
	27			80 690	80 690		6.89%	37.0	454	0	265
	31			89 754	89 754		4.82%	20.6	200	8	122
	35			96 804	96 804		6.51%	3 278.0	8 569	66	5 402
	39			103 609	n.a.	5.39%	8.86%	3 600	5 110	1	2 630
	43			109 048	n.a.	9.68%	12.97%	3 600	3 875	0	1 390
d18512	23	2	3	4 280	4 280		1.03%	1.4	46	0	8
	27			4 301	4 301		1.30%	5.7	103	0	32
	31			4 638	4 638		5.84%	2 574.1	15 090	331	7 901
	35			4 741	n.a.	3.56%	6.68%	3 600	9 299	0	4 278
	39			4 917	n.a.	5.78%	8.22%	3 600	5 298	0	3 037
	43			5 100	n.a.	9.02%	11.19%	3 600	3 321	0	1 923
fnl4461	23	4	1	1 889	1 889		0.50%	0.6	54	0	10
	27			2 088	2 088		0.50%	1.6	81	0	8
	31			2 356	2 356		1.78%	16.8	238	0	73
	35			2 517	2 517		3.80%	102.6	560	0	168
	39			2 933	n.a.	3.02%	11.02%	3 600	8 883	0	2 639
	43			3 561	n.a.	3.55%	19.71%	3 600	7 174	0	1 408
nrw1379	23	3	2	2 690	2690		2.16%	0.9	39	0	6
	27			3 061	n.a.	3.68%	10.74%	3 600	21 519	1	10 552
	31			3 117	n.a.	5.63%	10.67%	3 600	18 099	0	7 165
	35			3 197	n.a.	6.21%	10.07%	3 600	11 748	0	4 310
	39			3 476	n.a.	12.11%	14.97%	3 600	7 324	0	2 253
	43			3 799	n.a.	14.69%	16.89%	3 600	5 533	0	1 466
pr1002	23	2	2	13 718	13 718		1.05%	0.4	45	0	7
	27			15 436	15 436		3.34%	5.1	121	0	37
	31			16 268	16 268		5.01%	146.6	1 686	0	813
	35			17 601	17 601		4.30%	384.2	3 143	0	1 484
	39			18 673	18 673		4.33%	1 761.7	9 266	0	4 414
	43			20 199	n.a.	2.31%	5.88%	3 600	4 388	0	2 456
ts225	23	2	2	22 000	22 000		0.00%	1.5	66	3	20
	27			29 395	29 395		0.26%	2.4	51	0	15
	31			32 541	32 541		2.66%	4.0	63	0	10
	35			36 405	36 405		7.78%	44.4	241	1	73
	39			40 395	n.a.	2.18%	10.55%	3 600	5 309	0	3 156
	43			43 056	n.a.	5.65%	13.31%	3 600	4 980	0	2 299

Tableau 3.II – Detailed results for the instances in class *C1*

Instance	$2n+1$	*M*	*Q*	*UB*	*Optimal*	*Gap*	*Root*	*Time*	*Cuts*	*Path*	*Nodes*
a280	23	2	12	455	455		4.89%	8.6	252	5	82
	27			479	479		4.62%	15.3	450	3	139
	31			592	553		5.95%	497.5	3105	19	1131
	35			655	635		6.93%	1082.3	2509	4	1106
	39			717	n.a.	6.94%	11.57%	3600	7622	11	3266
	43			793	n.a.	12.68%	15.98%	3600	4199	0	1753
att532	23	2	15	4190	4190		0.50%	0.3	17	0	3
	27			5033	5033		3.31%	4.7	97	0	44
	31			5665	n.a.	5.23%	11.08%	3600	13263	0	7842
	35			5920	n.a.	6.98%	11.60%	3600	10772	0	5551
	39			6184	n.a.	7.68%	11.26%	3600	5083	0	2560
	43			10025	n.a.	8.12%	9.85%	3600	4311	0	1566
brd14051	23	3	11	4386	4386		3.45%	4.0	66	0	31
	27			4459	4458		4.87%	960.4	5276	1820	6540
	31			4795	n.a.	3.99%	10.07%	3600	29334	0	5802
	35			4891	n.a.	8.21%	11.25%	3600	20218	0	3569
	39			6276	n.a.	2.81%	5.25%	3600	10015	0	2180
	43			6322	n.a.	3.37%	4.95%	3600	6494	0	1846
d15112	23	3	10	73872	73872		5.40%	5.1	98	12	78
	27			81657	81657		8.52%	452.7	2935	996	3734
	31			91799	91799		7.34%	1402.2	4781	1673	7027
	35			97040	n.a.	3.22%	9.21%	3600	8631	93	5481
	39			99729	n.a.	3.48%	7.88%	3600	5160	8	3784
	43			105242	n.a.	8.50%	11.03%	3600	3048	0	1706
d18512	23	2	14	4341	4341		2.40%	70.6	1467	186	1085
	27			4572	n.a.	2.35%	5.97%	3600	28051	3162	17907
	31			4893	n.a.	2.82%	6.91%	3600	13879	0	4795
	35			5099	n.a.	4.15%	5.84%	3600	14801	0	3672
	39			5359	n.a.	6.48%	9.56%	3600	8993	0	2535
	43			5768	n.a.	12.00%	12.69%	3600	5936	0	1520
fnl4461	23	3	10	1883	1883		0.66%	0.5	30	1	6
	27			2088	2088		2.19%	14.7	332	33	189
	31			2262	2262		2.99%	36.8	502	26	224
	35			2428	n.a.	3.09%	7.31%	3600	12539	2	6083
	39			2634	n.a.	8.35%	11.11%	3600	7019	7	3603
	43			2773	n.a.	8.79%	10.94%	3600	2522	0	1919
nrw1379	23	4	10	2690	2690		2.34%	1.3	38	0	9
	27			3055	n.a.	2.82%	10.48%	3600	27110	5	13395
	31			3116	n.a.	5.62%	10.61%	3600	13525	18	6614
	35			3197	n.a.	6.01%	10.14%	3600	11452	0	4628
	39			3422	n.a.	9.47%	12.02%	3600	4320	0	1482
	43			3769	n.a.	11.33%	13.08%	3600	5390	0	1603
pr1002	23	3	13	13527	13527		1.55%	2.0	39	18	68
	27			15221	15221		3.92%	14.2	170	60	271
	31			15676	15676		3.19%	30.9	333	35	301
	35			17009	17009		2.79%	75.7	574	52	427
	39			18136	18136		3.31%	504.8	2085	66	1558
	43			19613	19613		4.24%	1322.2	4018	95	2650
ts225	23	2	12	22000	22000		0.00%	0.9	39	0	9
	27			34000	34000		10.60%	985.3	9191	468	5419
	31			37703	n.a.	3.44%	10.57%	3600	15077	802	8884
	35			41703	n.a.	5.63%	15.23%	3600	10796	392	6549
	39			45703	n.a.	11.29%	20.03%	3600	4652	0	2348
	43			49097	n.a.	14.18%	20.21%	3600	3476	0	1759

Tableau 3.III – Detailed results for the instances in class *C2*

capacity of each stack (*Q*), the number of nodes (*n*), the number of solved instances (*Solved*), the average gap for unsolved instances (*Gap*), the average gap at the root node (*Root*), the average CPU time in seconds for solved instances (*Time*), the number of cuts of type (3.32), (3.34) and (3.39) (*Cuts*), the number of path inequalities (3.18) generated through the packing procedure (*Path*) and the number of nodes generated in the branch-and-bound tree (*Nodes*). The results show that instances with up to 48 nodes can be solved. The number of added inequalities is huge, which is mostly due to the increased computation time and the addition of local cuts. We also observe that the problem is much more difficult to solve with 2 stacks than with 4 stacks. While most of the instances with 32 nodes and 2 stacks could not be solved within 3 hours of computation time, the instances with 4 stacks were solved in a few seconds.

M	*Q*	*n*	*Solved*	*Gap*	*Root*	*Time*	*Cuts*	*Path*	*Nodes*
2	5	20	20		4.86%	4.9	1 438.0	1.3	380.8
2	6	24	20		6.97%	179.1	22 347.8	6.7	5 510.4
2	7	28	16	2.66%	8.57%	1 723.3	223 692.9	4.5	49 677.2
2	8	32	3	5.12%	11.07%	4 424.6	321 952.5	0.9	69 695.1
3	4	24	20		1.93%	3.1	580.0	0.9	99.6
3	5	30	20		3.85%	216.9	20 963.4	10.7	2 550.3
3	6	36	15	1.72%	5.30%	3 385.0	238 683.6	16.3	17 539.2
3	7	42	2	3.40%	7.15%	5 721.7	259 180.1	0.7	13 885.8
4	4	32	20		1.27%	14.0	1 196.1	11.5	179.8
4	5	40	20		2.08%	360.7	22 151.4	26.1	1 668.6
4	6	48	5	1.40%	3.83%	4 282.3	180 588.3	22.1	14 762.8
Total			161						

Tableau 3.IV – Average results for the DTSPMS instances with 3 hours of CPU time

A comparison with the exact approaches for the *DTSPMS* proposed in [28, 111] is reported in Table 3.V. We note that a processor of 1.6 GHz is used in [111] while a 2.33GHz Intel Core2 Q8200 processor is used in [28]. For each class of instances, we report the number of instances solved by each procedure and the average CPU time in seconds. We can observe that our branch-and-cut algorithm can solve a larger number of instances.

In Table 3.VI, our algorithm is compared with the previous branch-and-cut approach of Petersen et al. [132] on their test instances, using one hour of computation time (note that these are the only results reported in [132]). This alternative branch-and-cut algorithm consists in building two optimal routes, that is, one for the pickups and one for the deliveries. Then, the packing problem associated with the two routes is solved. An

			Lusby et al.		*Carrabs et al.*		*Our B&C*	
M	*Q*	*n*	*Solved*	*Time*	*Solved*	*Time*	*Solved*	*Time*
2	5	20	20	9.5	20	2.7	20	4.9
2	6	24	19	926.8	20	104.2	20	179.1
2	7	28	5	1 424.4	16	2423.2	16	1 723.3
2	8	32					3	4 424.6
3	4	24	20	4.0			20	3.1
3	5	30	20	492.2			20	216.9
3	6	36	*5*	n.a.			15	3 385.0
3	7	42					2	5 721.7
4	4	32					20	14.0
4	5	40					20	360.7
4	6	48					5	4 282.3

Tableau 3.V – Comparison with other approaches for the DTSPMS

optimal solution is obtained if a feasible packing is found. Otherwise, parts of the routes that are responsible for the infeasibility are forbidden through additional path inequalities (3.18). For each class of instances, where each class is made of 5 instances, we report the number of instances solved, the average gap for unsolved instances and the average CPU time in seconds for both algorithms. Once again, due to the new types of inequalities that we consider, our algorithm is superior by solving more instances and by exhibiting a smaller gap on the unsolved instances.

			Petersen et al.			*Our B&C*		
M	*Q*	*n*	*Solved*	*Gap*	*Time*	*Solved*	*Gap*	*Time*
2	4	16	5		22.8	5		0.5
2	5	20	5		285.0	5		6.9
2	6	24	1	4.85%	1 680.0	5		336.7
2	7	28	0	5.76%		3	2.34%	714.7
3	4	24	5		25.0	5		2.0
3	5	30	4	2.39%	1 737.8	5		114.2
3	6	36	1	5.79%	1 995.0	3	1.41%	1 235.8
3	7	42	0	9.45%		1	4.69%	2 460.1
4	4	32	5		33.8	5		20.0
4	5	40	4	5.57%	1 876.0	5		226.4
4	6	48	0	8.72%		1	1.49%	1 809.5
4	7	56	0	12.03%		0	3.31%	
Total			30			43		

Tableau 3.VI – Comparison between our branch-and-cut and the one of Petersen et al. [132]

In Petersen et al. [132], the authors also report *DTSPMS* results on modified *PDTSPL* instances. In these instances, the vehicle must pickup all items, return to the depot and then perform the deliveries. All instances have 3 stacks and the computation time is set to one hour. Results are presented in Tables 3.VII and 3.VIII. For the algorithm of Petersen et al., these tables show the final gap for unsolved instances, based on the best known solutions, and the CPU time in seconds, as reported in [132]. For our algorithm, the tables report the heuristic upper bound (*UP*), the lower bound (*LB*), which is the value of the optimal solution when the instance is solved, the final gap for unsolved instances (*Gap*), the gap at the root node (*Root*) and the CPU time in seconds (*Time*). Here, we were able to solve 61 instances out of 81, as compared with 46 instances for the algorithm of Petersen et al.

3.7 Conclusion

This paper has described the first exact algorithm for solving the pickup and delivery traveling salesman problem with multiple stacks. This branch-and-cut algorithm uses different sets of valid inequalities. Some of these inequalities have been inherited from previous work on pickup and delivery problems, while other new valid inequalities have been derived. The computational results show that we can find the optimum on instances with up to 43 nodes. The comparison with previously reported algorithms for the double TSP with multiple stacks, which is a special case of our problem, also demonstrates the effectiveness of our algorithm.

Instance	*Q*	*n*	*Petersen et al.*		*Our B&C*				
			Gap	*Time*	*UB*	*LB*	*Gap*	*Root*	*Time*
a280	19	3		2	585	585		0,51%	0,3
	23	4		51	654	654		0,76%	1,5
	27	5		19	696	696		0,72%	0,9
	31	5		31	792	792		1,01%	2,8
	35	6		2277	945	945		1,06%	12,9
	39	7	0,68%	3600	1024	1024		0,98%	32,3
	43	7	1,09%	3600	1103	1103		6,75%	99,6
	47	8	1,57%	3600	1179	1179		6,36%	539,6
	51	9	2,21%	3600	1219	1212	0,57%	2,38%	3600
att532	19	3		2	5361	5361		0,02%	0,1
	23	4		23	6399	6399		0,11%	0,3
	27	5		102	7261	7261		0,36%	0,5
	31	5		320	7562	7562		1,28%	3,9
	35	6	1,60%	3600	7863	7863		1,93%	39,0
	39	7	2,88%	3600	8208	8208		3,11%	949,1
	43	7	3,24%	3600	12639	12638		3,42%	1087,8
	47	8	3,66%	3600	13006	12920,5	0,66%	3,81%	3600
	51	9	3,11%	3600	16214	16042	1,06%	3,23%	3600
brd14051	19	3		0	7897	7897		0,91%	0,0
	23	4		1	8064	8064		0,00%	0,1
	27	5		41	8079	8079		0,04%	0,8
	31	5		3	8196	8196		0,00%	0,5
	35	6	0,32%	3600	8252	8252		0,36%	589,2
	39	7	0,30%	3600	8419	8419		0,36%	213,2
	43	7	0,50%	3600	8442	8442		0,53%	2150,7
	47	8	0,71%	3600	8560	8527	0,39%	0,74%	3600
	51	9	1,52%	3600	8644	8553,5	1,05%	1,54%	3600
d15112	19	3		28	93597	93597		1,49%	0,3
	23	4		39	100489	100489		1,54%	0,9
	27	5		211	108574	108574		1,96%	4,8
	31	5	2,44%	3600	127814	127806		4,03%	124,5
	35	6	3,65%	3600	131421	131408		4,50%	1683,2
	39	7	4,64%	3600	136488	133683	2,06%	5,25%	3600
	43	7	5,67%	3600	139965	135082,67	3,49%	5,91%	3600
	47	8	5,63%	3600	141404	136275	3,63%	5,88%	3600
	51	9	7,27%	3600	149772	140940,31	5,90%	7,45%	3600
d18512	19	3		1	7951	7951		0,00%	0,1
	23	4		1	8023	8023		0,00%	0,2
	27	5		6	8034	8034		0,00%	0,5
	31	5		19	8098	8098		0,00%	1,2
	35	6	0,33%	3600	8151	8151		0,36%	244,9
	39	7	0,42%	3600	8327	8327		0,44%	1493,6
	43	7	0,67%	3600	8482	8456	0,31%	0,70%	3600
	47	8	0,92%	3600	8555	8499	0,65%	0,95%	3600
	51	9	1,34%	3600	8672	8577,67	1,09%	1,36%	3600

Tableau 3.VII – Comparison between our branch-and-cut and the one of Petersen et al. [132]

			Petersen et al.		*Our B&C*				
Instance	*Q*	*n*	*Gap*	*Time*	*UB*	*LB*	*Gap*	*Root*	*Time*
fnl4461	19	3		1	3387	3387		0,09%	0,2
	23	4		9	3430	3430		0,12%	0,5
	27	5		185	3628	3628		0,41%	17,8
	31	5		192	3796	3796		0,37%	17,3
	35	6	0,42%	3600	3853	3853		0,88%	401,1
	39	7	1,14%	3600	4027	4016	0,27%	1,32%	3600
	43	7	2,15%	3600	4147	4101,5	1,10%	2,39%	3600
	47	8	3,36%	3600	4315	4201,57	2,63%	3,59%	3600
	51	9	3,93%	3600	4427	4280,38	3,31%	4,11%	3600
nrw1379	19	3		3	4572	4572		0,00%	0,2
	23	4		17	4733	4733		0,11%	0,7
	27	5		273	4872	4872		0,29%	12,3
	31	5		1230	4984	4984		0,34%	14,3
	35	6	0,33%	3600	5212	5212		0,54%	34,3
	39	7	1,41%	3600	5320	5284,13	0,67%	1,60%	3600
	43	7	1,97%	3600	5543	5476,67	1,20%	2,13%	3600
	47	8	1,98%	3600	5592	5513,5	1,40%	2,11%	3600
	51	9	3,20%	n,a,	6056	5879,88	2,91%	3,27%	3600
pr1002	19	3		0	21498	21498		0,14%	0,1
	23	4		15	22977	22977		0,38%	0,3
	27	5		184	25087	25087		0,87%	2,8
	31	5		929	25899	25899		1,39%	12,1
	35	6		731	27246	27245		2,22%	10,5
	39	7		1733	28196	28196		1,32%	20,6
	43	7		5	29875	29875		0,87%	1,7
	47	8		133	31463	31463		0,30%	7,9
	51	9		5	32319	32319		0,05%	6,4
ts225	19	3		0	34000	34000		0,00%	0,1
	23	4		443	43000	43000		0,00%	0,6
	27	5		2	48440	48440		0,00%	1,0
	31	5		4	50580	50580		0,05%	1,1
	35	6		2	50881	50881		0,07%	1,7
	39	7		17	51371	51371		0,11%	2,5
	43	7		8	52322	52322		0,21%	3,1
	47	8		6	54460	54460		0,46%	4,6
	51	9		808	62688	62688		1,60%	44,5
Average			2,82%	219,8			1,72%		162,3

Tableau 3.VIII – Comparison between our branch-and-cut and the one of Petersen et al. [132] (continued)

CHAPITRE 4

AN EXACT ALGORITHM FOR THE TWO-DIMENSIONAL ORTHOGONAL PACKING PROBLEM WITH UNLOADING CONSTRAINTS

Ce chapitre traite d'une méthode de séparation et évaluation avec plans sécants ("branch-and-cut") pour la résolution d'un problème de chargement d'items en deux dimensions de forme rectangulaire. On le retrouve souvent comme sous-problème dans les problèmes de tournées de véhicules avec chargement d'items en deux dimensions. La réalisabilité des routes dépend en effet de l'existence ou non d'un chargement réalisable des items à l'intérieur des véhicules.

Ce travail se concentre sur le développement d'une méthode de résolution efficace qui tient compte des particularités de ce problème. Le but ultime étant d'intégrer cet algorithme au sein d'une méthode de résolution exacte pour le problème de tournées de véhicules avec items en deux dimensions.

L'article a été soumis à la revue *Operations Research* en avril 2013 et les demandes de corrections à apporter par les arbitres ont été reçues en septembre 2013.

An Exact Algorithm for the Two-Dimensional Orthogonal Packing Problem with Unloading Constraints

Jean-François Côté †§ Michel Gendreau ‡§, Jean-Yves Potvin †§

†Département d'informatique et de recherche opérationnelle, Université de Montréal, C.P. 6128, succ. Centre-Ville, Montréal, Québec, Canada H3C 3J7.

‡Département de mathématiques et de génie industriel, École Polytechnique de Montréal, C.P. 6079, succ. Centre-Ville, Montréal, Québec, Canada H3C 3A7.

§CIRRELT, Université de Montréal,
C.P. 6128, succ. Centre-Ville, Montréal, Québec, Canada H3C 3J7.

Abstract

This paper describes a branch-and-cut algorithm for solving a two-dimensional orthogonal packing problem with unloading constraints, which often occurs as a subproblem of mixed vehicle routing and loading problems. At each node of the branching tree, a two-phase approach is applied to find a feasible packing of the items. Different techniques to reduce the size of the solution space and uncover infeasibility are also described. A numerical comparison with the best known exact method is reported at the end on benchmark instances.

Keywords : Vehicle routing, unloading constraints, packing, branch-and-cut.

4.1 Introduction

Typically, when solving a mixed vehicle routing and loading problem, a delivery route is first generated and then a feasibility check is performed to determine if the goods can be feasibly packed inside the vehicle. This feasibility problem, in two dimensions (2D) or three dimensions (3D), is far from obvious and the work reported here focuses on this issue.

Let us consider the delivery route $\mathscr{R}$ of two-dimensional rectangular objects (called items, thereafter), which is defined as a sequence of cardinality R over a set of customers $C = \{1, 2, ..., R\}$. Without loss of generality, we will assume in the following that customer j is the j-th customer in the sequence, that is, customer 1 is delivered first, customer 2 is delivered second, etc. Let us also consider the set of items $I = \{1, ..., n\}$ to be delivered to the customers, where each item $i \in I$ is characterized by its width w_i, its height h_i, as well as its associated customer and delivery order $seq_i \in \{1, 2, ..., R\}$ in route $\mathscr{R}$. Note that all items delivered to a given customer have the same delivery order. The loading area of the vehicle (called bin, thereafter) has width W and height H. Given these assumptions, we want to know if the items can fit inside the bin, that is, without overlap and in such a way that the items of any given customer are directly available at delivery time (i.e., the items can be moved out by directly pulling each one of them outside of the bin without moving any other item). The latter characteristic will be referred to as the *unloading constraints*. It should be noted that the *unloading constraints* are also referred to as *sequential loading*, *rear loading*, *multi-drop* or *LIFO* constraints in the literature. We prefer *unloading constraints* because these constraints relate to the deliveries.

Figure 5.1 presents an example for a route that starts from the depot 0 and then visits the customers 1, 2, 3, 4 and 5. Figures 5.1(a) and 5.1(b) show two possible packings for this route : the first one satisfies the unloading constraints while the second one does not (note that the items are taken out from the top of the bin, which corresponds to the rear of the vehicle). In the second case, the items of customers 2 must clearly be moved to allow the items of customer 1 to be unloaded.

This problem is the classical *2D Orthogonal Packing Problem* (2OPP) with the ad-

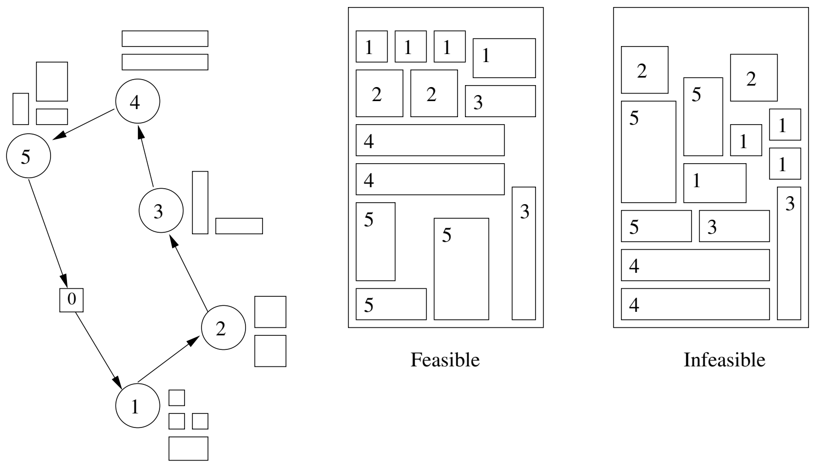

Figure 4.1 – Packing examples

dition of unloading constraints (UL). The 2OPP is often found as a subproblem of the *2D Strip Packing Problem* (2SPP) and the *2D Bin Packing Problem* (2BPP). Recent exact methods for the 2OPP can be found in [25, 38, 67, 122], while exact methods for the 2SPP are found in [4, 6, 22, 51, 116]. A recent work on the 2SPP with unloading constraints is also reported in [147] where a GRASP heuristic, previously proposed in [3], and two approximation algorithms are used to solve the problem.

The 2OPP-UL occurs in several papers as a subproblem of mixed vehicle routing and loading problems, when unloading constraints apply as well. In these problems, a fleet of vehicles must visit a set of customers while minimizing the total traveled distance. In addition, the items in the route of each vehicle must be feasibly packed. The literature on this problem distinguishes four main variants, depending if the items can be rotated by 90 degrees (*non-oriented*) or not (*oriented*) and if unloading constraints apply or not :

- $2|OU|$: two-dimensional oriented with unloading constraints ;
- $2|OR|$: two-dimensional oriented without unloading constraints ;
- $2|NU|$: two-dimensional non-oriented with unloading constraints ;
- $2|NR|$: two-dimensional non-oriented without unloading constraints.

Our work focuses on the $2|OU|$ variant. The $2|OR|$ and $2|NR|$ variants are related to the 2OPP and a vast literature can be found on the subject. We believe that the work reported here can also be generalized to the non-oriented variant $2|NU|$.

The heuristics reported in the literature for mixed vehicle routing and loading problems are often local search heuristics which are designed to improve the current solution by applying some modification to it (see, for example, [62, 71, 76, 105, 165]). A modification to a vehicle route is accepted only if all constraints related to the packing are satisfied. This is typically verified with simple heuristics like the Bottom-Left, Bottom-Left Fill and Touching Perimeter heuristics [165].

Only a few exact algorithms are reported in the literature for these problems due, in particular, to the inherent difficulty of the packing. In [89], a branch-and-cut algorithm is proposed where the classical Bottom-Left heuristic is first used and, if it fails, an exact branch-and-bound algorithm based on the work in [116] is applied. More recently, the authors in [46] describe a branch-and-cut-and-price algorithm where a similar, but more involved, tree-based search is proposed. A lower bound that takes into account the unloading constraints is also reported. The reader is referred to the exhaustive surveys in [19, 88] for the three-dimensional case.

When considering the 2OPP, a relaxation known as the *One-Dimensional Contiguous Bin Packing Problem* (1CBP) can be used to generate an initial lower bound on the required height of the bin. This relaxation is defined as follows :

Definition 4.1.1. *Given that each item i of width w_i is cut into h_i slices of unitary height, we ask for the packing of all slices into a minimum number of bins of capacity W, in such a way that if the first slice of item i is packed into bin j, then the kth slice is packed into bin $j+k$, $k \in \{1,2,...,h_i\}$.*

The number of bins obtained is a lower bound on the required height H. That is, if the resulting lower bound is larger than H then the problem is infeasible, otherwise the problem remains undecided. Note that if we have a feasible solution for the 2OPP, a feasible solution of the 1CBP can be obtained by considering the y coordinate of the bottom-left corner of each item as the height position of its starting slice in the 1CBP.

Conversely, one could try to build a feasible solution for the 2OPP from a 1CBP solution by using the height position of the first slice of every item as the set of y-coordinates and then by determining a set of x-coordinates that does not lead to any overlap. This kind of two-phase approach can be found in [51], although the authors go the other way around by first solving for the x-coordinates and then, for the y-coordinates. Intuitively, this approach looks good in our case, given that the width W (x-axis) of a vehicle's loading area is typically smaller than its height H (y-axis). It means that the first problem is smaller, while the second problem exhibits a larger degree of freedom when suitable y-coordinates must be found. Unfortunately, a preliminary computational study has shown that the structure of the x-coordinates obtained by solving the first problem seems to preclude, in most cases, the identification of y-coordinates that satisfy the unloading constraints.

Given the above observations, we propose a branch-and-cut algorithm to solve the 2OPP-UL, where the 1CBP is first solved to get y-coordinates, followed by a so-called x-check problem to get the corresponding x-coordinates. Several preprocessing routines and inequalities that take into account the unloading constraints are also proposed to tighten the problem.

The remainder of the paper is organized as follows. A basic mathematical model is first introduced in Section 4.2. Then, the outline of the problem-solving methodology is explained in Section 4.3. Various inequalities for LP relaxations of the 1CBP are reported in Section 4.4, including a description of the separation routines associated with these inequalities. Section 4.5 describes the various preprocessing routines, while the lower bounds are presented in Section 4.6. Computational results are finally reported in Section 4.7.

4.2 Model

In the following, the mathematical notation is first presented. Then, the notion of normal patterns is introduced to reduce, without impairing optimality, the set of solutions to be examined. After these preliminaries, an integer programming model for the 2OPP-

UL is reported.

4.2.1 Notation

The following notation will be used throughout the paper :

- I : set of items to deliver,
- I_i : set of items with delivery order i (same customer),
- $I_i^=$: set of items delivered at the same time than item i,
- $I_i^<$: set of items delivered before item i,
- $I_i^{<,w}$: set of items delivered before item i of width $> w$,
- $I_i^>$: set of items delivered after item i,
- $I_i^{>,w}$: set of items delivered after item i of width $> w$,
- $\Sigma_{(i)}^> = \sum_{\substack{k \in I \setminus \{i\} \\ seq_k > seq_i}} w_k h_k$: total area of the items delivered after item i,
- $\Sigma_{(i)}^{>,w} = \sum_{\substack{k \in I \setminus \{i\} \\ seq_k > seq_i \\ w_k > w}} w_k h_k$: total area of the items of width $> w$ delivered after item i ,
- $\Sigma_{(i,j)}^< = \sum_{\substack{k \in I \setminus \{i,j\} \\ seq_i < seq_k < seq_j}} w_k h_k$: total area of the items delivered after item i but before item j,
- $\Sigma_{(i,j)}^{<,w} = \sum_{\substack{j \in I \setminus \{i,j\} \\ seq_i < seq_k < seq_j \\ w_k > w}} w_k h_k$: total area of the items of width $> w$ delivered after item i but before item j.

This notation is extended in the following to include items that are delivered at the same time than item i (i.e., the $<$ and $>$ signs are replaced by $\leq$ and $\geq$ signs, respectively).

4.2.2 Normal Patterns

In general, the bottom-left corner of a given item $i \in I$ can be found at every (x, y) coordinate where $x \in [0, W - w_i]$ is the width position and $y \in [0, H - h_i]$ is the height position. This set of coordinates can however be reduced by considering only normal patterns [84], which are defined as follow :

$$P_i^W = \{x = \sum_{j \in I \setminus \{i\}} w_j \xi_j : 0 \leq x \leq W - w_i, \xi_j \in \{0,1\}, j \in I \setminus \{i\}\} \tag{4.1}$$

$$P_i^H = \{y = \sum_{j \in I \setminus \{i\}} h_j \xi_j : 0 \leq y \leq H - h_i, \xi_j \in \{0,1\}, j \in I \setminus \{i\}\} \tag{4.2}$$

When the unloading constraints are considered, the set P_i^H can be reduced by observing that items to be delivered after item i cannot be over it :

$$P_i^H = \{y = \sum_{j \in I_i^{\geq} \setminus \{i\}} h_j \xi_j : 0 \leq y \leq H - h_i, \xi_j \in \{0,1\}, j \in I_i^{\geq} \setminus \{i\}\} \tag{4.3}$$

We define $P^W = \bigcup_{i \in I} P_i^W$ and $P^H = \bigcup_{i \in I} P_i^H$ as the set of all possible width and height positions. We also define the set $P_i^W(r)$ as the set of positions that cover width position r. The set $P_i^H(t)$ is defined similarly for height position t :

$$P_i^W(r) = \{x \in P_i^W : [r - w_i + 1]^+ \leq x \leq r\},\ i \in I, r \in P^W \tag{4.4}$$

$$P_i^H(t) = \{y \in P_i^H : [t - h_i + 1]^+ \leq y \leq t\},\ i \in I, t \in P^H$$

where $[v]^+ = \max\{0, v\}$.

4.2.3 Mathematical model

Given the decision variables

- x_{ir} is 1 if the bottom-left corner of item i is located at width position r, 0 otherwise, $i \in I$, $r \in P_i^W$;

- y_{it} is 1 if the bottom-left corner of item i is located at height position t, 0 otherwise, $i \in I, t \in P_i^H$,

we formulate the *2OPP-UL* as follows :

$$\text{(2OPP-UL)} \qquad \sum_{r \in P_i^W} x_{ir} = 1 \qquad i \in I \tag{4.5}$$

$$\sum_{t \in P_i^H} y_{it} = 1 \qquad i \in I \tag{4.6}$$

$$\sum_{r \in P_i^W(\bar{r})} x_{ir} + \sum_{r \in P_j^W(\bar{r})} x_{jr} + \sum_{t \in P_i^H(\bar{t})} y_{it} + \sum_{t \in P_j^H(\bar{t})} y_{jt} \leq 3 \qquad i \in I, j \in I_i^{=}, \bar{r} \in P^W, \bar{t} \in P^H \tag{4.7}$$

$$y_{i\bar{t}} + \sum_{r \in P_i^W(\bar{r})} x_{ir} + \sum_{r \in P_j^W(\bar{r})} x_{jr} + \sum_{\substack{t \in P_j^H(\bar{t}) \\ t \leq \bar{t} + h_i - 1}} y_{jt} \leq 3 \qquad i \in I, j \in I_i^{<}, \bar{r} \in P^W, \bar{t} \in P_i^H \tag{4.8}$$

$$x_{ir} \in \{0,1\} \qquad r \in P_i^W, i \in I \tag{4.9}$$

$$y_{it} \in \{0,1\} \qquad t \in P_i^H, i \in I \tag{4.10}$$

Constraints (4.5) and (4.6) impose a (*x*,*y*) coordinate for each item. Constraints (4.7) ensure that no pair of items covers the same (*x*,*y*) coordinate. In particular, if items i and j overlap, it is possible to find a coordinate (*x*,*y*) such that the left hand side of constraint (4.7) equals 4. The unloading constraints are imposed through (4.8). If item i is to be delivered after j and both items cover the same x coordinate (i.e., the summation of the two terms involving the x-coordinates equals 2), then item j cannot be below i (i.e., the summation of the two terms involving the y-coordinates must be smaller than or equal to 1). The number of constraints of types (4.7) and (4.8) is *pseudo*-polynomial and grows very quickly with W and H, thus leading to a model which is not really useful in practice. Accordingly, the next section will describe an alternative approach to tackle this problem.

4.3 Problem-solving methodology

In the following, we first describe preprocessing techniques aimed at reducing the number of solutions to be examined. Then, instead of solving directly the 2OPP-UL model, a two-phase approach similar to the one reported in [51] for the 2SPP is applied. Basically, at each node of our branch-and-cut algorithm, a modified 1CBP (formulated as a *binary program*) is solved to find y coordinates. Then, the corresponding x-check problem is addressed in the second phase.

4.3.1 Preprocessing

Some preprocessing is first applied to tighten the problem. In this process, infeasibility might be uncovered. The algorithmic flow is the following :

1. Apply height position restrictions
2. Apply lifting
3. Calculate lower bounds SPP, L_2, L_3^H, L_3^W
4. From bottom to top-fill do
    (a) Apply precedence relations
    (b) Apply normal pattern dominance
    (c) Apply normal pattern removal
5. Select best fill
6. Calculate L_4^H, L_4^W

In step 1, the minimum and maximum possible height positions for each item are determined (section 4.5.1). This is followed by lifting procedures aimed at artificially increasing the item sizes (section 4.5.2). Then, some lower bounds are calculated (section 4.6). Different ways of filling the bin, either from the bottom, from the top, or from the bottom and top are considered (section 4.5.5), while applying normal pattern dominance and normal pattern removal to reduce as much as possible the set of normal patterns (section 4.5.4). The filling associated with the smallest set of normal patterns is then selected. At the end, two sophisticated bounds, which exploit the current set of normal patterns, are applied (section 4.6) .

At each step, infeasibility tests are performed to detect if the current set of normal patterns for any given item becomes empty or if the minimum and maximum height positions of any given item are not coherent with the dimensions of the bin (e.g., if the minimum height position of item i must be larger than $H - h_i$).

4.3.2 Modified 1CBP

The modified 1CBP has the same variables than the 2OPP-UL and is defined as follow :

$$
\begin{aligned}
&(\mathit{1CBP}) \qquad & \sum_{t \in P_i^H} y_{it} = 1 & \qquad & i \in I & \qquad (4.11)\\
&& \sum_{i \in I} \sum_{\bar{t} \in P_i^H(t)} w_i y_{i\bar{t}} \leq W & & t \in P^H & \qquad (4.12)\\
&& \sum_{y_{it} \in S} y_{it} \leq |S| - 1 & & S \in \mathscr{S} & \qquad (4.13)\\
&& y_{it} \in \{0,1\} & & i \in I,\ t \in P_i^H & \qquad (4.14)
\end{aligned}
$$

Basically, this is a standard 1CBP but with the number of bins fixed at H. Constraints (4.11) state that the first slice of each item must be assigned to a bin. Constraints (4.12) ensure that the capacity W of each bin is satisfied. Finally, constraints (4.13) correspond to *Benders' feasibility cuts*. The set $\mathscr{S}$ contains all subsets $S = \{y_{it}\} \in \mathscr{S}$ such that an infeasibility occurs in the x-check problem when all variables in S are equal to 1. Given that set $\mathscr{S}$ can be very large, these constraints are relaxed in the initial formulation of the problem.

Within the branch-and-cut algorithm, the x-check problem is called each time a binary solution to the current 1CBP linear relaxation is obtained. If the x-check problem is feasible, then we have a feasible solution to the 2OPP-UL. Otherwise, a Benders' cut of type (4.13) is added to the 1CBP model. Such an inequality is very weak because it typically removes a single (x, y) coordinate from the solution space. However, stronger cuts can be generated by applying an idea similar to the one in [51]. Let S be the subset of variables equal to 1 in a solution of the 1CBP for which the corresponding x-check problem is infeasible. Let $S' \subset S$ be such that the corresponding x-check problem is also infeasible. Then, any modification to the y coordinates of the items in $S \backslash S'$ induces a cut that subsumes the cut generated by S. Based on this observation, the set S is reduced as much as possible to obtain a minimal infeasible set (MIS) [39].

To find a MIS, the variables in $S = \{y_{it}\}$ are considered sequentially from the one with the smallest height position to the one with the largest height position. We remove the chosen variable from S and solve the x-check problem for the reduced set. If it is feasible, the variable is put back in S, otherwise a smaller set that is still infeasible is found. We repeat the procedure until all variables have been considered. Although the order of removal of the variables from set S can lead to different MISs, a single order was considered here (no significant improvement was observed in preliminary tests using multiple MISs). Note also that the resulting inequality cannot be lifted as in [51], because the infeasibility of the x-check problem can come from the unloading constraints.

Apart from the Bender's cuts, additional cuts are generated when the solution of the current 1CBP linear relaxation is fractional. These inequalities are described in Section 4.4.

4.3.3 x-check problem

The model for the x-check problem is obtained by replacing the y_{it} variables in the 2OPP-UL model by their optimal values in the 1CBP problem. Unfortunately, the resulting model is still too large to be solved in practice with CPLEX. The x-check problem is thus addressed with an enumerative tree search-based approach, which proves to be very efficient. Let $\bar{y}_i$ be the y-coordinate of item i in the 1CBP solution. The algorithm starts by filling the bottom of the bin and then moves up progressively. At the root node, no item is in the bin. Then, for each item i such that $\bar{y}_i = 0$ a child node is created with i

at coordinate $(0,0)$ in the bin. For subsequent nodes, let (x_{cur}, y_{cur}) be the current lowest leftmost position. For each item i such that $\bar{y}_i = y_{cur}$, a child node is created with item i at (x_{cur}, y_{cur}) if the item fits without overlap and if the unloading constraints are satisfied. An empty node is also created where coordinates (x_{cur}, y_{cur}) to $(x_{cur} + e_x, y_{cur} + e_y)$ are forbidden with $e_x = \min_{i \in I'}\{\min_{r \in P_i^W}\{r : r > x_{cur}\}\}$, $e_y = \min_{i \in I'}\{\bar{y}_i : \bar{y}_i > y_{cur}\}$ and I' the set of items that are not yet in the bin.

A node is fathomed when the needed empty space is larger than the available empty space. This simple scheme proved the feasibility or infeasibility of every x-check problem after generating only a few nodes in the branching tree.

4.4 Inequalities

Before describing our inequalities, we need to introduce the concept of *conflicting items* or, equivalently, *conflicting variables*. Let Y be the set of variables in our 1CBP. Let also y_{it_i} and y_{jt_j} be two variables in Y. Then, these variables are *conflicting* if they cannot cover a common x-coordinate without overlapping or violating the unloading constraints. Figure 4.2 b) shows an example where the variables associated with items 3 and 4 are conflicting. Clearly, at the time of delivery, item 4 needs to be moved to reach item 3, thus violating the unloading constraints. Two variables y_{it_i} and y_{jt_j} are conflicting if they satisfy one of these two conditions :

- $seq_i < seq_j$ and $t_i < t_j + h_j$,
- $seq_i = seq_j$ and $t_i < t_j + h_j$ and $t_j < t_i + h_i$,

Two variables y_{it} are also considered to be conflicting if they are associated with the same item $i \in I$. The reason is that only one of these variables should be equal to 1, see equation (4.11).

A subset $C \subseteq Y$ is *mutually conflicting* if for every pair of variables $y_{it_i}, y_{jt_j} \in C$, y_{it_i} is conflicting with y_{jt_j}. For example, the set $C = \{y_{1t_1}, y_{2t_2}, y_{3t_3}, y_{4t_4}\}$ is mutually conflicting in both Figures 4.2 a) and b). In the case of Figure 4.2 b), the solution is not UL-feasible because the summation over the widths of the items in set C exceeds W. Conversely, the same summation is equal to W in Figure 4.2 a), which is a UL-feasible solution. This observation can be generalized through the following :

Proposition 4.4.1. *In any feasible 2OPP-UL solution, the summation over the widths of the items in every mutually conflicting set is smaller than or equal to W.*

Proof. By contradiction, let us assume that we have a set $C = \{y_{i_1t_1}, y_{i_2t_2}, ..., y_{i_kt_k}\}$ of mutually conflicting variables in a feasible *2OPP-UL* solution such that $seq_{i_1} \geq seq_{i_2} \geq ... \geq seq_{i_k}$ and the summation over the widths of the corresponding items is strictly larger than W (for the sake of simplicity, it is assumed that the i_j's are all different). Since variable $y_{i_1t_1}$ is conflicting with $y_{i_2t_2}$, the required width for the two corresponding items is $w_{i_1} + w_{i_2}$. The same applies to $y_{i_3t_3}$ which is conflicting with $y_{i_1t_1}$ and $y_{i_2t_2}$, thus requiring a width of $w_{i_1} + w_{i_2} + w_{i_3}$. Following the same line of reasoning, the required width is $\sum_{j=1}^{k} w_{i_j}$ which is strictly larger than W (by hypothesis). This observation contradicts the fact that the 2OPP-UL solution is feasible. This proof can be easily extended to the case where the i_j's are not all different.

These observations led to the conflict inequalities which are descried in the next section.

4.4.1 Conflict inequalities

Let us consider the following proposition :

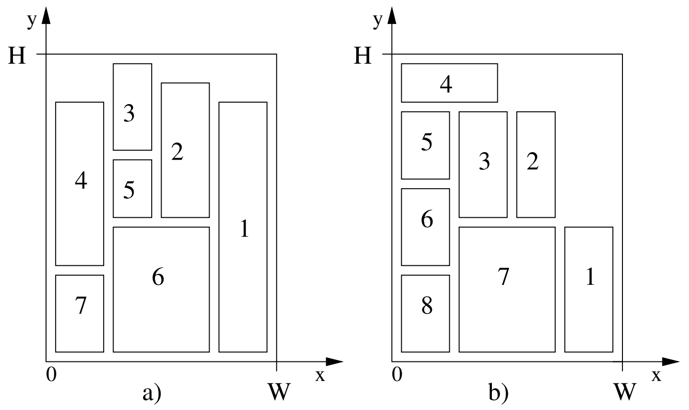

Figure 4.2 – Mutually conflicting sets

Proposition 4.4.2. *In every feasible 2OPP-UL solution, we have :*

$$\sum_{y_{it} \in C} w_i y_{it} \leq W \qquad \forall C \in \mathscr{Y} \text{ such that } \sum_{i \in I(C)} w_i > W, \tag{4.15}$$

where $\mathscr{Y}$ is the set of all mutually conflicting subsets of Y. Inequalities (4.15) are classical *knapsack constraints*. For any given mutually conflicting set C, we can instead use the following *cover inequality* :

$$\sum_{y_{it} \in C} y_{it} \leq |I(C)| - 1 \tag{4.16}$$

where $I(C)$ is the set of items associated with the variables in C. One can observe that inequality (4.16) is a special case of the classical *GUB cover inequality*. We recall that C is a GUB cover if $\sum_{i \in I(C)} w_i > W$ and if no two variables are associated with the same item. Set C can be transformed into a GUB cover by removing all variables but one that belong to the same item. In addition, set C is a minimal GUB cover if no proper subset of C is a GUB cover. Assuming that C is both mutually conflicting and a minimal GUB cover, we can consider an extension E such that $C \subseteq E$, E is mutually conflicting and $\max_{i \in I(C)}\{w_i\} \leq \min_{i \in I(E) \setminus I(C)}\{w_i\}$. Inequality (4.16) can then be extended and dominated by :

$$\sum_{y_{it} \in E} y_{it} \leq |I(C)| - 1 \tag{4.17}$$

We prefer here to use a special type of *GUB lifted cover inequality* because a single weight w_i is associated with the y_{it} variables, for any given item i. Let C be a minimal GUB cover and E be a mutually conflicting set such that $C \subseteq E$. Then we can derive the following inequality :

$$\sum_{y_{it} \in E, i \in I(C)} y_{it} + \sum_{y_{it} \in E, i \notin I(C)} \alpha_i y_{it} \leq |I(C)| - 1 \tag{4.18}$$

We will refer to the α_i's as the *lifting coefficients*. They can be computed by solving a series of knapsack problems [82]. We will refer to these inequalities as *conflict* *inequalities*.

4.4.2 Max flow inequalities

Another type of inequality is obtained by considering any mutually conflicting set C such that $\sum_{i \in I(C)} w_i \leq W$. For example, let us assume that items 1, 5 and 6 in Figure 4.3 are fixed at their current position. Then, we can identify the feasible areas for all other items. For example, the items to be delivered after 1 but before 5 must be in areas A_1, B_5, B_6 or E. Let $t_i^{min} = \min_{y_{jt} \in C}\{t : j = i\}$ and $t_i^{max} = \max_{y_{jt} \in C}\{t : j = i\}$. A maximum flow problem, defined on an appropriate network, can be solved to know if the remaining free items can lead to a feasible solution given that the items associated with set C are fixed at their current position. More precisely, the network would be as follows :

- a source node s,
- a destination node t,
- for each $i \in I(C)$, nodes A_i and B_i, where A_i stands for the area immediately under item i and B_i for the area immediately above it,
- a node E for the remaining area,
- for each $i \in I \backslash I(C)$, a node k_i,

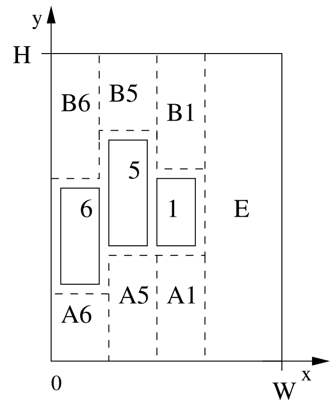

Figure 4.3 – Areas associated with mutually conflicting sets

- for each $i \in I(C)$, an arc from s to A_i with a capacity equal to $t_i^{max} w_i$,
- for each $i \in I(C)$, an arc from s to B_i with a capacity equal to $(H - t_i^{min} - h_i)w_i$,
- an arc from s to E with a capacity equal to $H(W - \sum_{i \in I(C)} w_i)$,
- for each $i \in I(C)$ and $j \in I \backslash I(C)$, an arc from A_i to k_j with a capacity equal to $\min\{w_i, w_j\}h_j$ if $h_j \leq t_i^{max}$ and $seq_i \leq seq_j$,
- for each $i \in I(C)$ and $j \in I \backslash I(C)$, an arc from B_i to k_j with a capacity equal to $\min\{w_i, w_j\}h_j$ if $h_j \leq H - t_i^{min} - h_i$ and $seq_i \geq seq_j$,
- for each $i \in I \backslash I(C)$, an arc from E to k_i with a capacity equal to $min\{w_i, W - \sum_{i \in I(C)} w_i\}h_i$
- for each $i \in I \backslash I(C)$, an arc from k_i to t with a capacity equal to $h_i w_i$.

Let $maxflow(C)$ be the maximum flow from s to t in this network. We then have :

Proposition 4.4.3. *If $maxflow(C)$ is strictly smaller than $\sum_{i \in I \backslash I(C)} h_i w_i$ then the mutually conflicting set C leads to an infeasible solution. The partial solution can be forbidden through the following constraints :*

$$\sum_{y_{it} \in C} y_{it} \leq |I(C)| - 1 \qquad \forall C \in \mathscr{Y} \text{ such that } maxflow(C) < \sum_{i \in I \backslash I(C)} h_i w_i \tag{4.19}$$

We will refer to these inequalities as the *max flow inequalities*.

4.4.3 Separation procedures

In the previous section, we introduced some properties of feasible *2OPP-UL* solutions. In many cases, the 1CBP solution given to the x-check problem will not satisfy these properties. It is thus useful to look for a violated property before calling the x-check problem. The inequality generated from a violated property will also be stronger as it will focus on what truly makes the solution infeasible. Accordingly, at each node of our branch-and-cut algorithm, we first check if the current solution violates inequalities (4.15) or (4.19).

A simple heuristic and an exact method have been developed to separate violated inequalities. Given that these inequalities are based on mutually conflicting sets, the first algorithm generates conflicting sets in a heuristic way, while the second one generates all of them.

Starting with an empty set, the heuristic first adds a randomly chosen variable to this set. Then, at each iteration, a new randomly chosen variable in considered and added to the set if it is conflicting with the other variables already in the set. At the end, we check if the resulting set violates a conflict inequality. If this is the case, the mutually conflicting set is kept. The method is called n times, where n is the number of items.

The exact method generates all mutually conflicting sets using a tree search-based algorithm. First, the y_{it} variables are sorted in ascending order based on item index i and height position t. Let $Ord(y_{it})$ be the order of variable y_{it}. At the root node, for which there is no mutually conflicting set, a child node is created for each variable and the associated conflicting set is a singleton with only this variable. In subsequent nodes, a new variable is added to the set only if it is of higher order than any variable in the set and if the variable is in conflict with all variables in the set. Each mutually conflicting set produced in this way is checked to see if it violates an inequality. If it does, the conflicting set is kept.

If the number of variables with a positive value is high, there might just be too many mutually conflicting sets. Based on preliminary experiments, the exact method is called only if this number is smaller than or equal to 1.1 times the number of items n. Otherwise, the heuristic method is used. Note also that each inequality must be violated by a value greater than or equal to 0.2 and that a maximum of 10 inequalities are added at each call.

The resulting inequalities are strengthened before they are added to the model. For a conflict inequality (4.15), a corresponding inequality of type (4.18) is added as it cuts fractional parts of the feasible domain. To generate an inequality of type (4.18) from an inequality of type (4.15), set C is first transformed into a minimal GUB cover by removing the variables with the largest weights. Then, variables with a positive value that are in conflict with all variables already in C are added to the set (considering these variables first proved to generate stronger inequalities). Then, the same procedure is applied to the remaining variables. We then find the *lifting coefficients* for the resulting extension by solving a series of *knapsack problems* (see [102] for details). As previously mentioned, only one extension is created by considering the variables to be added to C in random order.

For a max-flow inequality (4.19), the extension $E = \{y_{it} \in Y : i \in I(C) \text{ and } t_i^{min} \leq t \leq t_i^{max}\}$ is used. That is, only the variables associated with the items that are already in C and such that t is between t_i^{min} and t_i^{max} are considered.

4.5 Preprocessing

In this section, the various procedures that are applied during the initial preprocessing phase are described.

4.5.1 Height position restrictions

If the first items to be delivered are at the bottom of the bin, the corresponding solution is likely to be infeasible due to the unloading constraints. Figure 4.4 a) shows that the items to be delivered before item i must necessarily be in areas B and C. That is, if they are in area A, item i must be moved to get to these items, which is forbidden. Similarly, the items to be delivered after i must be in areas A and C. Figure 4.4 b) shows that if item i is at some height position $y_i \in [0,H]$, the unloading constraints might force some items outside of the bin (we recall that the height position of an item is defined as the height position of its bottom-left corner).

Motivated by some lower bounds reported in [46], we define y_i^{min} and y_i^{max} as the minimal and maximal height positions of item $i \in I$:

$$y_i^{min,1} = \max\left\{ \left\lceil \frac{\Sigma_{(i)}^{>,W-w_i}}{W} \right\rceil, \max_{j \in I_i^{>,W-w_i}} \{h_j\} \right\} \tag{4.20}$$

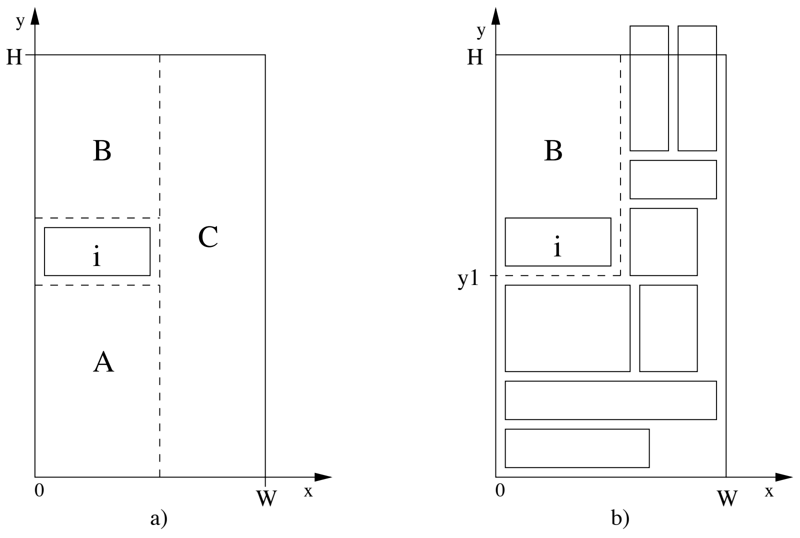

Figure 4.4 – Height position restrictions

$$y_i^{min,2} = \left\lceil \frac{[\Sigma_{(i)}^{>} - H(W - w_i) + \sum_{j \in I_i^{\leq,w_i}} h_j(w_j - w_i)]^+}{w_i} \right\rceil \qquad (4.21)$$

$$y_i^{max,1} = H - \max\left\{ \left\lceil \frac{\Sigma_{(i)}^{<,W-w_i}}{W} \right\rceil, \max_{j \in I_i^{<,W-w_i}} \{h_j\} \right\} \qquad (4.22)$$

$$y_i^{max,2} = H - \left\lceil \frac{[\Sigma_{(i)}^{<} - H(W - w_i) + \sum_{j \in I_i^{\geq,w_i}} h_j(w_j - w_i)]^+}{w_i} \right\rceil \qquad (4.23)$$

$$y_i^{min} = \max\{y_i^{min,1}, y_i^{min,2}\} \qquad (4.24)$$

$$y_i^{max} = \min\{y_i^{max,1}, y_i^{max,2}\} \qquad (4.25)$$

Equations (4.20) and (4.21) are obtained by considering only the items that must lie below some item i. More precisely, the value $y_i^{min,1}$ corresponds to the maximum between a continuous relaxation over all items that must lie below i and the item of maximum height that must also lie below i. The value $y_i^{min,2}$ is the height position obtained by pushing item i downward as long as the bottom of the bin is not reached or an item in area A, by moving to area C, does not force an item outside of the bin. Similar values in equations (4.22) and (4.23) are obtained by considering the items that must lie above i.

The obtained values y_i^{min} and y_i^{max} in equations (4.24) and (4.25) can be improved in two different ways. First, by considering items that cannot be beside i. Let us suppose, for example, that item j is to be delivered after i and that $w_i + w_j > W$. In this case, $y_j^{min} + h_j$ is clearly a lower bound on y_i^{min}. The second improvement can be obtained by considering only normal patterns. In fact, the value y_i^{min} should be the minimum height position of item i only if it corresponds to a normal pattern (i.e., the bottom of item i touches the top of another item or the bottom of the bin). If this is not the case, it can be reset to the minimum height position larger than y_i^{min} which corresponds to a normal pattern. The same line of reasoning can be applied to y_i^{max}. We thus obtain :

$$y_i^{min} = \max\{y_i^{min,1}, y_i^{min,2}, \max_{j \in I_i^{>,W-w_i}} \{y_j^{min} + h_j\}, \min\{t : t \in P_i^H\}\} \tag{4.26}$$

$$y_i^{max} = \min\{y_i^{max,1}, y_i^{max,2}, \min_{j \in I_i^{<,W-w_i}} \{y_j^{max} - h_i\}, \max\{t : t \in P_i^H\}\} \tag{4.27}$$

4.5.2 Lifting

As shown by previous authors [4, 22], if we do not find any combination of widths such that their sum is equal to W, then it is possible to reduce the width W as some unused space will remain. The same idea can be applied to the height H. The values W^* and H^* in equations (4.28) and (4.29) can be obtained through dynamic programming. At the end, we reset W to W^* and H to H^*.

$$W^* = \max\{\sum_{i \in I} z_i w_i : \sum_{i \in I} z_i w_i \leq W, \ z_i \in \{0,1\}, i \in I\} \tag{4.28}$$

$$H^* = \max\{\sum_{i \in I} z_i h_i : \sum_{i \in I} z_i h_i \leq H, \ z_i \in \{0,1\}, i \in I\} \tag{4.29}$$

A similar procedure can then be applied to the item sizes. The increase in width and height of item $i \in I$ is given by the following equations :

$$\Delta w_i = W - \max\{w_i - \sum_{j \in I \backslash \{i\}} z_j w_j : \sum_{j \in I \backslash \{i\}} z_j w_j \leq W - w_i,\ z_j \in \{0,1\}, j \in I \backslash \{i\}\} \quad (4.30)$$

$$\Delta h_i = H - \max\{h_i - \sum_{j \in I \backslash \{i\}} z_j h_j : \sum_{j \in I \backslash \{i\}} z_j h_j \leq H - h_i,\ z_j \in \{0,1\}, j \in I \backslash \{i\}\} \quad (4.31)$$

Clearly, not all items can be found at a given height position $t \in P^H$, either because it is not part of their normal patterns or because y_i^{min} is too large or y_i^{max} is too small for some item $i \in I$. Thus, it might well happen that the width occupied by the set of remaining items I^t at height position t can only be smaller than W. In this case, we can further reduce the width W to W_t :

$$W_t = \max\{\sum_{i \in I^t} z_i w_i : \sum_{i \in I^t} z_i w_i \leq W : z_i \in \{0,1\}, i \in I^t\} \qquad t \in P^H \quad (4.32)$$

We can extend this line of reasoning to individual items. Let us assume that item i is at height position t. Then, we can consider the other items one by one and use the previously described maximum flow algorithm to detect if it can be beside item i or not (see section 4.4.2). After removing all items that cannot be beside i, there might always be an empty space beside i. In this case, we can increase the width of w_i to w_{it} to cover the empty space. The w_{it} values can be obtained by solving (in pseudo-polynomial time) subset sum-like problems using dynamic programming.

4.5.3 Precedence relations

Let us consider items i and j such that i is delivered after j (i.e., $seq_i > seq_j$). We want to know if there is a feasible solution where height position t_j is smaller than t_i. If there is no such solution, we have a precedence relation between items i and j, because the height position of i must be smaller than or equal to the the height position of j for a solution to be feasible.

Formally, we identify two different types of precedence relations : a weak precedence relation where $t_i \leq t_j$ and a strong precedence relation where $t_i + h_i \leq t_j$. Furthermore, we consider a side-by-side relation where $t_i \leq t_j < t_i + h_i$ or $t_j \leq t_i < t_j + h_j$ always holds. The sets of weak precedence, strong precedence and side-by-side relation associated with item i are denoted $H(i)^-$, $H(i)^+$ and $H(i)^=$, respectively.

In the following, we present conditions for such precedence relations to occur between items i and j.

Condition 1 : If $y_i^{max} + h_i \leq y_j^{min}$, then $j \in H^+(i)$.

Condition 2 : If $y_i^{max} \leq y_j^{min}$ then $j \in H^-(i)$.

Condition 3 : If $maxflow(C) < \sum_{k \in I \setminus \{i,j\}} h_k w_k$ for all $t_i \in P_i^H$ and $t_j \in P_j^H$ such that $t_j < t_i$ and $C = \{y_{it_i}, y_{jt_j}\}$ is a mutually conflicting set, then $j \in H^-(i)$.

Condition 4 : If $w_i + w_j > W$, then $j \in H^+(i)$..

Condition 5 : If $maxflow(C) < \sum_{k \in I \setminus \{i,j\}} h_k w_k$ for all $t_i \in P_i^H$ and $t_j \in P_j^H$ such that $t_j < t_i + h_i$ and $C = \{y_{it_i}, y_{jt_j}\}$ is a mutually conflicting set, then $j \in H^+(i)$.

Condition 6 : If $maxflow(C) < \sum_{k \in I \setminus \{i,j\}} h_k w_k$ for all $t_i \in P_i^H$ and $t_j \in P_j^H$ such that $t_i > t_j$ or $t_i < t_j$ and $C = \{y_{it_i}, y_{jt_j}\}$ is a mutually conflicting set, then $j \in H^=(i)$.

From Conditions 1 to 3, we obtain inequalities (4.33) where the height position of item j is forced to be greater than or equal to the height position of item i, when i is weakly preceding j. From Conditions 4 and 5, we get inequalities (4.34) where the height position of item j is forced to be greater than or equal to the height position of item i plus its height h_i, when i is strongly preceding j. Finally, Condition 6 leads to inequalities (4.35) where items i and j are forced to be side-by-side.

$$y_{it} \leq \sum_{\substack{\bar{t} \in P_j^H \\ \bar{t} \geq t}} y_{j\bar{t}} \qquad i \in I, j \in H^-(i),\ t \in P_i^H \tag{4.33}$$

$$y_{it} \leq \sum_{\substack{\bar{t} \in P_j^H \\ \bar{t} \geq t + h_i}} y_{j\bar{t}} \qquad i \in I, j \in H^+(i),\ t \in P_i^H \tag{4.34}$$

$$y_{it} \leq \sum_{\bar{t} \in P_j^H(t)} y_{j\bar{t}} \qquad i \in I, j \in H^=(i),\ t \in P_i^H, \tag{4.35}$$

4.5.4 Reconsidering the normal patterns

In this section, we describe two different ways of reducing the initial set of normal patterns.

4.5.4.1 Normal pattern dominance

It is possible to reduce the set of normal patterns by considering a dominance relation among the patterns associated with a given item. Let i be an item and $t_i \in P_i^H$ some height position corresponding to a pattern. If i is at height position t_i and nothing fits over i, then nothing will fit over i at any height position $t > t_i$. Let $t_i^{min} \in P_i^H$ be the lowest height position such that nothing fits over item i. Then every height position $t \in P_i^H$ such that $t \geq t_i^{min}$ can be removed. This type of relation is called *Normal Pattern Dominance*.

4.5.4.2 Normal pattern reduction

Previously, we introduced a feasibility test based on the solution of a maximum flow problem to know if a partial solution based on a mutually conflicting set of items leads to an infeasible solution. This test can also be used to prove the feasibility of a normal pattern. Suppose that item i is at height position $t_i \in P_i^H$. If all mutually conflicting sets that include t_i fail the feasibility test then t_i can be removed from P_i^H. But since there might be a huge number of mutually conflicting sets, a more viable approach is the following. For any given item i and height position $t_i \in P_i^H$, t_i can be removed from P_i^H if there is an item j such that for all $t_j \in P_j^H$ and mutually conflicting set $C = \{y_{it_i}, y_{jt_j}\}$, $maxflow(C)$ is smaller than $\sum_{k \in I \setminus \{i,j\}} h_k w_k$. The value t_i is then added to a set INF_i that contains infeasible height positions for item i.

By using the previously defined y_i^{min} values, y_i^{max} values and INF_i sets, an improved procedure can be designed to calculate a reduced set of normal patterns. The pseudo-code of this procedure is found in Algorithm 1 for the typical case where the bin is filled from the bottom to the top.

The normal patterns are generated by a call to the method *Calculate Normal Patterns*. Three sets *Cur*, *Prev* and *Glob* are first initialized with position 0 which is the bottom of the bin. Set *Prev* contains all normal patterns generated up to (but not including) the current customer j. Note that *Cur* is reset to *Prev* before generating the patterns of each item associated with customer j. The patterns for item $i \in I_j$ are generated recursively by the method *Generate Patterns* if the current customer has more than one item. When the set of all patterns associated with item i has been produced, the global variable P_i^H is assigned with this set. The newly generated patterns are also added to set *Glob*. When all items of the current customer are done, *Prev* is set to *Glob* just before handling the next customer. The methods *Generate Patterns* and *Generate Patterns Recursively* consider all possible ordered combinations of items in set I and make sure that the corresponding patterns are feasible normal patterns.

The complexity of this algorithm is $O(Hn(o-1)!)$ where H is the height of the bin, n is the number of items and $o = \max_{j=1,\dots,R} |I_j|$. The algorithm performs well when the number of items per customer is small. However, if $(o-1)!$ is larger than n, it is better to use the dynamic algorithm based on equations (4.4) and (4.5), as reported in [84], which is in the order of $O(Hn^2)$.

Algorithm 1 Calculate Normal Patterns

```
Require: $y_i^{min}, y_i^{max}$ and $INF_i$
  $Prev \leftarrow \{0\}$
  $Cur \leftarrow \{0\}$
  $Glob \leftarrow \{0\}$
  for $j = R$ to 1 do
    for $i \in I_j$ do
      $Cur \leftarrow Prev$
      if $I_j \backslash \{i\} \neq \emptyset$ then
        Generate Patterns($I_j \backslash \{i\}$, $Cur$)
      end if
      $P_i^H \leftarrow \{t \in Cur \backslash INF_i : y_i^{min} \leq t \leq y_i^{max}\}$
      for $t \in P_i^H$ do
        $Cur \leftarrow Cur \cup \{t + h_i\}$
      end for
      $Glob \leftarrow Glob \cup Cur$
    end for
    $Prev \leftarrow Glob$
  end for
```

Algorithm 2 Generate Patterns

```
Require: $I$ : set of items, $P$ : set of patterns
  $min_y \leftarrow \min_{i \in I} y_i^{min}$
  $max_y \leftarrow \max_{i \in I} y_i^{max}$
  for $t = max_y$ to $min_y$ do
    if $t \in P$ then
      for $i \in I$ do
        Generate Patterns Recursively($t + h_i$, $I \backslash \{i\}$, $P$)
      end for
    end if
  end for
```

Algorithm 3 Generate Patterns Recursively

```
Require: $t$ : a starting position, $I$ : set of items, $P$ : set of patterns
  if $t \in P$ then
    Exit
  end if
  $P \leftarrow P \cup \{t\}$
  for $i \in I$ do
    if $y_i^{min} \leq t \leq y_i^{max}$ and $t \notin INF_i$ then
      Generate Patterns Recursively($t + h_i$, $I \backslash \{i\}$, $P$)
    end if
  end for
```

4.5.5 Top-fill versus bottom-fill

Typically, a bottom-left approach is used to generate the set of normal patterns (as in Algorithm 1). More precisely, the bottom of every item must be in contact with either the bottom of the bin or the top of another item. Furthermore, the left side of each item must be in contact with the left side of the bin or the right side of another item.

Due to the unloading constraints, solutions to our problem exhibit a special structure which can be exploited by choosing to fill the bin from the bottom, from the top or from both the bottom and the top (mixed fill), in the hope of reducing the set of normal patterns. With regard to the mixed approach, if the items with delivery order j (customer j) fill the bin from the bottom, then the items with delivery order k (customer k) with $k > j$ also fill the bin from the bottom. Conversely, if the items with delivery order j fill the bin from the top, then the items with delivery order k with $k < j$ also fill the bin from the top.

For a sequence of R customers, there are $R+1$ ways to choose the cut point between the customers whose items will fill the bin from the bottom and the customers whose items will fill the bin from the top. It is then possible to choose, among these $R+1$ cut points, the one that leads to the smallest number of normal patterns. Figure 4.5 presents an example of a mixed fill where the cut point is chosen between customers 3 and 4, that is, items of customers 1, 2 and 3 fill the bin from the top and items of customers 4, 5 and 6 fill the bin from the bottom.

Proposition 4.5.1. *If a solution is feasible for a given cut point, it is feasible for every cut point.*

Proof. Let us assume that a feasible solution s_1 is obtained when the cut point is in position 1, that is, just before the first customer (which corresponds to filling the bin from the bottom, as it is typically done). Then, a feasible solution s_2 for the cut point in position 2, i.e. between customers 1 and 2, is obtained from s_1 by pushing the items of customer 1 up until the top of the bin is reached. Similarly, a feasible solution s_3 for the cut point in position 3, i.e. between customers 2 and 3, is obtained from s_2 by pushing the items of customer 2 up until the top of the bin or the bottom of another item is reached.

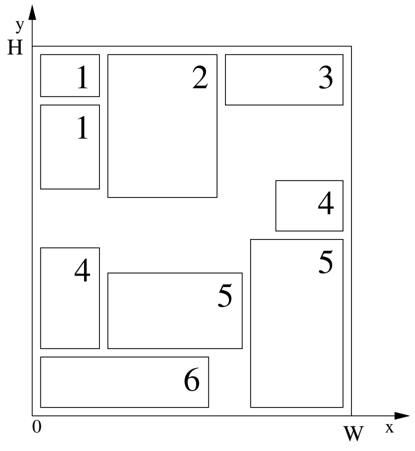

Figure 4.5 – Mixed Fill

The result is obtained by repeating this argument until the cut point is in position $R+1$.

This proposition also means that if a solution is infeasible for a given cut point, it is infeasible for every cut point. Thus, we are guaranteed not to miss any feasible solution. As indicated in the demonstration of the above proposition, there might also be a gap between the top-filled and bottom-filled items. Inspired by the work in [22], a constraint is associated with each item and each height position to force down (up) a top-filled item so that its bottom (top) touches another item which must be delivered after (before) it. These constraints are the following :

$$y_{it} \leq \sum_{j=seq_i}^{R} \sum_{\substack{k \in I_j \\ k \neq j}} y_{k(t-h_k)} \qquad t \in P_i^H,\ i \in I \text{ and } i \text{ is bottom-filled} \tag{4.36}$$

$$y_{it} \leq \sum_{j=1}^{seq_i} \sum_{\substack{k \in I_j \\ k \neq j}} y_{k(t+h_k)} \qquad t \in P_i^H,\ i \in I \text{ and } i \text{ is top-filled} \tag{4.37}$$

4.6 Lower bounds

In this section we present some lower bounds on the required height of the bin.

4.6.1 Simple Lower Bounds

First, we can use the lower bound for the *SPP* reported in [116]. This so-called continuous lower bound, denoted L_1, calculates the number of strips of width W and height 1 needed to accommodate all items.

$$L_1 = \left\lceil \frac{\sum_{i \in I} w_i h_i}{W} \right\rceil \tag{4.38}$$

From the definition of y_i^{min} and y_i^{max} in equations (4.26) and (4.27), the following lower bound can also be derived :

$$L_2 = \max_{i \in P} \{ y_i^{min} + h_i + (H - y_i^{max}) \} \tag{4.39}$$

It is worth noting that this bound improves upon the one reported in [46], through the addition of the third term in the numerator of $y_i^{min,2}$ and $y_i^{max,2}$ in equations (4.21) and (4.23) and through the consideration of the third and fourth components in the definition of y_i^{min} and y_i^{max} in equations (4.26) and (4.27).

4.6.2 Lower Bounds based on the Cutting Stock Problem

The bounds presented in this section are based on the Gilmore-Gomory formulation of the *Cutting Stock Problem* (CSP). We define a cutting pattern as a subset of items $I' \subset I$. A pattern is said to be h-feasible if $\sum_{i \in I'} h_i \leq H$ and w-feasible if $\sum_{i \in I'} w_i \leq W$. Let $\mathscr{K}^H$ and $\mathscr{K}^W$ be the sets of all h-feasible and w-feasible cutting patterns. Let also the variable v_k be equal to the number of times cutting pattern k appears in a solution with $a_{ik} = 1$ if item i is in cutting pattern k, 0 otherwise. The CSP based on w-feasible cutting patterns (CSP^W) can be formulated as follow :

$$(CSP^W) \qquad \min \sum_{k \in \mathscr{K}^W} v_k \tag{4.40}$$

$$\sum_{k \in \mathscr{K}^W} a_{ik} v_k \geq h_i \qquad i \in I \tag{4.41}$$

$$v_k \geq 0 \; \textit{and integer} \qquad k \in \mathscr{K}^W \tag{4.42}$$

The optimal solution value of (CSP^W) provides a lower bound on the required height of the bin. Thus, an instance is infeasible if the obtained value is larger than H. A CSP^H can be defined similarly to obtain a lower bound on the required width. We will call these two lower bounds L_3^H and L_3^W, respectively.

The CSP is famous for the strength of its linear relaxation and it is often the case that the round up value of the linear relaxation is optimal. Column (cutting pattern) generation is typically used to solve this problem and it is well known that the pricing subproblem is a knapsack problem (KP), where the item values come from the dual variables. We refer to [13] for more details on this issue in the context of the 2OPP. A column of negative reduced cost generated at a given width or height position often corresponds to subsets of items that cannot be side by side or one over the other because the corresponding KP does not account for any unloading structure. Clearly, integrating some unloading structure can only improve L_3^W and L_3^H.

For example, we can improve L_3^H to obtain L_4^H by considering only the items $i \in I$ that can be at some height position, based on the current set P^H and the values y_i^{min} and y_i^{max}. Although we might still end up with infeasible columns, this approach has proven to be very effective.

We can also improve L_3^W to obtain L_4^W. Here, we must generate a feasible *stack* of items of negative reduced cost at some width position, as illustrated in Figure 4.6. For generating stacks, an algorithm similar in spirit to the one used for generating normal patterns is applied (see section 4.5.4). Let $V(j,h)$ be the sum of the item values over all items that are below position h in an optimal stack for customers with a delivery order between j and R. Also, let S be a subset of items associated with a given customer and consider that function $\delta(S,h)$ returns the minimum height required by the items in S, if they are all below position h (if this is not feasible then $\delta(s,h) = -\infty$). To find a stack of negative reduced cost, one needs to solve the following recursion for $V(1,H)$:

$$\begin{aligned} V(j,h) = \max\{\ & V(j+1,h), \\ & \max_{s \subseteq I_j}\{\ \sum_{i \in S} \lambda_i + V(j+1, \delta(S,h))\ \}\ \} \end{aligned}$$

It should be noted that $V(j,h) = \infty$ for $j > R$ in this recursion.

If either L_4^H or L_4^W is larger than H or W, the items cannot fit within an area of size HW, but the value obtained is not necessarily a lower bound on the required height or width of the bin. For example, if $L_4^H = H + \Delta H$ with $\Delta H > 1$, the items might well fit within an area of size (H+1)W, because increasing H by only one unit typically leads to many new feasible height and width positions in sets P^H and P^W.

We end this section by observing that the lower bounds for the Strip Packing Problem or the Two-Dimensional Bin Packing Problem are also valid lower bounds for the 2OPP-UL. We refer the reader to [4, 22] for an exhaustive list of lower bounds for the *SPP*.

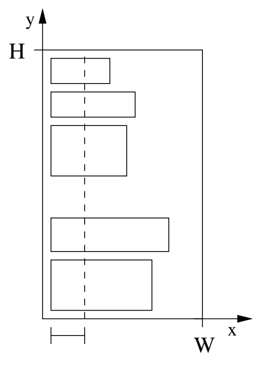

Figure 4.6 – Feasible stacking

4.7 Computational results

In this section, we first analyze the lower bounds presented in section 4.6. Then, the impact of the preprocessing is quantified. Finally, our branch-and-cut algorithm is compared with the algorithm in [89], where the authors implemented the tree search-based enumeration scheme proposed in [116] with additional fathoming criteria. This is the best known exact algorithm for solving packing problems with unloading constraints. The authors were kind enough to give us their code for this comparison. Our branch-and-cut algorithm is implemented in C++ and calls Cplex 12.5. The tests were performed on a 2.2 GHz AMD Opteron 275 processor running under Linux.

4.7.1 Instances

The Two-Dimensional Loading Capacitated Vehicle Routing Problem ($2L-CVRP$) instances reported in [76] were used for testing purposes. As indicated in Table 4.I, there are 5 different types of instances, with 36 instances of each type, for a total of 180 instances. The height H and width W of the bin are equal to 40 and 20, respectively. The number of items per customer is indicated in column *Items per cust*. In types 2 to 5, each item also has one of three different dimensions, referred to as *vertical*, *horizontal* and *homogeneous*. The exact number of items per customer and dimension values were randomly generated in the intervals shown in Table 4.I. The largest instances have up to 255 customers, 786 items and a fleet of 51 vehicles.

We also created additional instances by simply modifying the dimensions of the bin, as indicated below. With 180 instances in each class, we end up with a total of 900 different $2L-CVRP$ instances.

	# Items	Vertical		Horizontal		Homogeneous	
Type	per cust.	Height	Width	Height	Width	Height	Width
1	1	1	1	1	1	1	1
2	[1,2]	[.4H,.9H]	[.1W,.2W]	[.1H,.2H]	[.4W,.9W]	[.2H,.5H]	[.2W,.5W]
3	[1,3]	[.3H,.8H]	[.1W,.2W]	[.1H,.2H]	[.3W,.8W]	[.2H,.4H]	[.2W,.4W]
4	[1,4]	[.2H,.7H]	[.1W,.2W]	[.1H,.2H]	[.2W,.7W]	[.1H,.4H]	[.1W,.4W]
5	[1,5]	[.1H,.6H]	[.1W,.2W]	[.1H,.2H]	[.1W,.6W]	[.1H,.3H]	[.1W,.3W]

Tableau 4.I – Types of instances

- Class 1 : H = 40, W = 20
- Class 2 : H = 32, W = 25
- Class 3 : H = 50, W = 16
- Class 4 : H = 80, W = 14
- Class 5 : H = 130, W = 14

4.7.2 Lower Bounds

Tables 4.II, 4.III and 4.IV compare our lower bounds on the packing instances generated from the 180 original $2L-CVRP$ instances in [76]. Note that these bounds are useless for Type 1 because all items have their width and height equal to 1. For this comparison, solutions (sets of routes) were generated for each $2L-CVRP$ instance with the *ALNS* heuristic in [142]. For efficiency purposes, some bounds taken from [4, 22, 116] were calculated before inserting a customer into a route. These bounds could detect, in particular, situations where the additional area required by the items of the current customer would lead to infeasibility by exceeding the total area of the bin. Accordingly, lower bound L_1 is automatically satisfied. For each $2L-CVRP$ instance, the *ALNS* heuristic was run for 20,000 iterations and a maximum of 800 best solutions produced during the search were kept. The total number of routes (or packing instances) in these solutions appears in column *# Instances* of Table 4.II. The average number of items per packing instance is also shown in column *# Items*. After applying the height position restrictions and the lifting procedure, the lower bounds *SPP*, L_2, L_2^{COR}, L_3^H and L_3^W were applied on each individual packing instance to detect infeasibility (see the algorithmic flow in section 4.3.1). In the case of *SPP*, it should be noted that we really have a collection of lower bounds, including the dual functions in [22] and the so-called L_8 bound in [4]. Also, L_2^{COR} corresponds to the bound reported in the work of Cordeau et al. [46], which is improved by L_2. The value in each entry is the percentage of infeasible packing instances detected by the corresponding lower bound. The CPU time in seconds is only shown for L_3^H and L_3^W, because the other ones are too small.

Table 4.III then shows the performance of the preprocessing routines, without the application of L_4^H and L_4^W, and the resulting number of undecided packing instances. Then, the performance of L_4^H and L_4^W, which are applied to the undecided instances at the end of the preprocessing, is reported (see the algorithmic flow in section 4.3.1).

Overall, we can see that the performance substantially diminishes from the type 2 instances to the type 5 instances, which are clearly the most difficult ones. Lower bounds L_3^H and L_3^W largely outperform the simple bounds SPP, L_2 and L_2^{COR} (with our L_2 bound only slightly better than L_2^{COR} on instances of type 2). The preprocessing routines, as a whole, are quite good, but L_4^H and L_4^W still allow a substantial number of additional infeasible instances to be detected.

Table 4.IV shows the percentage of packing instances that were found infeasible only by the corresponding lower bound or only by the preprocessing routines. It also reports the percentage of instances where L_2 was better than SPP and L_4^H was better than L_4^W (by detecting infeasibility while the other did not). We can see that L_4^H is worth considering even if it is not as good as L_4^W in Table 4.II. Note also that the simple bounds are still useful, in spite of the 0.0% value in each entry, because they can detect infeasible instances quickly, that is, before going into the more sophisticated preprocessing routines.

4.7.3 Impact of preprocessing

For this study, we started with the routes produced by the *ALNS* heuristic when applied to the five classes of $2L-CVRP$ instances, while keeping a maximum of 800 visited solutions for each instance. The feasibility of the obtained routes (packing instances) was then assessed with a simple heuristic reported in [89], which is derived from the classical Bottom-Left heuristic. If a packing instance could be proven feasible, it

			SPP	L_2^{COR}	L_2	L_3^H		L_3^W	
Type	# Instances	# Items	Inf.	Inf.	Inf.	Inf.	sec	Inf.	sec
2	84571	8.7	24.8%	2.8%	3.0%	38.5%	0.008	40.3%	0.007
3	81637	11.0	8.6%	1.2%	1.2%	20.0%	0.011	25.5%	0.011
4	81806	13.2	3.3%	0.2%	0.2%	7.5%	0.013	12.9%	0.012
5	70132	19.8	0.0%	0.0%	0.0%	0.1%	0.015	0.3%	0.016

Tableau 4.II – Comparison of lower bounds SPP, L_2^{COR}, L_2, L_3^H and L_3^W

Type	# Instances	# Items	Preprocessing Inf.	Preprocessing sec	# Instances	L_4^H Inf.	L_4^H sec	L_4^W Inf.	L_4^W sec
2	84571	8.7	89.3%	0.002	9066	13.3%	0.010	23.9%	0.007
3	81637	11.0	54.3%	0.024	37292	11.1%	0.011	23.1%	0.008
4	81806	13.2	20.4%	0.089	65154	6.1%	0.012	16.4%	0.008
5	70132	19.8	0.2%	0.389	70020	0.2%	0.014	0.7%	0.011

Tableau 4.III – Comparison of L_4^H and L_4^W after preprocessing

Type	SPP	L_2^{COR}	L_2	L_3^H	L_3^W	Preprocessing	L_4^H	L_4^W	$L_2 >$ SPP	$L_4^H >$ L_4^W
2	0.0%	0.0%	0.0%	0.0%	0.0%	44.9%	0.2%	1.1%	0.5%	0.2%
3	0.0%	0.0%	0.0%	0.0%	0.0%	27.2%	0.6%	5.0%	0.6%	1.3%
4	0.0%	0.0%	0.0%	0.0%	0.0%	10.6%	0.7%	6.2%	0.1%	1.4%
5	0.0%	0.0%	0.0%	0.0%	0.0%	0.1%	0.1%	0.3%	0.0%	0.1%

Tableau 4.IV – Comparison of different lower bounds

was discarded. Then, a simple enumeration procedure was applied for 20 seconds and, again, if the feasibility or infeasibility of the packing instance could be proven, it was discarded. A maximum of 8 undecided packing instances were kept for each $2L-CVRP$ instance. At the end, a total of 2,183 packing instances were collected. Clearly, this process led to difficult packing instances, which is exactly what we wanted. The number of instances collected by type and class are shown in Table 4.V. We note, in particular, that all instances of type 1 were discarded, while only one instance of type 2 was still undecided. Thus, the packing instances of type 1 and 2 are quite easy to solve.

Table 4.VI shows the average CPU time and number of max flow problems generated during the preprocessing phase (since this number has a significant impact on the CPU time) over each class, based on the 2,183 remaining instances. The average number of items per instance and dimensions of the bin for each class are also shown. Then, Table 4.VII compares the impact of each individual preprocessing routine. The values shown on each line correspond to the increase in percentage in the number of normal patterns

	Type					
Class	1	2	3	4	5	Total
1	0	0	30	198	200	428
2	0	1	128	198	200	527
3	0	0	2	155	206	363
4	0	0	0	187	182	369
5	0	0	11	246	239	496
Total	0	1	171	984	1027	2183

Tableau 4.V – Number of instances by class and type

for each class when the corresponding preprocessing routine is removed from the basic implementation (with all routines). In this Table, L stands for lifting, HR for height restrictions, *PR* for precedence relations, *NPD* for Normal Pattern dominance, *NPR* for normal pattern removal and TBF for mixed top, bottom-fill. Table 7 also includes the average number of maximum flow problems and the CPU time when the corresponding preprocessing routine is removed. The method with the largest impact is clearly TBF. Without it, the number of normal patterns increases by 22% to 57% when compared to the implementation with all routines. The HR method has also a significant impact on the CPU time by reducing the number of maximum flow problems to be solved.

4.7.4 Comparison with another exact algorithm

The results of the comparison between our algorithm and the one reported in [89] on the 2,183 difficult packing instances with unloading constraints are shown in Table 5.II. Column *# Inst.* is the number of instances of each type in each class. Then, columns *Fea.* and *Inf.* show the number of proven feasible and infeasible instances, within the allowed 1,200 seconds of computation time, while the average CPU time in seconds is shown in column *sec*. Column *Solved* reports the total number of solved instances (either feasible or infeasible).

We observe that our algorithm performs very well in comparison with the one reported by Iori et al. [89]. In particular, we can solve 1088 new instances (while only one previously solved instance was not solved by our algorithm). The computation times are also significantly smaller. It should be noted that the smallest undecided instance has 20 items, as compared with 13 items for the algorithm of Iori et al. Conversely, the largest instance solved with our algorithm has 52 items, as compared with 29 items for Iori et al.

	Class 1	Class 2	Class 3	Class 4	Class 5	All
CPU time (sec.)	0.3	0.1	0.5	1.6	9.5	2.6
# Max flow	16 683.2	9 194.0	27 960.6	77 845.9	313 626.7	93 942.4
# Items	17.6	16.6	18.6	23.9	36.9	23.0
$[H,W]$	[40,20]	[32,25]	[50,16]	[80,14]	[130,14]	

Tableau 4.VI – Preprocessing with all routines

Routine	Class 1	Class 2	Class 3	Class 4	Class 5	# Max flow	CPU time (sec.)
L	0.47%	1.28%	0.06%	0.08%	0.17%	84 003.0	2.2
HR	0.02%	0.00%	1.02%	2.10%	4.29%	286 133.6	8.5
PR	0.03%	0.00%	0.00%	0.18%	0.56%	98 421.2	2.8
NPD	9.79%	11.70%	6.98%	1.91%	0.42%	94 935.9	2.6
NPR	0.03%	0.00%	0.00%	0.18%	0.56%	43 517.0	1.5
TBF	55.26%	57.77%	50.83%	37.09%	22.84%	92 248.7	2.6

Tableau 4.VII – Impact of each preprocessing routine

		#	*Iori et al. (2007)*					*Our Branch & Cut*				
Class	Type	Inst.	Fea.	sec.	Inf.	sec.	Solved	Fea.	sec.	Inf.	sec.	Solved
1	3	30	4	34.1	26	111.7	30	4	0.2	26	0.4	30
	4	198	17	164.9	106	377.2	123	27	1.1	171	1.2	198
	5	200	5	509.9	1	595.3	6	53	196.1	108	95.4	161
2	2	1	0		1	0.3	1	0		1	0.1	1
	3	128	4	2.0	124	61.3	128	4	0.1	124	0.3	128
	4	198	17	175.9	110	380.9	127	23	0.8	175	0.6	198
	5	200	9	309.3	4	730.6	13	64	63.0	117	43.9	181
3	3	2	0		2	19.6	2	0		2	0.1	2
	4	155	20	223.0	53	431.9	73	29	3.0	126	3.4	155
	5	206	9	337.5	0		9	54	199.4	103	118.4	157
4	4	187	18	298.5	30	366.7	48	41	90.1	141	60.0	182
	5	182	3	280.6	0		3	37	200.6	46	222.9	83
5	3	11	0		5	555.6	5	2	14.3	9	6.3	11
	4	246	8	421.7	0		8	45	268.5	119	217.5	164
	5	239	0		0		0	2	602.3	11	296.8	13
Total		2183	114		462		576	385		1279		1664

Tableau 4.VIII – Comparison of two algorithms for the 2L-CVRP

4.8 Conclusion

This paper has demonstrated the effectiveness of a new branch-and-cut algorithm for a two-dimensional packing problem with unloading constraints through numerical results on a set of benchmark instances. The next step will now consist in integrating this algorithm into a problem-solving methodology for a mixed vehicle routing and loading problem. Alternative approaches, in particular dynamic programming, are also currently considered to tackle the packing problem.

CHAPITRE 5

THE VEHICLE ROUTING PROBLEM WITH STOCHASTIC TWO-DIMENSIONAL ITEMS

Dans ce travail, nous proposons une méthode de résolution pour un problème rencontré dans les compagnies de livraison de meubles. À chaque jour, un ensemble d'items rectangulaires doivent être livrés chez des clients. Malheureusement, au moment de la planification des tournées les dimensions de certains items ne sont pas connues avec certitude. Il est toutefois possible de définir une distribution de probabilité discrète sur les dimensions des items à partir d'un historique des livraisons effectuées dans le passé.

Le problème est modélisé sous la forme d'un problème de programmation stochastique à deux niveaux que nous résolvons à l'aide de la méthode *L-shaped* en nombres entiers de [100]. La réalisabilité des routes produites est validée avec les algorithmes développés dans le travail précédent. De plus, nous proposons des bornes inférieures sur la valeur du recours qui peuvent être utilisées pour le problème classique de tournées avec demandes stochastiques. Nous démontrons l'efficacité de l'approche sur des instances déterministes et ainsi que sur des instances stochastiques.

The Vehicle Routing Problem with Stochastic Two-Dimensional Items

Jean-François Côté †§ Michel Gendreau ‡§, Jean-Yves Potvin †§

†Département d'informatique et de recherche opérationnelle, Université de Montréal, C.P. 6128, succ. Centre-Ville, Montréal, Québec, Canada H3C 3J7.

‡Département de mathématiques et de génie industriel, École Polytechnique de Montréal, C.P. 6079, succ. Centre-Ville, Montréal, Québec, Canada H3C 3A7.

§CIRRELT, Université de Montréal, C.P. 6128, succ. Centre-Ville, Montréal, Québec, Canada H3C 3J7.

Abstract

We consider a stochastic vehicle routing problem where a discrete probability distribution characterizes the two-dimensional size (height and width), as well as the weight of a subset of items to be delivered to customers. Although some item sizes and weights are not known with certainty when the routes are planned, they become known when it is time to load the vehicles, just before their departure. If it happens that not all items can be loaded in a vehicle, the items of one or more customers are put aside which lead to a penalty (or recourse cost). The objective is to minimize the sum of the routing and recourse costs.

The problem is modeled as a two-stage stochastic program and solved with the integer L-shaped method. Some new inequalities and lower bounds are proposed. Computational results are reported on test instances specifically generated for this problem, as well as classical instances for the deterministic case.

5.1 Introduction

In the last decades, several variants of the classical Vehicle Routing Problem (VRP) have been introduced. In its simplest form, the VRP is aimed at at building routes, starting and ending at a central depot, to serve a set of customers with a fleet of identical vehicles. These routes must then satisfy various side constraints. Typically, each customer has a known demand (quantity of goods or number of items to be delivered or picked-up) and the total demand on a route should not exceed vehicle capacity.

Recently, mixed vehicle routing and loading problems have been studied [88]. In these problems, the packing of the items inside the loading area of the vehicle must be taken into account. In this work, a Two-Dimensional Orthogonal Packing Problem (2OPP) is considered where rectangular items must be delivered to customers. The items cannot be rotated and must fit in the rectangular loading area of each vehicle without overlap while satisfying unloading constraints. That is, at each delivery location, it should be possible to unload the items of the current customer by pulling them out of the vehicle without moving any item of other customers. Figure 5.1 shows an example for a route starting from the depot 0 and visiting the customers 1, 2, 3, 4 and 5, in this order. The figure shows two packings for this route : the first one satisfies the unloading constraints while the second one does not (note that the items are taken out from the top, which corresponds to the rear of the vehicle). In the second case, items of customers 2 and 5 must be moved to allow the items of customer 1 to be unloaded.

The 2OPP is often found as a subproblem of the Two-Dimensional Strip Packing Problem (2SPP) and the Two-Dimensional Bin Packing Problem (2BPP). Recent exact methods for the 2OPP can be found in [25, 38, 67, 122], while exact methods for the 2SPP are found in [4, 6, 22, 51, 116]. A recent work on the 2SPP with unloading constraints is reported in [147] where a GRASP heuristic, previously proposed in [3], and two approximation algorithms are used to solve the problem. The authors in [50] also report an exact method for the 2OPP with unloading constraints based on a Benders decomposition of the problem, which can solve instances with up to 52 items.

Mixed vehicle routing and loading problems are often addressed by local search heu-

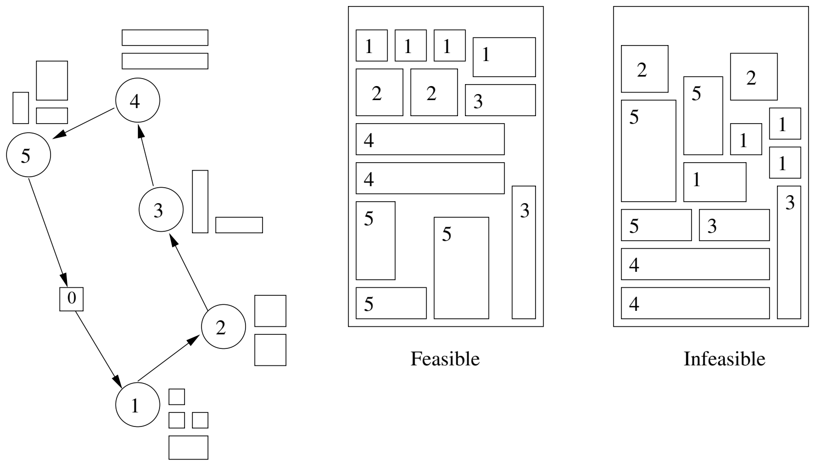

Figure 5.1 – Packing examples

ristics which improve the current solution by applying different classes of modifications to the current routes (see, for example, [62, 71, 76, 105, 165]). Any modification to a vehicle route must lead to a feasible packing. To this end, simple packing heuristics like the Bottom-Left, Bottom-Left Fill and Touching Perimeter heuristics [165] are typically used. Only a few exact algorithms are reported in the literature for vehicle routing and loading problems due to the difficulty of the packing. In [89], a branch-and-cut algorithm is proposed where the classical Bottom-Left heuristic is first used to solve the packing and, if it fails, an exact branch-and-bound algorithm, based on the work in [116], is applied. The reader is referred to the surveys in [19, 88] for the three-dimensional case.

A collaboration with an industrial partner unveiled an important issue that arises in some practical applications. Quite often, the size of a few items might not be available when the delivery routes are planned, although the item type provides an indication about possible sizes and their corresponding probabilities. It means that a discrete probability distribution can be associated with these items, leading to a new class of stochastic VRPs, namely the Vehicle Routing with Stochastic Two-Dimensional Items (S2L-CVRP). When the sizes become known prior to vehicle loading and departure, it is possible that not all items assigned to a given vehicle in the planned solution will fit in the vehicle. In this case, the items of one or more customers are left on the dock (for delivery on some other day), leading to a penalty or recourse cost based on the number of unserved customers. The solution approach proposed here relies on the concept of a priori optimization [16]. That is, planned routes are built in a first-stage solution, without knowing the exact size of some items. Then, in a second-stage, the sizes become known and the recourse policy is applied when a failure occurs. The objective is to minimize the sum of the routing and (expected) recourse costs.

Different types of stochastic VRPs are reported in the literature, depending on the nature of the stochastic components. Of particular interest is the VRP with stochastic demands, where the (scalar) demand at some customers is not known with certainty [162]. These problems are typically solved with the integer L-shaped method [100], which is an extension of the method reported in [158] for continuous stochastic programs. The integer L-shaped method is basically a branch-and-cut algorithm where the expectation component of the objective is linearly bounded with optimality cuts. The method was later improved in [85, 90, 101] by introducing a weaker form of optimality cuts called Lower Bounding Functionals (LBFs) which are valid for a wider range of integer and fractional solutions. For example, VRPs with stochastic demands with up to 80 vertices (and 2 vehicles) have been solved in [90] using LBFs. However, the method becomes less effective when the number of vehicles increases because LBFs are derived from aggregation of partial routes. In [33], the authors partially address this problem by proposing a branch-and-price algorithm, but the latter is tailored for customer demands following a Poisson distribution and the size of the search space increases very quickly with the magnitude of the demand values. In our work, we go one step further along the LBF research avenue by proposing disaggregated LBFs. The idea comes from the disaggregated optimality cuts proposed in [146] where the recourse cost is distributed over a number of variables, not just a single variable.

The remainder of the paper is organized as follows. First, a formal definition and a mathematical model for our problem is presented in Section 5.2. Then, the integer L-shaped method is described in Section 5.3. Section 5.4 introduces our disaggregated LBFs, called *L*-cuts, which are based on several lower bounds on the recourse cost. These bounds are obtained by considering the set of customers in each route of the current solution. Another type of set-based inequalities is then presented in Section 5.5.

The optimality cuts are introduced in Section 5.6. The approach for solving the packing problems that arise during the execution of our algorithm is described in Section 5.7. Computational results are finally reported in Section 5.8.

5.2 Model

The Vehicle Routing Problem with Stochastic Two-Dimensional Items or S2L-CVRP is defined as follows. We are given a complete undirected graph $G = (V, E)$ where $V = \{0, 1, 2, ..., n\}$ is the set of vertices of cardinality $n + 1$ and $E = \{(j, k) : j, k \in V : j < k\}$ is the set of edges with their associated cost c_{jk}, $(j, k) \in E$. Also, vertex 0 is the depot while $C = V \backslash \{0\}$ is the set of customers. We assume that K identical vehicles are available to execute delivery routes that start and end at the depot. The loading area of each vehicle has height H, width W and a maximum weight capacity Q. In this work, unloading constraints are also considered : at every service location, the items of the current customer can be unloaded by pulling them out of the vehicle without moving any item from other customers.

Each customer $j \in C$ has a demand for m_j two-dimensional items. Let I be the set of all items with cardinality $\sum_{j \in C} m_j = m$. For each item $i \in I$, there are d_i possible sizes in height, width and weight with an associated probability distribution ($d_i = 1$ for a deterministic item). That is, $\sum_{r=1}^{d_i} p_i^r = 1$ for every item $i \in I$, where p_i^r is the probability that item i has width w_i^r, height h_i^r and weight q_i^r, $w_i^r \leq W$, $h_i^r \leq H$, $q_i^r \leq Q$. Then,

$$\bar{a}_j = \sum_{i=1}^{m_j} \sum_{r=1}^{d_i} p_i^r h_i^r w_i^r$$

is the average area covered by the items of customer j and

$$\bar{q}_j = \sum_{i=1}^{m_j} \sum_{r=1}^{d_i} p_i^r q_i^r$$

is their average weight. In a feasible solution, the items delivered on each route must fit within the loading area of the vehicle, their total weight should not exceed capacity Q

and the unloading constraints should be satisfied. When stochastic items are delivered on a given route, the average or expected area covered by these items plus the actual area covered by the deterministic items must be less than or equal to the loading area of the vehicle. Similarly, the expected weight of the stochastic items plus the actual weight of the deterministic items should be less than or equal to the vehicle capacity.

Although the actual sizes and weights of the stochastic items are unknown when the delivery routes are planned, they become known just prior to the loading and departure of the vehicles. If it happens that the items to be transported by a vehicle do not fit into the loading area, a recourse action must be considered. In our application, the items of one or more customers are left on the dock. In this case, these customers will be delivered later, which negatively impacts service quality. Accordingly, the recourse cost is based on the number of unserved customers.

The S2L-CVRP can be formulated as a stochastic VRP with additional constraints. In our case, we use the classical two-index formulation where x_{jk} is equal to 1 if edge $(j,k) \in E$ is used (with $j < k$), 0 otherwise . We also denote $F(x)$ the expected cost of the recourse of solution $x = (x_{jk})$. The formulation of the S2L-CVRP is then :

$$\min \sum_{j<k} c_{jk} x_{jk} + F(x) \tag{5.1}$$

$$\sum_{j \in C} x_{0j} = 2K \tag{5.2}$$

$$\sum_{h<j} x_{hj} + \sum_{k>j} x_{jk} = 2 \qquad j \in C \tag{5.3}$$

$$\sum_{j,k \in S} x_{jk} \leq |S| - \left\lceil \max\left\{ \frac{\sum_{j \in S} \bar{a}_j}{HW}, \frac{\sum_{j \in S} \bar{q}_j}{Q} \right\} \right\rceil \qquad S \subset C, 2 \leq |S| \leq n \tag{5.4}$$

$$\sum_{(j,k) \in R} x_{jk} \leq |R| - 1 \qquad R \in \mathscr{R}^{inf} \tag{5.5}$$

$$x_{jk} \in \{0,1\} \qquad 0 \leq j < k \leq n \tag{5.6}$$

The objective (5.1) is to minimize the sum of the routing and expected recourse costs. Constraints (5.2) force the fleet of K vehicles to be used. Constraints (5.3) state that exactly two edges must be used by a vehicle to visit a customer. Constraints (5.4) are the subtour-breaking and rounded-capacity constraints. Then, constraints (5.5) prohibit all routes that do not satisfy the loading requirements, including unloading constraints, which is denoted by the set $\mathscr{R}^{inf}$. This set contains infeasible routes with only deterministic customers, as well as routes with stochastic customers for which all possible realizations or scenarios are infeasible. In these constraints, route R is defined by the set of edges covered by the corresponding vehicle. Finally, the binary requirement on the decision variables is found in constraints (5.6). Note that by forcing the x_{jk} variables to be 0 or 1, back-and-forth routes to a single customer are forbidden (as it is typically done, see [89, 101]).

The recourse cost $F(x)$ is

$$F(x) = \sum_{R \in \mathscr{R}_x} F(R) \tag{5.7}$$

where $\mathscr{R}_x$ is the set of routes in solution x and $F(R)$ is the recourse cost of route R. Note that $F(R) = 0$ when route R has only deterministic items. Let Ω_R be the set of all possible realizations or scenarios for route R and p_{ω_R} the probability of scenario $\omega_R \in \Omega_R$. This probability is equal to the product of each item's probability of the scenario. Then $F(R)$ is calculated as follows :

$$F(R) = c_f \cdot \sum_{\omega_R \in \Omega_R} p_{\omega_R} F(\omega_R) \tag{5.8}$$

where c_f is the cost associated with each unserved customer and $F(\omega_R)$ is the number of unserved customers under scenario ω_R. Thus, $F(R)$ is the expected recourse cost of route R over all scenarios.

5.3 The Integer L-Shaped Method

The integer L-shaped method is used to solve our problem. This method can be seen as a variant of the classical branch-and-cut algorithm for the deterministic VRP. First, the rounded-capacity (5.4) and infeasible path (5.5) constraints are relaxed and $F(x)$ in the objective is replaced by a lower bound θ to obtain the following initial model :

$$\min \sum_{j<k} c_{jk} x_{jk} + \theta \tag{5.9}$$

$$\sum_{j \in C} x_{0j} = 2K \tag{5.10}$$

$$\sum_{h<j} x_{hj} + \sum_{k>j} x_{jk} = 2 \qquad j \in C \tag{5.11}$$

$$x_{jk} \in \{0,1\} \qquad 0 \leq j < k \leq n \tag{5.12}$$

This problem is solved with CPLEX, the latter being in charge of computing solutions to the linear relaxations and branching on fractional variables. The inequalities are generated through methods that are called automatically by the CPLEX solver.

A pseudo-code for our L-Shaped method is shown in Algorithm 4. At each node of the branching tree, we first check for violated rounded capacity inequalities or RCIs (5.4). Then, two alternatives must be considered depending if the solution is fractional or not. If it is fractional, we look for violated *L*-cuts (5.15) and infeasible set inequalities (5.37). If there are none, then we branch on fractional variables. For an integer solution, the general idea is to go from the easiest to the hardest. First, the focus is on the non-ordered set of customers *S* associated with each route to generate *L*-cuts and infeasible set inequalities (5.37) which apply to a larger number of solutions (i.e., all routes where the customers in set *S* are visited consecutively, whatever the order of those visits). When no new inequalities of these types can be generated, we validate every route for infeasible path inequalities (5.5) and *D*-optimality cuts (5.40). These inequalities have a more restricted scope, as they apply to a single route, and are generated through more computationally expensive procedures. At the end, the best feasible integer solution found by the L-Shaped method is returned.

The rounded capacity inequalities (5.4) are classical inequalities that are generated using the package CVRPSEP from [113] and they will not be discussed anymore. The other inequalities and lower bounds mentioned in Algorithm 4 will be described in the following.

5.4 Lower Bounding Functionals

The L-cuts described in this section are in the class of Lower Bounding Functionals (LBFs). Although these cuts are used to bound θ in the objective, they are not optimality cuts because they come from a lower bound on the recourse (based on the number of unserved customers). They apply to non-ordered sets of customers visited consecutively in a route of the solution x^ν associated with the current node in the branching tree. Using a lower bound weakens the inequality but, on the other hand, its scope is larger than a single route, as it is valid for any route where the customers in a given set are visited consecutively, whatever their order.

5.4.1 L-cuts

Formally, let us consider the set of customers S in a route of solution x^ν with at least one stochastic item. For this set, we also assume that $\sum_{j\in S}\bar{a}_j \leq HW$ (i.e., an inequality of type (5.4) has been generated). Let us also denote $L(\omega_S)$ a lower bound on the number of unserved customers for a given realization or scenario $\omega_S \in \Omega_S$ with $L(S) = c_f \sum_{\omega_S\in\Omega_S} p_{\omega_S} L(\omega_S)$ (see Section 5.4.2 for a description of this lower bound).

Then, by arbitrarily selecting the customer j_S of minimum index among customers with stochastic items in set S, we have :

$$\theta_{j_S} \geq L(S)\cdot\left(\sum_{j\in S} x_{0j} + x(S) - |S|\right) \tag{5.13}$$

Algorithm 4 L-Shaped Method

```
Require: a problem $N$
1: Solve the linear relaxation of problem $N$ to obtain solution $x^v$
2: if violated RCIs (5.4) are found then add them and go to Step 1
3: if $x^v$ is fractional then
4:    if violated $L$-cuts (5.15) and infeasible set inequalities (5.37) are found then add
      them and go to Step 1
5:    else branch on fractional variables and call the L-Shaped Method recursively with
      each subproblem
6: else
7:    for each route $R$ in $x^v$ do
8:       $S \leftarrow$ set of customers in route $R$
9:       if $S$ contains only customers with deterministic items then
10:         Calculate a lower bound $LB(S)$ on the number of required vehicles
11:         if $LB(S) > 1$ then add an infeasible set inequality (5.37)
12:      else
13:         Calculate a lower bound on the recourse cost of route $R$ based on set $S$ for
            each possible scenario
14:         if the lower bound on the recourse is positive over all scenarios then add
            infeasible set inequality (5.37) with $LB(S) = 2$
15:         else if the lower bound on the recourse is positive for at least one scenario
            then add $L$-cut (5.15)
16:      end if
17:   end for
18:   if inequalities were added then go to Step 1
19:   for each route $R$ in $x^v$ do
20:      if route $R$ contains only customers with deterministic items then
21:         if the packing problem is infeasible then add an infeasible path inequality
            (5.5)
22:      else
23:         Calculate the exact recourse cost of route $R$ for each scenario by solving the
            corresponding packing problem
24:         if the recourse is positive over all scenarios then add an infeasible path in-
            equality (5.5)
25:         else if the recourse is positive for least one scenario add a $D$-optimality cut
26:      end if
27:   end for
28:   if inequalities were added then go to Step 1
29:   else a new feasible integer solution has been found
30: end if
```

where $x(S) = \sum_{j,k \in S} x_{jk}$ sums the visited edges among all pairs of customers in set S. Given that S corresponds to the set of customers in a route of solution x^v, two customers in set S are directly connected to the depot while $x(S)$ is equal to $|S|-1$. Accordingly, the right hand side of (5.13) reduces to $L(S)$ and defines a bound on θ_{j_S}.

This inequality can be extended by considering that the customers in set S can be visited (consecutively) before or after any other node, not only the depot. To this end, we propose the following approach. First, M variables θ_l, $l = 1,...,M$, where M is some predefined number, are created. Our goal is to assign a set of variables θ_l, denoted Θ_S, with each subset of customers S for which $L(S) > 0$. Each time such a subset is found using the procedure described in Section 5.4.3, it is processed by Algorithm 5 and then added to an (initially empty) set $\bar{\mathscr{S}}$. As described in the pseudo-code, the subsets S' already in $\bar{\mathscr{S}}$ are processed one by one and each time $(S \cup S')$ is feasible for at least one scenario, $\Theta(S)$ is updated by setting it to the union of $\Theta(S)$ and $\Theta(S')$. If $\Theta(S)$ remains empty at the end of this loop, $\Theta(S)$ is assigned to some unused θ_l variable. If all variables are used, $\Theta(S)$ is assigned to $\Theta(S^*)$, where $S^* = \text{argmax}_{S' \in \bar{\mathscr{S}}}\{|S \cap S'|\}$.

Algorithm 5 Assignment of θ_l variables

```
Require: $S$ : subset of customers with $L(S) > 0$
Require: $\bar{\mathscr{S}}$ : set of previously generated subsets of customers with $L(S) > 0$
Require: $\bar{\Theta}$ : set of unused $\theta_l$ variables
 1: for $S' \in \bar{\mathscr{S}}$ do
 2:     if $S \cup S'$ is feasible then
 3:         $\Theta(S) \leftarrow \Theta(S) \cup \Theta(S')$
 4:     end if
 5: end for
 6: if $\Theta(S) = \emptyset$ then
 7:     if $\bar{\Theta} \neq \emptyset$ then
 8:         Let $\theta_{l'}$ be an unused variable in $\bar{\Theta}$
 9:         $\Theta(S) \leftarrow \{\theta_{l'}\}$
10:         $\bar{\Theta} \leftarrow \bar{\Theta} \backslash \{\theta_{l'}\}$
11:     else
12:         $S^* = \text{argmax}_{S' \in \bar{\mathscr{S}}}\{|S \cap S'|\}$
13:         $\Theta(S) \leftarrow \Theta(S^*)$
14:     end if
15: end if
```

We then have :

$$\theta \geq \sum_{l=1}^{M} \theta_l \tag{5.14}$$

$$\sum_{\theta_l \in \Theta(S)} \theta_l \geq L(S) \cdot (x(S) - |S| + 2) \qquad S \in \bar{\mathscr{S}} \tag{5.15}$$

Proposition. The inequalities (5.14) and (5.15) provide a lower bound θ on the recourse cost.

Proof. We first need to define the recourse cost for any fractional or integer solution. Let x^ν be a solution of the linear relaxation of model (5.9) - (5.12) with possibly some additional, previously generated, inequalities. We define $\mathscr{S}^\nu = \{S \subseteq C \mid x^\nu(S) > |S| - 2,\ L(S) > 0 \text{ and feasible}\}$ the set of all subsets of customers S with $L(S) > 0$ such that a path defined over S is feasible for at least one scenario.

Let also $\mathscr{X}$ be a subset of $\mathscr{S}^\nu$ such that :

(a) For each $S, S' \in \mathscr{X}, S \neq S'$, $S \cup S'$ is not feasible under any scenario.

It means that the subsets in $\mathscr{X}$ are maximal when considered pairwise. Now, let $\mathscr{P}^\nu$ be the set of all subsets $\mathscr{X}$ of $\mathscr{S}^\nu$ for which condition (a) is satisfied. Then, the recourse cost $\mathscr{L}(x^\nu)$ can be defined as :

$$\mathscr{L}(x^\nu) = \max_{\mathscr{X} \in \mathscr{P}^\nu} \left\{ \sum_{S \in \mathscr{X}} L(S) \cdot (x^\nu(S) - |S| + 2) \right\} \tag{5.16}$$

Consider the example in Figure 2 where it is assumed that $\{3,4,5\}$, $\{4,5,6\}$ and $\{3,4,5,6\}$ are feasible and $L(\{3,4,5\})$, $L(\{4,5,6\})$, $L(\{3,4,5,6\})$ are all positive (i.e., they all cover at least one customer with stochastic items). Then, it is not possible for $\{3,4,5\}$ and $\{4,5,6\}$ to be together in some set $\mathscr{X}$ because these two sets can be combined to form the larger feasible set $\{3,4,5,6\}$. Accordingly, $L(\{3,4,5\})$ and $L(\{4,5,6\})$

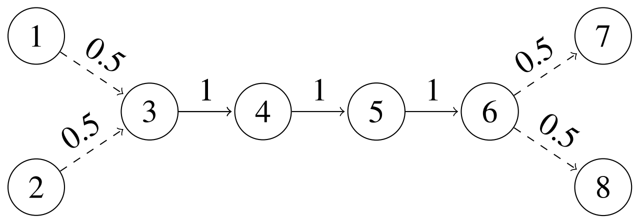

Figure 5.2 – A fractional solution

will not be summed up in (5.16), which is fine. Otherwise, $L(\{4,5\})$ would be added twice.

For an integer solution, every set of customers S in $\mathscr{X}$ corresponds to a full route in x^ν and $\mathscr{L}(x^\nu)$ sums up the $L(S)$ values of the set of customers associated with each route. In the case of a fractional solution, $\mathscr{L}(x^\nu)$ might not be equal to the exact recourse cost. Consider the sets $S_1 = \{3,4,5,6\}$, $S_2 = \{1,3,4,5,6\}$, $S_3 = \{2,3,4,5,6\}$, $S_4 = \{3,4,5,6,7\}$ and $S_5 = \{3,4,5,6,8\}$ in Figure 2 with $L(S_q) > 0$, $q = 1,...,5$. All those sets are such that $x^\nu(S) > |S| - 2$. Then, the set of subsets $\mathscr{X}$ maximizing (5.16) is $\mathscr{X} = \{S_2, S_3, S_4, S_5\}$ for which $L(x^\nu) = \frac{1}{2}[L(S_2) + L(S_3) + L(S_4) + L(S_5)] \geq \frac{1}{2}[L(S_1) + L(S_1) + L(S_1) + L(S_1)] = 2L(S_1)$, because S_1 is included in S_2, S_3, S_4 and S_5. Thus, the contribution of S_1 is counted twice.

Now, let us consider $\bar{\mathscr{S}}$ the set of previously generated subsets of customers S with $L(S) > 0$ in Algorithm 2. Let $\bar{\mathscr{X}}$ be a subset of $\bar{\mathscr{S}}$ such that :

(b) For each $S' \neq S'' \in \bar{\mathscr{X}}$, $\theta(S') \cap \theta(S'') = \emptyset$.

Let $\bar{\mathscr{P}}$ be the set of all $\bar{\mathscr{X}}$ which satisfy condition (b). Then, from inequality (5.15) and condition (b), we have :

$$\sum_{\theta_l \in \bigcup_{S \in \bar{\mathscr{X}}} \Theta(S)} \theta_l \geq \sum_{S \in \bar{\mathscr{X}}} L(S) \cdot (x(S) - |S| + 2) \qquad \bar{\mathscr{X}} \in \bar{\mathscr{P}} \tag{5.17}$$

where $\bigcup_{S \in \bar{\mathscr{X}}} \theta(S)$ is the union of the sets of variables $\Theta(S)$ over all S in $\bar{\mathscr{X}}$. We also have a fortiori :

$$\sum_{l=1}^{M} \theta_l \geq \sum_{S \in \bar{\mathscr{X}}} L(S) \cdot (x(S) - |S| + 2) \qquad \bar{\mathscr{X}} \in \bar{\mathscr{P}} \tag{5.18}$$

and :

$$\sum_{l=1}^{M} \theta_l \geq \max_{\bar{\mathscr{X}} \in \bar{\mathscr{P}}} \left\{ \sum_{S \in \bar{\mathscr{X}}} L(S) \cdot (x(S) - |S| + 2) \right\} \tag{5.19}$$

Variable θ, as defined in (5.14) and (5.15), is a lower bound on the recourse cost if the following inequality is satisfied for any solution x^ν :

$$\mathscr{L}(x^\nu) \geq \sum_{l=1}^{M} \theta_l \tag{5.20}$$

Sets $S \in \bar{\mathscr{S}}$ with $x^\nu(S) \leq |S| - 2$ can be ignored when considering $\bar{\mathscr{P}}$ because $L(S) \cdot (x^\nu(S) - |S| + 2)$ is negative for these sets. Let $\bar{\mathscr{P}}^\nu = \{\bar{\mathscr{X}} \in \bar{\mathscr{P}} |\, x^\nu(S) > |S| - 2 \; \forall S \in \bar{\mathscr{X}}\}$. Then, we have :

$$\sum_{l=1}^{M} \theta_l \geq \max_{\bar{\mathscr{X}} \in \bar{\mathscr{P}}^\nu} \left\{ \sum_{S \in \bar{\mathscr{X}}} L(S) \cdot (x^\nu(S) - |S| + 2) \right\} \tag{5.21}$$

Using inequalities (5.20) and (5.21) as well as the definition of $\mathscr{L}(x^\nu)$ in (5.16), we obtain :

$$\max_{\mathscr{X} \in \mathscr{P}^\nu} \left\{ \sum_{S \in \mathscr{X}} L(S) \cdot (x^\nu(S) - |S| + 2) \right\} \geq \max_{\bar{\mathscr{X}} \in \bar{\mathscr{P}}^\nu} \left\{ \sum_{S \in \bar{\mathscr{X}}} L(S) \cdot (x^\nu(S) - |S| + 2) \right\} \tag{5.22}$$

For inequality (5.22) to be true for any solution x^ν, we must show that $\bar{\mathscr{P}}^\nu \subseteq \mathscr{P}^\nu$. That is, if $\bar{\mathscr{X}} \in \bar{\mathscr{P}}^\nu$ then $\bar{\mathscr{X}} \in \mathscr{P}^\nu$. But we know from Algorithm 2 that $\Theta(S') \cap \Theta(S'') = \emptyset$ for $S', S'' \in \bar{\mathscr{X}}$, $S' \neq S''$ implies that $S' \cup S''$ is not feasible. Thus, the sets in $\bar{\mathscr{X}}$ satisfy condition (a). Given that these sets also satisfy $x^\nu(S) > |S| - 2$, $\bar{\mathscr{X}} \in \mathscr{P}^\nu$.

The benefits of this extension come from the fact that the subsets of customers S do not need to be connected to the depot and the recourse cost can be bounded as long as there is a path going through S. On the other hand, two sets S' and S'' with only a few customers (or even none) in common might be associated with the same θ_l variables, which induces a weaker bound on the recourse, see line 12 in Algorithm 2. But, initial experiments on difficult instances have shown that the subsets of customers S are often associated with a single variable. We will refer to inequalities (5.15) as L-cuts in the following.

5.4.2 Lower bounds

To obtain a lower bound on the recourse $L(S)$ for a set of customers S in a route of solution x^v, we need to calculate a lower bound $L(\omega_S)$ for each scenario $\omega_S \in \Omega_S$. A tight lower bound can be obtained by solving a special knapsack problem with two-dimensional items where the knapsack stands for the loading area of the vehicle and where one unit of gain is achieved when all items of a given customer are in the loading area (note that the exact position of each item in the loading area does not need to be considered because S is not ordered). Given that solving this knapsack problem is computationally expensive, we rather consider three different relaxations and one feasibility test, leading to four lower bounds. These lower bounds are calculated in the order $L_1(\omega_S)$, $L_2(\omega_S)$, $L_3(\omega_S)$ and $L_4(\omega_S)$. As soon as one of these bounds is found to be strictly positive, the calculations stop and $L(\omega_S)$ is assigned to this positive value.

Bound L_1

Let h_i and w_i be the height and width of item i of customer j under scenario $\omega_S \in \Omega_S$ with $a_j = \sum_{i=1,\ldots,m_j} h_i w_i$. Also, let z_j be a binary variable which is equal to 1 when all items of customer j under scenario ω_S are in the loading area. Then $L_1(\omega_S)$ can be formulated as follows :

$$L_1(\omega_S) = |S| - \max\left\{\sum_{j\in S} z_j : \sum_{j\in S} a_j z_j \le HW,\ z_j \in \{0,1\},\ j \in S\right\} \tag{5.23}$$

This bound can be easily obtained. We just need to sort the set of customers in non decreasing order of a_j and add them iteratively until the loading area HW is exceeded.

Bound L_2

The next lower bound is based on the solution of dual feasible functions [37]. More precisely, we consider L_{dff}^{BM} in [22] which returns a lower bound on the required height of the loading area to accommodate the items of all customers in set S. If this value is larger than the height H of the loading area, then at least one customer cannot be served.

$$L_2(\omega_S) = \begin{cases} 1 & \text{if } L_{dff}^{BM}(\omega_S) > H \\ 0 & \text{otherwise} \end{cases} \tag{5.24}$$

Bound L_3

The lower bound $L_3(\omega_S)$ is obtained by considering the Gilmore-Gomory formulation of the Cutting Stock Problem (CSP). Here, a pattern is defined through a subset of items I' taken from the set of items delivered to the customers in set S. A pattern is said to be H-feasible if $\sum_{i \in I'} h_i \leq H$ and W-feasible if $\sum_{i \in I'} w_i \leq W$. Let P^H and P^W be the sets of all such H-feasible and W-feasible patterns. We also have two different types of variables : z_j which is equal to 1 when all items of customer j are in the solution, 0 otherwise, and y_p which is the number of times pattern p is selected. Finally, we have $a_{ip} = 1$ if item i is in pattern p, 0 otherwise. Then, $L_3(\omega_S)$ is defined as follows :

$$L_3(\omega_S) = |S| - \max \sum_{j \in S} z_j \tag{5.25}$$

$$\sum_{p \in P^W} a_{ip} y_p = h_i z_j \qquad j \in S,\ i = 1 \text{ to } m_j \tag{5.26}$$

$$\sum_{p \in P^H} a_{ip} y_p = w_i z_j \qquad j \in S,\ i = 1 \text{ to } m_j \tag{5.27}$$

$$\sum_{p \in P^W} y_p \le H \tag{5.28}$$

$$\sum_{p \in P^H} y_p \le W \tag{5.29}$$

$$\sum_{j \in S} q_j z_j \le Q \tag{5.30}$$

$$y_p \ge 0 \text{ and integer} \qquad p \in P^H \cup P^W \tag{5.31}$$

$$z_j \in \{0,1\} \qquad j \in S \tag{5.32}$$

Constraints (5.26) and (5.27) guarantee that h_i W-feasible and w_i H-feasible patterns are selected if item i is in the solution. Constraints (5.28), (5.29) and (5.30) relate to the size of the loading area and the maximum weight.

Solving the mathematical programming model (5.25) - (5.32) to obtain L_3 is computationally expensive and we rather solve, using column generation, a continuous relaxation where the y_p variables are continuous and $0 \le z_j \le 1$.

Bound L_4

The last lower bound $L_4(\omega_S)$ is based on the exact solution of the One-Dimensional Contiguous Bin Packing Problem (1CBP), a tight relaxation of the 2OPP, using the branch-and-bound algorithm in [51]. If the problem is feasible then all customers fit within the loading area, otherwise at least one customer must be removed.

$$L_4(\omega_S) = \begin{cases} 1 & \text{if 1CBP under scenario } \omega_S \text{ is infeasible} \\ 0 & \text{otherwise} \end{cases}$$

5.4.3 Separation procedure

This section describes the separation procedure aimed at identifying violated L-cuts when the current solution x^v is fractional (see line 4 in Algorithm 4). The pseudo-code of this procedure is found in Algorithm 3 where :

$$a_j^{\max} = \sum_{i=1}^{m_j} \max_{r=1,\ldots,d_i} h_i^r w_i^r$$

$$x'(S',S'') = \sum_{j \in S'} \sum_{k \in S''} x_{jk}^v$$

.

Algorithm 6 Generation of L-cuts

```
Require: $C_s$ : set of customers with stochastic items
 1: for $j \in C_s$ do
 2:     $S \leftarrow \{j\}$
 3:     repeat
 4:         if $\sum_{k \in S} a_k^{\max} > HW$ then
 5:             for $\omega_S \in \Omega_S$ do
 6:                 Calculate $L(\omega_S)$
 7:             end for
 8:             if at least one feasible scenario $\omega_S$ then
 9:                 Generate $L$-cut and go to Step 1.
10:             end if
11:         end if
12:         $j^* \leftarrow \text{argmin}_{k \in C \backslash S}\{x'(S \cup \{k\}, V \backslash (S \cup \{k\}))\}$
13:         $S \leftarrow S \cup \{j^*\}$
14:     until $\sum_{k \in S} \bar{a}_k > HW$ or $x'(S, V \backslash S) \geq 4$
15: end for
```

As indicated in this pseudo-code, each customer with stochastic items is considered in turn to initialize set S. At each step of the following iterative procedure, the customer j^* minimizing $x'(S \cup \{k\}, V \backslash (S \cup \{k\}))$ over $k \in C \backslash S$ is selected and added to S. Then, if (1) the mean area covered by the items of all customers in S is smaller than HW, (2) the maximum possible area covered by those same items is greater than HW and (3) the summation over the variables x_{jk}^v with $j \in S$ and $k \in V \backslash S$ is less than 4, there is an opportunity to generate a new inequality. For quick filtering purposes, the packing problem associated with each scenario is first solved using the Bottom-Left heuristic. If it happens that all packing problems are feasible, no L-cut can be generated. Otherwise, the lower bound $L(S)$ is calculated by summing $L(\omega_S)$ over every scenario $\omega_S \in \Omega_S$. If $L(S) > 0$, a new L-cut (5.15) is added to the model. Note that when $x'(S, V \backslash S) \geq 4$, then $x(S) \leq |S| - 2$ and set S covers at least two different routes in solution x^v. So, there is no hope of generating a new L-cut. Note also that when there is no feasible scenario in line 8, then an infeasible set inequality (5.37) can be generated with $LB(S) = 2$.

5.5 Other set-based inequalities

When a route contains only deterministic items, a lower bound $LB(S)$ on the number of vehicles required to serve the set of customers S in the route can be calculated (lines 9-11 in Algorithm 4). If the lower bound indicates that more than one vehicle is needed, then an infeasible set inequality can be generated. The bound $LB(S)$ is derived from the bounds LB_1, LB_2, LB_3 and LB_4, which are described below. As soon as one of these bounds is greater than 1 (i.e., more than one vehicle is required to serve set S), the calculations stop and $LB(S)$ is set to this value. It should be noted that LB_2 and LB_3 come from a previous work on the Strip Packing Problem (SPP) where bounds are proposed on the required height of the loading area to accommodate all customers in set S [4, 51]. Dividing these values by the height H of the loading area provides a lower bound on the number of vehicles.

Bound LB_1

The first lower bound LB_1 is the classical *continuous* bound on the required area, where $a_j = \sum_{i=1}^{m_j} h_i w_i$.

$$LB_1(S) = \left\lceil \frac{\sum_{j \in S} a_j}{HW} \right\rceil \tag{5.33}$$

Bound LB_2

The lower bound LB_2 is obtained by taking the maximum value between L_{dff}^{BM} in [22] and L_3 in [4].

$$LB_2(S) = \left\lceil \frac{\max\{L^{BM}_{dff}(S), L_3(S)\}}{H} \right\rceil \tag{5.34}$$

Bound LB_3

The lower bound LB_4 comes from the work in [51].

$$LB_3(S) = \left\lceil \frac{L_4(S)}{H} \right\rceil \tag{5.35}$$

Bound LB_4

The lower bound LB_4 is obtained by solving the One-Dimensional Contiguous Bin Packing Problem (1CBP), a tight relaxation of the 2OPP, using the branch-and-bound algorithm in [51]. If the problem is infeasible, at least two vehicles are required to serve the set of customers S.

$$LB_4(S) = \begin{cases} 2 & \text{if 1CBP is infeasible for set } S \\ 1 & \text{otherwise} \end{cases} \tag{5.36}$$

Then, if $LB(S) > 1$, we can generate the following infeasible set inequality :

$$x(S) \leq |S| - LB(S) \tag{5.37}$$

5.6 Optimality cuts

The optimality cuts are considered at the end of Algorithm 4 (lines 23-25). At this point, the current solution x^ν is integer and no new inequalities have been generated in the previous steps. Assuming that x^ν is feasible, the following optimality cut can be generated to bound θ in the objective :

$$\theta \geq \sum_{R \in \mathscr{R}_{x^\nu}} F(R) \cdot \left(\sum_{R \in \mathscr{R}_{x^\nu}} \sum_{(j,k) \in R} x_{jk} - (n + K - 1) \right) \tag{5.38}$$

Given that the x_{jk} variables involved in the double summation are all equal to 1 in solution x^ν, this double summation equals $n+K$ for x^ν and the right hand side of equation (5.38) reduces to its recourse cost. For any other solution, the inequality is trivially satisfied.

It should be noted that this optimality cut is aggregated over all routes and applies only to solution x^ν, which is definitely a weakness. We thus propose disaggregated optimality cuts or *D*-optimality cuts which apply to individual routes. These cuts have been proposed in [146], but have never been implemented in practice. To generate them, the initial relaxed model (5.9) - (5.12) is extended by first defining a θ_j variable for each customer j with stochastic items and by adding inequality (5.39), where C_s stands for the set of customers with stochastic items.

During the execution of the L-shaped method, the customer with stochastic items of minimum index j_R is arbitrarily selected among each route with stochastic items in solution x^ν and a cut of type (5.40) is generated for every one of those routes.

$$\theta \geq \sum_{j \in C_s} \theta_j \tag{5.39}$$

$$\theta_{j_R} \geq F(R) \cdot \left(\sum_{(j,k) \in R} x_{jk} - |R| + 1 \right) \tag{5.40}$$

Like the aggregated cut (5.38), the right-hand side of (5.40) for route R reduces to its recourse cost. For every other route $R' \neq R$, the inequality is trivially satisfied.

To calculate the exact recourse $F(R)$ of route R, we need to account for the number of unserved customers under every possible scenario. To this end, we start with the lower bound on the number of unserved customers $L(\omega_S)$ in Section 5.4.2, where S is the set of customers in route R and $\omega_S \in \Omega_S$ is a possible realization or scenario for set S (or, equivalently, $\omega_R \in \Omega_R$ is a possible realization or scenario for route R). For any ω_S, $|S| - L(\omega_S)$ is an upper bound on the number of customers contained in the loading area. We thus enumerate all subsets of customers in route R with at most $|S| - L(\omega_S)$ customers and sort them from largest to smallest. Ties are broken by giving priority to subsets which cover a larger area. Then, we consider these subsets one by one and solve the corresponding packing problem (see Section 5.7) until a feasible solution is found with the corresponding number of unserved customers and recourse cost. The exact recourse cost $F(R)$ of route R is obtained at the end by summing these recourse costs over all possible scenarios, weighted by the corresponding scenario probability, see equation (5.8). Although this approach might appear computationally expensive, our tests have shown that only a few packing problems need to be solved.

5.7 Packing problems

This section discusses the methodology for solving the packing problems that are generated during the execution of Algorithm 4. For routes with only deterministic items, solving the packing problem is aimed at determining if the route is feasible or if a new infeasible path inequality (5.5) can be generated (lines 20-21). For routes with stochastic items, a packing problem is solved for every possible scenario to calculate the exact recourse cost of the route and generate a D-optimality cut (5.40) if the route is feasible or a new infeasible path inequality (5.5) otherwise (lines 23-25). The latter case occurs when the recourse cost is positive under every scenario (i.e., there is always at least one unserved customer).

Different approaches are used to quickly detect if a route is feasible or infeasible. As described in the following, the 2OPP and then the full 2OPP with unloading constraints (UL) are considered in this order. The 2OPP is considered first because it is a simpler problem than the 2OPP-UL and it is easily obtained by relaxing the unloading constraints.

The various approaches listed below are called one by one until the infeasibility of the 2OPP (and thus, the 2OPP-UL as well) is proven :

1. A simple lower bound is obtained by summing the areas of all items in the route. The latter is infeasible if this sum is larger than the loading area.

2. The more sophisticated lower bound L_{dff}^{BM} on the required height of the loading area is then used [22]. If this bound is larger than the height H of the loading area, then the route is infeasible. It should be noted that only the first three dual feasible functions are used here. The fourth one, which proved to be time consuming and not really effective during preliminary tests, was disregarded.

3. Another lower bound on the required height of the loading area is obtained by invoking the alternating constructive procedure reported in [4].

4. Two additional lower bounds on the height and width of the loading area are based on the Gilmore-Gomory formulation of the Cutting Stock Problem. They correspond to L_3^H and L_3^W in [50]. If these bounds are larger than H and W, respectively, the route is infeasible.

5. The One-Dimensional Contiguous Bin Packing Problem (1CBP), a tight relaxation of the 2OPP, is finally solved with the branch-and-bound algorithm in [51]. If there is no feasible solution to the 1CBP, then the route is infeasible.

If the 2OPP has not been proven to be infeasible, we then consider the real problem, namely the 2OPP-UL, and apply the following procedures in this order to determine its feasibility or infeasibility :

1. The problem is first solved with an approximate method, namely a variant of the heuristic reported in [104], originally developed for the Two-Dimensional Strip Packing Problem (2SPP). This is a two-phase heuristic, where a solution is first constructed and then improved with simulated annealing. Here, the original construction heuristic is replaced by the Bottom-Left and Max-Touching Parameter heuristics [165] to address the unloading constraints. If a feasible packing is found with this heuristic, the route is feasible.

2. The lower bound L_2 for the 2OPP-UL, reported in [50], is then used to estimate the required area. If the value of L_2 is larger than the loading area HW, then the route is infeasible.

3. The branch-and-bound algorithm reported in [22], originally developed for the 2SPP, has been adapted to the 2OPP-UL. It is applied with the following additional fathoming criterion : if an item does not fit at any position among a set of precalculated positions, then the current partial solution cannot lead to any feasible solution and the node can be fathomed. In practice, this algorithm can often find feasible solutions very quickly. We allow the generation of a maximum of 1,000,000 nodes in the branching tree before stopping the algorithm. If the algorithm returns a feasible packing, then the route is feasible. If the algorithm ends without finding any feasible packing, then the route is infeasible. Otherwise, we have to move to the next step.

4. The exact algorithm reported in [50] for solving the 2OPP-UL is finally applied. It is based on a mathematical formulation of the 1CBP to which constraints are added to satisfy the unloading requirements. In practice, this algorithm proved to be very good at detecting infeasibility in short computation times.

At the end, we know if the packing problem is feasible or infeasible and, if feasible, we have the corresponding solution.

5.8 Computational Results

In this section, we first compare our method over a set of existing instances proposed in [89] for the deterministic 2L-CVRP. Next, we explain how we generated new instances for the S2L-CVRP, before analyzing the contribution of the previously proposed inequalities on those instances. The final results are reported at the end.

Our L-Shaped Method was coded in C++ and called the CPLEX 12.5 solver. The tests were performed on a 3.07 Ghz Intel Xeon X5675 running under the Linux system. Note that the number of variables M used for generating L-cuts was set to the number of customers with stochastic items. Preliminary tests showed that larger values do not provide any benefit.

5.8.1 Comparison on the 2L-CVRP

By setting the number of possible values for the size and weight of each item to 1, deterministic instances are obtained. In this case, our L-Shaped Method reduces to a branch-and-cut algorithm which can be compared to the one reported by Iori et al. [89]. The algorithm of Iori et al. was coded in C and was run on a 3GHz Pentium IV with the CPLEX 9.0 solver. The packing problems were solved with a branch-and-bound algorithm based on the work in [116] for the Strip Packing Problem. This algorithm was run for 86,400 seconds on each instance, as compared to 7,200 seconds for ours.

To test their algorithm, the authors in [89] created five different types of instances from an original set of 36 instances, for a total of 180 instances. In all cases, the loading area of each vehicle has height $H = 40$ and width $W = 20$. In the first type of instances, every customer has a single item of width and height equal to 1. Since the packing is not constraining, these instances reduce to a classical vehicle routing problem with one-dimensional or scalar demand (weight). With regard to the other instances, each customer has 1 or 2 items in type 2, 1 to 3 items in type 3, 1 to 4 items in type 4 and 1 to 5 items in type 5. Furthermore, each item can have one of three different shapes, namely Vertical, Horizontal or Homogeneous. The exact number of items per customer and the shape of each item were randomly generated in the intervals shown in Table 5.I. The largest instances have up to 255 customers, 786 items and a fleet of 51 vehicles.

The results are shown in Table 5.II under the headings "*Iori et al.*" and "*Our B&C*". In each case, we indicate the number of instances of each type solved by both algorithms,

	# Items	Vertical		Horizontal		Homogeneous	
Type	per cust.	Height	Width	Height	Width	Height	Width
1	1	1	1	1	1	1	1
2	[1,2]	[.4H, .9H]	[.1W, .2W]	[.1H, .2H]	[.4W, .9W]	[.2H, .5H]	[.2W, .5W]
3	[1,3]	[.3H, .8H]	[.1W, .2W]	[.1H, .2H]	[.3W, .8W]	[.2H, .4H]	[.2W, .4W]
4	[1,4]	[.2H, .7H]	[.1W, .2W]	[.1H, .2H]	[.2W, .7W]	[.1H, .4H]	[.1W, .4W]
5	[1,5]	[.1H, .6H]	[.1W, .2W]	[.1H, .2H]	[.1W, .6W]	[.1H, .3H]	[.1W, .3W]

Tableau 5.I – Types of instances

as well as the average CPU time in seconds. The number of additional instances that were solved by our algorithm when compared to the algorithm of Iori et al., as well as the average CPU time in seconds for solving these instances, is also indicated. A total of 26 additional instances were solved by our algorithm

Overall, our algorithm was able to solve instances with up to 71 customers and 226 items while the algorithm of Iori et al. was limited to a maximum of 35 customers and 114 items. For the 55 instances solved by both algorithms, ours took only a few seconds as compared to hundreds or even thousands of seconds for the other algorithm. This is a very substantial improvement, even if we take into account the different specifications of the two machines used to run the algorithms. Note that this improvement is mostly explained by the use of sophisticated packing algorithms.

5.8.2 Stochastic instances

The same 2L-CVRP instances described in the previous section were used to generate our stochastic instances. However, given that the packing problems do not have any impact when solving the instances of type 1, they are not considered anymore in the following. We took all instances of types 2, 3, 4 and 5 with at most 71 customers (the largest number of customers that our algorithm can address), for a total of 20 instances of each type. From each one of these $4 \cdot 20 = 80$ instances, we generated six different stochastic instances, for a total of 480 instances, by varying the percentage of customers with stochastic items and the maximum size of the discrete domain for the height, width and weight of each stochastic item, as shown in Table 5.III. Note that when the domain can take up to x different values, each stochastic item has between 2 and x dif-

	Iori et al.		*Our B&C*			
Type	Solved	Time	Solved	Time	New Solved	Time
1	12	4731.9	12	2.1	7	49.8
2	11	1123.6	11	9.8	2	2314.4
3	12	1332.0	12	6.4	3	385.2
4	10	1030.9	10	11.7	5	171.2
5	10	488.8	10	1.8	9	633.6
Avg.		1741.4		6.4		710.8
Sum	55		55		26	

Tableau 5.II – Comparison of two algorithms for the 2L-CVRP

ferent height, width and weight values, with a given probability distribution defined over these values. The width and height values for each item were selected in the intervals $[\max\{1, h/2\}, \min\{h + h/2, H\}]$ and $[\max\{1, w/2\}, \min\{w + w/2, W\}]$, where h and w are the height and width of the item in the original deterministic instance. All real values were rounded to get only integers. The c_f parameter which is used to compute the recourse cost in equations (5.7) and (5.8) was set to 10.

We observed that our algorithm is very sensitive to the number of items per route, due to the difficulty of the packing problem. To get an increasing number of items per route from the instances of type 2 to the instances of type 5, without exceeding the computational limits of our algorithm, an average of 4 customers per route was allowed through the definition of appropriate weights (leading to an increasing average number of items per route from type 2 to type 5) . Basically, a weight was generated for each customer based on a normal law of mean $Q/4$, where Q is the vehicle capacity in the original 2L-CVRP instances. The weight obtained was then split randomly among the items of the corresponding customer. Some final adjustments were performed to guarantee that at least two customers could fit in a route.

The number of vehicles and an upper bound on the objective value were obtained with the adaptive large neighborhood search heuristic (ALNS) in [142]. The maximum number of iterations was set to 25,000 and a time limit of 1,000 seconds was imposed.

5.8.3 Impact of the various inequalities

We first report some results aimed at evaluating the effectiveness of the proposed inequalities. We created 5 different algorithmic variants for this purpose. The first setting “All cuts” correspond to the L-Shaped method described in Section 5.3, including D-

% Stoch. cust.	Domain size
10	2
50	2
100	2
100	3
50	5
10	9

Tableau 5.III – S2L-CVRP instances

optimality cuts for integer solutions and *L*-cuts for integer and fractional solutions. The application of *L*-cuts on fractional solutions was removed in the four other variants. So, “No Frac. *L*-cuts” is the original L-Shaped method minus *L*-cuts on fractional solutions, “No *L*-Cuts” removes all *L*-cuts on both fractional and integer solutions, “No D-cuts” uses *L*-cuts on integer solutions and replaces the *D*-optimality cuts by the global optimality cuts (5.38) and “No L-D-cuts” removes all *L*-cuts and replaces the *D*-optimality cuts by the global optimality cuts. In all cases, the resulting algorithm was run for a maximum of 1,200 seconds.

The results are summarized in Table 5.IV on the $6 \cdot 20 = 120$ stochastic instances of each type. The table shows the number of solved instances and the following averages : CPU time in seconds, CPU time in seconds for solving the packing problems, gap in percentage between the final solution and the upper bound from the ALNS heuristic, number of *L*-cuts, number of infeasible set constraints, number of infeasible path constraints, number of *D*-optimality cuts and number of VRP (integer) solutions. To allow a fair comparison, the averages for the various inequalities were calculated only over the instances solved by all variants. Similarly, the average gap was taken only over the instances that were not solved by any variant. The number of solved-by-all and not-solved-by-any instances is indicated in Table 5.IV for each type.

The first observation is about the *L*-cuts on fractional solutions, which do not appear to be useful when “All cuts” is compared with “No Frac. *L*-cuts”. In particular, the number of solved instances slightly increases and the computation time decreases when they are removed. So, it appears that many of these cuts do not provide useful bounds on the objective.

The most useful inequalities are the *L*-cuts on integer solutions. When these cuts are present, the *D*-optimality cuts seem almost useless by comparing the results of “No Frac. *L*-cuts” and “No *D*-cuts”. However, when the *L*-cuts are not present, the *D*-optimality cuts play a useful role, as indicated by the poor performance of “No *L*-*D*-cuts”. Overall, “No Frac. *L*-cuts” is the best approach with regard to the number of solved instances and CPU time. Accordingly, this variant was used for the results reported in the next section.

Type 2 Variant	Solved	Total CPU	Packing CPU	Gap (%)	*L*-cuts	Inf. Set	Inf. Path	*D*-cuts	VRP
All Cuts	67	65.2	7.2	7.10%	269.3	300.8	12.4	21.0	27.9
No Frac L-cuts	70	43.7	8.2	7.22%	21.3	14.7	12.5	11.3	36.2
No L-Cuts	67	64.6	13.8	7.54%	0.0	10.4	29.1	24.6	37.4
No D-Cuts	69	61.4	7.2	7.24%	23.5	18.6	5.9	21.0	50.7
No L-D-Cuts	54	189.3	61.2	7.67%	0.0	12.4	36.4	631.7	651.8
Solved-by-all	54								
Not-solved-by-any	50								
Type 3									
All Cuts	76	69.8	15.8	7.95%	164.7	162.7	7.1	19.4	18.5
No Frac L-cuts	78	38.9	14.0	7.97%	17.0	2.9	7.8	10.0	25.4
No L-Cuts	75	66.0	30.3	8.35%	0.0	2.0	13.8	20.8	24.2
No D-Cuts	78	43.3	15.8	8.01%	17.9	3.1	4.1	19.4	36.1
No L-D-Cuts	65	139.9	79.4	8.50%	0.0	2.4	19.4	396.3	404.2
Solved-by-all	65								
Not-solved-by-any	41								
Type 4									
All Cuts	73	84.3	23.8	9.53%	118.8	55.4	4.0	12.6	13.2
No Frac L-cuts	77	46.6	21.0	9.46%	11.5	2.2	3.5	7.2	16.2
No L-Cuts	72	83.9	53.4	9.83%	0.0	1.8	6.0	15.3	15.9
No D-Cuts	75	46.6	23.8	9.48%	12.5	2.4	1.3	12.6	24.6
No L-D-Cuts	62	182.4	127.6	10.02%	0.0	2.4	6.9	291.9	298.0
Solved-by-all	62								
Not-solved-by-any	43								
Type 5									
All Cuts	80	25.2	18.4	5.50%	30.7	1.3	0.1	2.8	4.7
No Frac L-cuts	81	23.5	18.4	5.71%	6.3	0.1	0.2	2.5	6.7
No L-Cuts	65	73.6	68.8	6.17%	0.0	0.1	0.4	9.6	8.8
No D-Cuts	82	22.9	18.4	5.71%	6.4	0.1	0.0	2.8	7.9
No L-D-Cuts	54	156.8	146.8	6.54%	0.0	0.1	0.0	145.2	147.5
Solved-by-all	50								
Not-solved-by-any	36								
Overall									
All Cuts	296	63.0	16.5	7.58%	147.8	131.2	6.0	14.4	16.3
No Frac L-cuts	306	38.8	15.5	7.65%	14.2	4.9	6.1	7.9	21.4
No L-Cuts	279	72.1	41.0	8.02%	0.0	3.5	12.4	17.8	21.7
No D-Cuts	304	44.0	16.5	7.67%	15.3	5.9	2.9	14.4	30.3
No L-D-Cuts	235	166.5	102.6	8.23%	0.0	4.2	15.8	369.0	378.0
Solved-by-all	231								
Not-solved-by-any	170								

Tableau 5.IV – Comparison of different variants

5.8.4 Final results

The final results are reported in Tables 5.V and 5.VI after running our algorithm for a maximum of 7,200 seconds (2 hours) on each instance. Table 5.V summarizes the average results obtained over each set of 80 instances associated with a given percentage of customers with stochastic items and domain size. Otherwise, the format of this table is similar to Table 5.IV. Table 5.V shows in particular that a higher percentage of customers with stochastic items increases the complexity of the problem, as indicated by the number of solved instances and CPU time, in particular when going from 10% to 50%. Also, the total CPU time sharply increases when the domain of the probability distribution increases. For example, when the number of customers with stochastic items is low (10%), increasing the number of values in the domain of the probability distribution from 2 to 9 increases the total CPU time from 90.5 seconds to 337.4 seconds and the time for solving the packing problems from 22.4 seconds to 109.1 seconds. We also observed that the total CPU time on some of these instances was almost totally spent on the packing problems.

Table 5.VI shows another, more detailed, view of the results. Each identifier in this table is a 4-digit number : the first two digits identify the instance number from the 36 original instances in [89], while the last two digits identify the instance type. For example, identifiers 0202 to 0205 correspond to the deterministic instances of types 2 to 5 derived from the second original instance. As previously mentioned, six different stochastic instances were generated from the deterministic instance associated with the 4-digit identifier. In Table 5.VI, heading “Ins” is the 4-digit identifier, n is the number of customers, “Solved” is the number of instances solved to optimality, “Total CPU” is the average computation time in seconds spent over the instances solved to optimality and “Packing CPU” is the average computation time spent on the packing problems over the instances solved to optimality. The two last headings are “Gap”, which contains the average gap in percentage between the final solution and the heuristic solution over all instances that were not solved to optimality, and “Cuts” which is the average number of L-cuts, infeasible set inequalities, infeasible path inequalities and D-cuts that were added to the model over all instances solved to optimality.

These detailed results show in particular the limitations of our algorithm with regard to the problem size. Most instances from 15 to 32 customers can be solved within the time limit, but difficulties arise beyond 32 customers. In particular, the number of possible sets of customers S, when calculating $L(S)$, increases sharply. Overall, our method was able to solve 332 instances out of 480 using an average of 353.8 seconds of computation time.

5.9 Conclusion

This paper has introduced a stochastic variant of the 2L-CVRP where some item sizes are not known with certainty when the vehicle routes are planned. From a methodological standpoint, a new type of Lower Bounding Functionals, called L-cuts, has been introduced. The latter proved to be very effective when integrated within the reported L-Shaped Method. On the deterministic 2L-CVRP, the branch-and-cut algorithm derived from our L-Shaped Method also outperformed another state-of-the-art exact algorithm on a set of benchmark instances. Future work will consider different variants where, for example, it is possible to rotate items to better fill the loading area. Also, we want to address an extension with both pickups and deliveries along the routes. In practice, this type of problem occurs when an item must be exchanged for another at a customer location (due to some defect).

Domain size	% Stoch. cust.	Solved	Total CPU	Packing CPU	Gap (%)	L-cuts	Inf. Set	Inf. Path	D-cuts	VRP
2	10	59	90.5	22.4	6.4	6.9	9.8	9.8	3.5	22.6
2	50	53	254.1	32.6	7.8	32.4	4.6	7.7	14.6	34.8
2	100	55	564.1	340.8	7.1	41.4	0.3	1.6	18.7	30.9
3	100	50	491.2	311.2	6.8	41.2	0.0	0.7	17.6	27.7
5	50	53	385.7	90.2	8.6	39.5	4.6	6.4	18.5	37.8
9	10	62	337.4	109.1	7.6	17.0	10.9	12.4	8.4	33.0
Avg.			353.8	151.1	7.4	29.7	5.0	6.4	13.6	31.1
Sum		332				178.4	30.2	38.6	81.3	186.8

Tableau 5.V – Summary of final results

Ins	n	Solved	Total CPU	Packing CPU	Gap (%)	Cuts	Ins	n	Solved	Total CPU	Packing CPU	Gap (%)	Cuts
0102	15	6	0.5	0.5	-	2.5	1102	29	6	996.3	28.8	-	195.5
0103	15	6	2.6	2.6	-	10.8	1103	29	4	888.1	112.3	6.3	150.8
0104	15	6	3.0	2.9	-	9.5	1104	29	5	362.6	164.6	4.5	166.2
0105	15	6	11.1	11.1	-	2.3	1105	29	5	242.3	160.1	1.9	33.8
0202	15	6	1.4	1.3	-	19.3	1202	30	4	881.5	41.5	9.7	270.0
0203	15	6	4.0	4.0	-	27.2	1203	30	6	110.7	27.8	-	83.0
0204	15	6	3.2	3.2	-	3.0	1204	30	5	1262.7	341.8	9.3	92.0
0205	15	6	6.6	6.6	-	3.2	1205	30	6	1846.6	1821.6	-	46.0
0302	20	5	22.6	9.6	2.8	92.4	1302	32	4	429.2	9.1	1.6	70.8
0303	20	6	22.6	22.1	-	47.2	1303	32	6	62.0	15.3	-	72.7
0304	20	6	27.0	23.6	-	27.0	1304	32	6	138.3	39.5	-	41.0
0305	20	6	8.6	7.6	-	11.3	1305	32	6	861.0	855.4	-	19.8
0402	20	6	75.0	13.1	-	80.7	1402	32	1	4864.8	4.6	1.9	240.0
0403	20	6	10.8	8.0	-	22.2	1403	32	4	845.2	7.6	1.0	67.3
0404	20	6	35.7	35.6	-	23.3	1404	32	5	2112.3	50.5	0.5	98.2
0405	20	5	1156.1	1154.3	3.2	11.6	1405	32	4	398.4	35.2	1.0	58.8
0502	21	6	17.1	8.0	-	30.3	1502	32	1	277.8	1.4	1.7	49.0
0503	21	6	9.2	4.6	-	18.7	1503	32	0	-	-	2.9	-
0504	21	6	17.8	13.8	-	28.8	1504	32	0	-	-	3.8	-
0505	21	6	21.0	20.5	-	15.0	1505	32	1	6350.3	5222.1	5.1	27.0
0602	21	6	93.6	15.8	-	76.2	1602	35	1	1683.0	39.4	5.3	258.0
0603	21	6	78.0	28.5	-	43.7	1603	35	1	4380.6	79.2	6.9	210.0
0604	21	6	119.0	117.6	-	32.2	1604	35	2	656.1	36.9	9.0	124.0
0605	21	6	14.7	13.9	-	15.7	1605	35	6	266.4	28.6	-	40.2
0702	22	6	202.5	34.1	-	85.5	1702	40	0	-	-	7.9	-
0703	22	6	47.3	43.0	-	38.3	1703	40	2	736.4	1.7	6.7	27.5
0704	22	6	32.6	32.2	-	11.5	1704	40	0	-	-	7.9	-
0705	22	6	31.5	31.0	-	12.5	1705	40	3	649.4	77.3	4.8	25.3
0802	22	6	4.6	4.3	-	22.0	1802	44	0	-	-	4.1	-
0803	22	6	31.4	30.4	-	34.3	1803	44	0	-	-	4.8	-
0804	22	6	21.4	21.2	-	12.7	1804	44	0	-	-	8.4	-
0805	22	6	46.3	45.6	-	13.3	1805	44	2	43.2	21.0	2.1	23.0
0902	25	5	106.6	20.9	11.7	187.2	1902	50	0	-	-	11.2	-
0903	25	6	352.7	307.8	-	69.5	1903	50	0	-	-	11.9	-
0904	25	6	189.1	151.8	-	63.7	1904	50	0	-	-	14.6	-
0905	25	5	1041.6	1041.0	5.0	10.2	1905	50	0	-	-	5.2	-
1002	29	6	127.8	32.3	-	190.0	2002	71	0	-	-	14.2	-
1003	29	6	100.4	11.0	-	65.0	2003	71	0	-	-	15.0	-
1004	29	6	1434.3	178.1	-	101.2	2004	71	0	-	-	15.3	-
1005	29	6	526.4	508.3	-	38.8	2005	71	0	-	-	10.4	-

Tableau 5.VI – A more detailed view of final results

CHAPITRE 6

CONCLUSION

Cette thèse de doctorat, constituée de trois articles, présente des algorithmes pour la résolution de problèmes de tournées de véhicules avec contraintes de chargement. Dans chacun des trois articles, nous avons proposé des méthodes exactes exploitant des formulations mathématiques. Nous résumons ici les principales contributions de cette thèse ainsi que les avenues de recherche future.

6.1 Principales contributions

Le chapitre 2 présente d'abord une revue de la littérature portant sur les problèmes de tournées avec contraintes de chargement. Nous avons répertorié de nombreuses variantes et décrit les principales méthodes de résolution. À la fin de ce chapitre, nous avons abordé les contributions les plus fondamentales pour les problèmes de tournées de véhicules de nature stochastique.

Le chapitre 3 décrit une méthode de résolution pour le problème de confection d'une seule tournée avec cueillettes et livraisons où l'aire de chargement du véhicule est divisée en un certain nombre de piles. Une contrainte de type "premier entré, dernier sorti" est imposée sur la séquence des cueillettes et des livraisons. Notre principale contribution se situe au niveau de l'introduction de nouvelles inégalités valides qui permettent de résoudre ce problème plus efficacement. De plus, le problème généralise certaines problématiques déjà rapportées dans la littérature. Nos résultats, tant sur notre problème que sur des des cas particuliers, démontrent que notre approche est plus efficace et permet de résoudre des instances de plus grande taille.

Le chapitre 4 aborde un problème de réalisabilité (ou décidabilité). En l'occurrence, il s'agit de déterminer si un ensemble d'items en deux dimensions peut être contenu à l'intérieur de l'espace de chargement d'un véhicule. Un des côtés du véhicule est muni d'une porte permettant de charger et de décharger les items. Une contrainte exige que chaque item soit directement accessible au moment de sa livraison, sans qu'il soit nécessaire de déplacer d'autres items dans le véhicule. Cette problématique apparaît souvent comme un sous-problème du problème de tournées de véhicules avec chargement d'items en deux dimensions. Dans ce travail, nous nous sommes intéressés à la résolution du problème de chargement à l'aide d'une méthode exacte qui s'est révélée très efficace en permettant de résoudre rapidement de nombreuses instances tests. Enfin, nous présentons aussi dans ce travail différentes procédures de prétraitement et tests de réalisabilité.

Dans le chapitre 5, nous introduisons une nouvelle problématique souvent rencontrée en pratique. Plus précisément, les dimensions de certains items ne sont pas connues avec certitude lors de la planification des tournées. Il est toutefois possible de définir une distribution de probabilités discrète sur les dimensions possibles d'un item. Ce travail présente un algorithme exact qui repose sur la méthode L-shaped en nombres entiers. Nous introduisons plusieurs bornes inférieures pour notre problème qui sont également valides pour d'autres problèmes de tournées de nature stochastique. Enfin, en plus de démontrer l'utilité de nos bornes, nous démontrons le fort potentiel de notre méthode sur des instances tests.

6.2 Avenues de recherche

Bien qu'il existe déjà une assez vaste littérature sur les problèmes de tournées avec contraintes de chargement, nous croyons que les avenues de recherche future sont multiples.

Une première idée serait d'étendre les résultats rapportés dans le deuxième article, qui porte sur le problème de chargement en deux dimensions, aux problèmes en trois dimensions. Pour ces derniers, une foule de contraintes supplémentaires peuvent être considérées : fragilité, aire de support minimale, espaces vides prohibés, etc. De telles contraintes pourraient ainsi être intégrées aux procédures que nous avons développées de façon à résoudre plus efficacement les problèmes en trois dimensions.

Une seconde idée serait d'autoriser un plus vaste éventail de configurations possibles pour le chargement des items. Dans notre premier article, par exemple, les items doivent pouvoir être déplacés en ligne droite quand on veut les sortir du véhicule. En acceptant le déplacement latéral d'un item, de nouvelles configurations deviendraient alors possibles, permettant du même coup de mieux remplir l'espace de chargement.

Nous pourrions également considérer des problèmes de chargement où les tournées sont constituées à la fois de cueillettes et livraisons d'items en deux dimensions. Ceci survient par exemple lorsqu'une compagnie de livraison doit procéder à des échanges. Plus précisément, il arrive qu'un client ayant déjà reçu un premier item demande à échanger pour un motif quelconque : bris, insatisfaction, item différent de celui demandé, etc. Il faut alors qu'un autre item provenant du centre de distribution soit livré au client tandis que l'item à remplacer est chargé dans le véhicule. Dans plusieurs cas, l'item ainsi recueilli doit être retourné à un magasin particulier. Un autre exemple est observé dans certaines compagnies qui offrent des services de cueillette de vieux appareils.

BIBLIOGRAPHIE

[1] Agbegha G.Y., Ballou R.H., Mathur K., Optimizing auto-carrier loading. Transportation Science 32, 174-188, 1998.

[2] Alba M., Cordeau J.-F., Dell'Amico M., Iori M., A branch-and-cut algorithm for the double traveling salesman problem with multiple stacks. INFORMS Journal on Computing 25, 41-55, 2013.

[3] Alvarez-Valdes R., Parreño F., Tamarit J.M., A GRASP algorithm for constrained two-dimensional non guillotine cutting problems. Journal of the Operational Research Society 56, 414-425, 2005.

[4] Alvarez-Valdes R., Parreño F., Tamarit J. M., A branch and bound algorithm for the strip packing problem. OR Spectrum 31 431-459, 2009.

[5] Amaral A., Letchford A., An improved upper bound for the two-dimensional non-guillotine cutting problem. Working paper, Lancaster University, UK, 2003.

[6] Arahori Y.,Imamichi T., Nagamochi H., An exact strip packing algorithm based on canonical forms. Computers & Operations Research 39, 2991-3011, 2012.

[7] Archetti, C., Hertz, A., Speranza, M.G., A tabu search algorithm for the split delivery vehicle routing problem. Transportation Science 40, 64-73,2006.

[8] Balas E., Fischetti M., Pulleyblank W.R., The precedence-constrained asymmetric traveling salesman polytope. Mathematical Programming 68, 241-265, 1995.

[9] Baker B. S., Coffman Jr. E. G., Rivest R. L., Orthogonal packing in two dimensions. SIAM Journal on Computing 9, 846-855, 1980.

[10] Baldacci R., Boschetti M.A., A cutting-plane approach for the two-dimensional orthogonal non-guillotine cutting problem. European Journal of Operational Research 183, 1136-1149, 2007.

[11] Battarra M., Erdoğan G., Laporte G., Vigo D., The traveling salesman problem with pickups. deliveries and handling costs. Transportation Science 44, 383-399, 2010.

[12] Beasley J. E., An exact two-dimensional non-guillotine cutting tree search procedure. Operations Research 33, 49-64, 1985.

[13] Belov G., Rohling H., LP bounds in an interval-graph algorithm for orthogonal-packing Feasibility. Operations Research 61, 483-497, 2013.

[14] Ben Messaoud S. Caractérisation, modélisation et algorithmes pour des problèmes de découpe guillotine. Thèse de doctorat, Université de technologie de Troyes, France, 2004.

[15] Berkey J.O., Wang P.Y., Two-dimensional finite bin packing algorithms. Journal of the Operational Research Society 38, 423-429, 1987.

[16] Bertsimas D. J., Jaillet P., Odoni A. R., A priori optimization. Operations Research 38, 1019-1033, 1990.

[17] Bonomo F., Mattia S. and Oriolo G., Bounded coloring of co-comparability graphs and the pickup and delivery tour combination problem. Theoretical Computer Science 412, 6261-6268, 2011.

[18] Bontoux B., Techniques hybrides de recherche exacte et approchée : Application à des problèmes de transport. Thèse de Doctorat, Université d'Avignon et des Pays de Vaucluse, 2008.

[19] Bortfeldt A., Wäscher G., Container loading problems : A state-of-the-art review. European Journal of Operational Research 229, 1-20, 2013.

[20] Bortfeldt A., Mack D., A heuristic for the three-dimensional strip packing problem. European Journal of Operational Research 183, 1267-1279, 2007.

[21] Bortfeldt A., A hybrid algorithm for the capacitated vehicle routing problem with three-dimensional loading constraints. Computers & Operations Research 39, 2248-2257, 2013.

[22] Boschetti M.A., Montaletti L., An exact algorithm for the two-dimensional strip-packing problem, Operations Research 58, 1774-1791, 2010.

[23] Burke EK, Kendall G, Whitwell G. A new placement heuristic for the orthogonal stock-cutting problem. Operations Research 52, 655-671, 2004.

[24] Caprara A., Monaci M., On the two-dimensional knapsack problem. Operations Research Letters 32, 5-14, 2004.

[25] Caprara A., Monaci M., Bidimensional packing by bilinear programming. Mathematical Programming 118, 75-108, 2009.

[26] Carlier J., Clautiaux F., Moukrim A., New reduction procedures and lower bounds for the two-dimensional bin packing problem with fixed orientation. Computers & Operations Research 34, 2223-2250, 2007.

[27] Carrabs F., Cerulli, R., Cordeau, J.-F., An additive branch-and-bound algorithm for the pickup and delivery traveling salesman problem with LIFO or FIFO loading. INFOR 45, 223-238, 2007.

[28] Carrabs F., Cerulli R., Speranza M.G., A branch-and-bound algorithm for the double TSP with two stacks. Networks 61, 58-75, 2013.

[29] Carrabs F., Cordeau, J.-F., Laporte, G., Variable neighborhood search for the pickup and delivery traveling salesman problem with LIFO loading. INFORMS Journal on Computing 19, 618-632, 2007.

[30] Ceschia S., Schaerf A., Local search for a multi-drop multi-container loading problem. Journal of Heuristics 19, 275-294, 2012.

[31] Casazza M., Ceselli A., Nunkesser M., Efficient algorithms for the double traveling salesman problem with multiple stacks. Computers & Operations Research 39, 1044-1053, 2012.

[32] Chazelle B. The bottom-left bin packing heuristic : An efficient implementation. IEEE Transactions on Computers C-32, 697-707, 1983.

[33] Christiansen C. H., Lysgaard J., A branch-and-price algorithm for the capacitated vehicle routing problem with stochastic demands. Operations Research Letters 35, 773-781, 2007.

[34] Christofides N., Mingozzi A., Toth P., The Vehicle Routing Problem, Christofides N., Mingozzi A., Toth P., Sandi C., eds. Combinatorial optimization. Wiley, New York, 315-338, 1979.

[35] Christofides N., Whitlock C., An algorithm for two-dimensional cutting problems. Operations Research 25, 30-44, 1977.

[36] Clarke G., Wright J.W., Scheduling of vehicles from a central depot to a number of delivery points. Operations Research 12, 568-581, 1964.

[37] Clautiaux F., Alves C., de Carvalho J.V., A survey of dual-feasible and superadditive functions, Annals of Operations Research 179, 317-342, 2010.

[38] Clautiaux F., Carlier J., Moukrim A., A new exact method for the two-dimensional orthogonal packing problem. European Journal of Operational Research 183, 1196-1211, 2007.

[39] Codato G., Fischetti M., Combinatorial benders cuts for mixed-integer linear programming. Operations Research 54, 756-766, 2006.

[40] Coffman E.G. Jr., Galambos. G., Martello S., Vigo D. Bin packing approximation algorithms : Combinatorial analysis, Dans : Du D.-Z., Pardalos P.M. (eds) Handbook of Combinatorial Optimization. Kluwer, Dordrecht, 151-208, 1999.

[41] Coffman E.G. Jr., Garey M.R., Johnson D.S., Approximation algorithms for bin packing : A survey. Dans Approximation Algorithms for NP-Hard Problems, PWS Publishing co, Boston, 46-93, 1997.

[42] Cordeau J.-F, A branch-and-cut algorithm for the dial-a-ride problem, Operations Research 54, 573-586, 2006.

[43] Cordeau J.-F., Dell'Amico M., Iori M., Branch-and-cut for the pickup and delivery traveling salesman problem with FIFO loading. Computers & Operations Research 37, 970-980, 2010.

[44] Cordeau J.-F., Desaulniers G., Desrosiers J., Solomon M.M., Soumis F., The VRP with time windows. Dans : Toth P., Vigo D., The Vehicle Routing Problem. SIAM Monographs on Discrete Mathematics and Applications, SIAM, Philadelphia, 157-194, 2001.

[45] Cordeau J.-F., Iori M., Laporte G., Salazar-González J.-J., A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with LIFO loading, Networks 55, 46-59, 2010.

[46] Cordeau J.-F., Iori M., Ropke S., Vigo D., Branch-and-cut-and-price for the capacitated vehicle routing problem with two-dimensional loading constraints. Présenté à ROUTE 2007, Jekyll Island, U.S.A., May 2007.

[47] Cordeau J.-F., Laporte G., Mercier A., A unified tabu search heuristic for vehicle routing problems with time windows. Journal of the Operational Research Society 52, 928-936, 2001.

[48] Côté J.-F., Une heuristique à grand voisinage pour un problème de confection de tournée pour un seul véhicule avec cueillettes et livraisons et contrainte de chargement. Mémoire de maîtrise, Université de Montréal, 2010.

[49] Côté, J.-F., Gendreau M., Potvin J.-Y., Large neighborhood search for the single vehicle pickup and delivery problem with multiple loading stacks. Networks 60, 19-30, 2012.

[50] Côté J.-F., Gendreau M., Potvin J.-Y., An exact algorithm for the two-dimensional orthogonal packing problem with unloading constraints. Rapport technique, CIRRELT-2013-26, 2013.

[51] Côté J.-F., Iori M., Dell'Amico M., Combinatorial Benders' cuts for the strip packing problem. Forthcoming in Operations Research, 2013.

[52] Crainic T. G., Perboli G., Tadei R. Extreme point-based heuristics for three-dimensional bin packing. INFORMS Journal on Computing 20, 368-384, 2008.

[53] Croes, G., A method for solving traveling salesman problems. Operations Research 6, 791-812, 1958.

[54] Dantzig G.B., Ramser J.H., The truck dispatching problem. Management Science 6, 80-91, 1959.

[55] Dell'Amico M., Falavigna S., Iori M., Optimization of a Real-World Auto-Carrier Transportation Problem. Forthcoming in Transportation Science.

[56] Dell'Amico M., Martello S., Optimal scheduling of tasks on identical parallel processors. ORSA Journal on Computing 7, 191-200, 1995.

[57] Dell'Amico M., Martello S., A lower bound for the non-oriented two-dimensional bin packing problem. Discrete Applied Mathematics 118, 13-24, 2002.

[58] Desaulniers G., Desrosiers J., Solomon M.M., Column Generation, Springer, 2005.

[59] Doerner K.F., Fuellerer G., Hartl R.F., Gronalt M., Iori M., Metaheuristics for the vehicle routing problem with loading constraints. Networks 49, 294-307, 2007.

[60] Dror M., Laporte G., Trudeau P., Vehicle routing with stochastic demands : Properties and solution frameworks. Transportation Science 23, 166-176, 1989.

[61] Dueck G., Scheuer T., Threshold Accepting : A general purpose optimization algorithm appearing superior to simulated annealing. Journal of Computational Physics 90, 161-175, 1990.

[62] Duhamel C., Lacomme P., Quilliot A., Toussaint H., A multi-start evolutionary local search for the two-dimensional loading capacitated vehicle routing problem, Computers & Operations Research 38, 617-640, 2011.

[63] Dumitrescu I., Ropke S., Cordeau J.-F., Laporte G., The traveling salesman problem with pickup and delivery : Polyhedral results and a branch-and-cut algorithm. Mathematical Programming 121, 269-305, 2010.

[64] Erdoğan G., Cordeau J.-F., Laporte G., The pickup and delivery traveling salesman problem with first-in-first-out loading, Computers & Operations Research 36, 1800-1808, 2009.

[65] Fanslau T., Bortfeldt A. A tree search algorithm for solving the container loading problem. INFORMS Journal on Computing 22, 222-235, 2010.

[66] Fekete S., Schepers J., A combinatorial characterization of higher-dimensional orthogonal packing. Mathematics of Operations Research 29, 353-368, 2004.

[67] Fekete S., Schepers J., van der Veen J.C., An exact algorithm for higher-dimensional orthogonal packing. Operations Research 55, 569-587, 2007.

[68] Felipe A., Ortuño M.T., Tirado G., The double traveling salesman problem with multiple stacks : A variable neighborhood search approach. Computers & Operations Research 36, 2983-2993, 2009.

[69] Felipe A., Ortuño M.T., Tirado G., Using intermediate infeasible solutions to approach vehicle routing problems with precedence and loading constraints. European Journal of Operational Research 211, 66-75, 2011.

[70] Fischetti M., Toth P., An additive bounding procedure for combinatorial optimization problems. Operations Research 37, 319-328, 1989.

[71] Fuellerer G., Doerner K.F., Hartl R.F., Iori M., Ant colony optimization for the two-dimensional loading vehicle routing problem. Computers & Operations Research 36, 655-673, 2009.

[72] Fuellerer G., Doerner K.F., Hartl R.F., Iori M., Metaheuristics for vehicle routing problems with three-dimensional loading constraints. European Journal of Operational Research 201, 751-759, 2010.

[73] Gendreau M., Hertz A., Laporte G., New insertion and postoptimization procedures for the traveling salesman problem. Operations Research 40, 1086-1094, 1992.

[74] Gendreau M., Hertz A., Laporte G., A tabu search heuristic for the vehicle routing problem. Management Science 40, 1276-1290, 1994.

[75] Gendreau M., Iori M., Laporte G., Martello S., A tabu search algorithm for a routing and container loading problem. Transportation Science 40, 342-350, 2006.

[76] Gendreau M., Iori M., Laporte G., Martello S., A tabu search heuristic for the vehicle routing problem with two-dimensional loading constraints. Networks 51, 4-18, 2008.

[77] Gendreau M., Laporte G., Séguin R., An exact algoritm for the vehicle routing problem with stochastic demands and customers. Transportation Science 29, 143-155, 1995.

[78] Gendreau M., Laporte G., Séguin R., Stochastic vehicle routing. European Journal of Operational Research 88, 3-12, 1996.

[79] George J.A., Robinson D.F., A heuristic for packing boxes into a container. Computers & Operations Research 7, 147-156, 1980.

[80] Glover F., Tabu search : Part 1. ORSA Journal on Computing 1, 190-206, 1989.

[81] Golden B., Raghavan S., Wasil E., The vehicle routing problem : Latest advances and new challenges. Operations research/Computer science interfaces series 43. Springer, Berlin, 2008.

[82] Gu Z., Nemhauser G.L., Savelsbergh M.W.P., Lifted cover inequalities for 0-1 integer programs : Complexity. INFORMS Journal on Computing 11, 117-123, 1999.

[83] Grötschel M., Padberg M.W., Polyhedral theory, Dans : Lawler E., Rynnoy Kan A.H.G., Shmoys D.B. (eds) The traveling salesman problem, 251-305, Wiley, New York, 1985.

[84] Herz J.C., Recursive computational procedure for two-dimensional stock cutting. IBM Journal of Research and Development 16, 462-469, 1972.

[85] Hjorring C., Holt J., New optimality cuts for a single-vehicle stochastic routing problem. Annals of Operations Research 86, 569-584, 1999.

[86] Hochbaum D.S., Shmoys D.B., Using dual approximation algorithms for scheduling problems : Practical and theoretical results. Journal of the Association for Computing Machinery 34, 144-162, 1987.

[87] Iori M., Martello S., Monaci M., Metaheuristic algorithms for the strip packing problem, Dans : Optimization and Industry : New Frontiers, Pardalos P. and Korotkich V. (eds.), Kluwer, Boston, 159-179, 2003.

[88] Iori M., Martello S., Routing problems with loading constraints. TOP 18, 4-27, 2010.

[89] Iori M., Salazar-González J.-J., Vigo D., An exact approach for the vehicle routing problem with two-dimensional loading constraints. Transportation Science 41, 253-264, 2007.

[90] , Jabali O., Rei W., Gendreau M., Laporte G., New valid inequalities for the multi-vehicle routing problem with stochastic demands, Rapport Technique, CIRRELT-2012-58, 2012.

[91] Johnson D. S., Near-optimal bin packing algorithms. Thèse de doctorat, MIT, Cambridge, MA, 1973.

[92] Joncour C., Problèmes de placement 2D et application à l'ordonnancement : Modélisation par la théorie des graphes et approches de programmation mathématique. Thèse de Doctorat, Université de Bordeaux, 2010.

[93] Kenmochi M., Imamichi T., Nonobe K., Yagiura M., Nagamochi H., Exact algorithms for the two-dimensional strip packing problem with and without rotations. European Journal of Operational Research 198, 73-83, 2009.

[94] Khebbache-Hadji S., C. Prins, A. Yalaoui, M. Reghioui, Heuristics and memetic algorithm for the two-dimensional loading capacitated vehicle routing problem with time windows. Central European Journal of Operations Research 21, 307-336, 2013.

[95] Kirkpatrick S., Gelatt C.D., Vecchi M.P., Optimization by simulated annealing. Science 220, 671-680, 1983.

[96] Labadi N., Prins C., Reghioui M., A memetic algorithm for the vehicle routing problem with time windows. RAIRO Operations Research 42, 415-431, 2008.

[97] Lacomme P., Toussaint H., Duhamel C., A GRASP x ELS for the vehicle routing problem with basic three-dimensional loading constraints. Engineering Applications of Artificial Intelligence 26, 1795-1810, 2013.

[98] Ladany S.P., Mehrez A., Optimal routing of a single vehicle with loading and unloading constraints. Transportation Planning and Technology 8, 301-306, 1984.

[99] Laporte G., What you should know about the vehicle routing problem. Naval Research Logistics 54, 811-819, 2007.

[100] Laporte G., Louveaux F.V., The integer L-shaped method for stochastic integer programs with complete recourse. Operations Research Letters 13, 133-142, 1993.

[101] Laporte G., Louveaux F.V., Van Hamme L., An Integer L-Shaped algorithm for the capacitated vehicle routing problem with stochastic demands. Operations Research 50, 415-423, 2002.

[102] Kaparis K., Letchford A.N., Separation algorithms for 0-1 knapsack polytopes. Mathematical Programming 124, 69-91, 2010.

[103] Letchford A., Lodi A., Mathematical programming approaches to the traveling salesman problem. Dans : Cochran J.J. ed., Wiley Encyclopedia of Operations Research and Management Science. Wiley, Chichester, 2011.

[104] Leung S.C.H., Zhang D., Kwang M.S., A two-stage intelligent search algorithm for the two-dimensional strip packing problem. European Journal of Operational Research 215, 57-69, 2011.

[105] Leung S.C.H., Zhang Z., Zhang D., Hua X., Lim M.K., A meta-heuristic algorithm for heterogeneous fleet vehicle routing problems with two-dimensional loading constraints. European Journal of Operational Research 225, 199-210, 2013.

[106] Leung S.C.H., Zhou X., Zhang D., Zheng J., Simulated annealing for the vehicle routing problem with two-dimensional loading constraints. Flexible Services and Manufacturing Journal 22, 21-82, 2010.

[107] Leung S.C.H., Zhou X., Zhang D., Zheng J., Extended guided tabu search and a new packing algorithm for the two-dimensional loading vehicle routing problem, Computers & Operations Research 38, 205-215, 2011.

[108] Li Y., Lim A., Oon W.C., Qin H., Tu D., The tree representation for the pickup and delivery traveling salesman problem with LIFO loading. European Journal of Operational Research 212, 482-496, 2011.

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

[110] Liu D., Teng H., An improved BL-algorithm for genetic algorithm of the orthogonal packing of rectangles. European Journal of Operational Research 112, 413-420, 1999.

[111] Lusby R., Larsen J., Ehrgott M., Ryan D., An exact method for the double TSP with multiple stacks. International Transactions in Operational Research 17, 637-652, 2010.

[112] Lysgaard J., Reachability cuts for the vehicle routing problem with time windows. European Journal of Operational Research 175, 210-223, 2006.

[113] Lysgaard J., Letchford A.N., Eglese R.W., A new branch-and-cut algorithm for the capacitated vehicle routing problem. Mathematical Programming 100, 423-445, 2004.

[114] Ma H., Zhu W., Xu S., Research on the algorithm for 3L-CVRP with considering the utilization rate of vehicles. Dans Proceedings of International Conference Intelligent Computing and Information Science, 621-629, 2011.

[115] Malapert A., Guéret C., Jussien N., Langevin A., Rousseau L.-M., Two-dimensional pickup and delivery routing problem with loading constraints. Rapport Technique 2008-37, CIRRELT-2008-37, 2008.

[116] Martello S., Monaci M., Vigo D., An exact approach to the strip-packing problem. INFORMS Journal on Computing 15, 310-319, 2003.

[117] Martello S., Pisinger D., Vigo D., The three-dimensional bin backing problem. Operations Research 48, 256-267, 2000.

[118] Martello S., Toth P., Knapsack problems : Algorithms and computer implementations. Wiley, Chichester, 1990.

[119] Martello S., Vigo D., Exact solution of the two-dimensional finite bin packing problem. Management Science 44, 388-399, 1998.

[120] Massen F., Deville Y., Van Hentenryck P., A relaxation-guided approach for vehicle routing problems with black box feasibility. Eigth International Workshop on Local Search Techniques in Constraint Satisfaction (LSCS2011). A Satellite Workshop of CP 2011, Italy, September 2011.

[121] Mester D., Bräysy O., Active guided evolution strategies for the large scale vehicle routing problems with time windows. Computers & Operations Research, 32, 1593-1614, 2005.

[122] Mesyagutov M., Scheithauer G., Belov G., LP bounds in various constraint programming approaches for orthogonal packing. Computers & Operations Research 39, 2425-2438, 2012.

[123] Miller C., Tucker A., Zemlin R, Integer programming formulation of traveling salesman problems. Journal of the ACM 7, 326-329, 1960.

[124] Mills P., Tsang E., Ford J., Applying an extended guided local search to the quadratic assignment problem. Annals of Operations Research 118, 121-135, 2003.

[125] Mladenovic N., Hansen P., Variable neighborhood search. Computers & Operations Research 24, 1097-1100, 1997.

[126] Moscato P., Memetic Algorithms : A short introduction. In : Come D., Dorigo M., Glover F. eds. New Ideas in Optimization. McGraw-Hill, New York, 219-234, 1999.

[127] Moura A., Oliveira J.F., A GRASP approach to the container loading problem. IEEE Intelligent Systems 20, 50-57, 2005.

[128] Moura A. Oliveira J.F., An integrated approach to the vehicle routing and container loading problems. OR Spectrum 31, 775-800, 2009.

[129] Osman I.H., Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. Annals of Operations Research 41, 421-452, 1993.

[130] Parreño F., Alvarez-Valdes R., Oliveira J.F., Tamarit J. M., A hybrid GRASP/VND algorithm for two and three-dimensional bin packing. Annals of Operations Research 179, 203-220, 2008.

[131] Petersen H.L., Madsen O.B.G., The double travelling salesman problem with multiple stacks - formulation and heuristic solution approaches. European Journal of Operational Research 198, 139-147, 2009.

[132] Petersen H.L., Archetti C., Speranza M.G., Exact solutions to the double travelling salesman problem with multiple stacks. Networks 56, 229-243, 2010.

[133] Picard J., Queyranne M. The time-dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling. Operations Research 26, 86-110, 1978.

[134] Pisinger D., Ropke S., A general heuristic for vehicle routing problems. Computers & Operations Research 34, 2403-2435, 2007.

[135] Prins C., A GRASP X evolutionary local search hybrid for the vehicle routing problem. Dans : Bio-inspired algorithms for the vehicle routing problem, Studies in computational intelligence, 161, Springer, Berlin, 35-53, 2009.

[136] Prins C., A simple and effective evolutionary algorithm for the vehicle routing problem. Computers & Operations Research 31, 1985-2002, 2004.

[137] Reimann M., Stummer M., Doerner K.F., A savings-based ant system for the vehicle routing problem. Dans : Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kaufmann, 1317-1325, 2002.

[138] Riff M.C., Bonnaire X., Neveu B., A revision of recent approaches for two-dimensional strip-packing problems. Engineering Applications of Artificial Intelligence 198, 823-827, 2009.

[139] Ruland K.S., Rodin E.Y., The pickup and delivery problem : Faces and branch-and-cut algorithm. Computers & Mathematics with Applications 33, 1-13, 1997.

[140] Ropke S., Cordeau J.-F., Branch and cut and price for the pickup and delivery problem with time windows. Transportation Science 43, 267-286, 2009.

[141] Ropke S., Cordeau J.-F., Laporte G., Models and branch-and-cut algorithms for pickup and delivery problems with time windows. Networks 49, 258-272, 2007.

[142] Ropke S., Pisinger D., An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation Science 40, 455-472, 2006.

[143] Miao L., Ruan Q., Woghiren K., Ruo Q., A hybrid genetic algorithm for the vehicle routing problem with three-dimensional loading constraints. RAIRO - Operations Research 46, 63-82, 2011.

[144] Savelsbergh M.W.P., The vehicle routing problem with time windows : Minimizing route duration. INFORMS Journal on Computing 4, 146-154, 1992.

[145] Schrimpf G., Schneider J., Stamm-Wilbrandt H., Dueck G., Record breaking optimization results using the ruin and recreate principle. Journal of Computational Physics 159, 139-171, 2000.

[146] Séguin R., Problèmes Stochastiques de tournées de véhicules. Thèse de doctorat, Université de Montréal, 1994.

[147] da Silveira J.L.M., Miyazawa F.K., Xavider E.C., Heuristics for the strip packing problem with unloading constraints. Computers & Operations Research 40, 991-1003, 2013.

[148] Shaw P., Using constraint programming and local search methods to solve vehicle routing problems, Dans : Proceedings of the Fourth International Conference on Principles and Practice of Constraint Programming, M. Maher and J.-F. Puget (eds.), Springer-Verlag, Berlin, 417-431, 1998.

[149] Solomon M.M., Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations Research 35, 254-265, 1987.

[150] Strodl J., Doerner K. F., Tricoire F., Hartl R., On index structures in hybrid metaheuristics for routing problems with hard feasibility checks : an application to the 2-dimensional loading vehicle routing problem. Dans : Hybrid Metaheuristics, Lecture Notes in Computer Science 6373, Blesa M.J., Blum C., Raidl G., Roli A., Sampels M., Springer-Verlag, Berlin, 160-173, 2010.

[151] Tadei R., G. Perboli, Della Croce F., A heuristic algorithm for the auto-carrier transportation problem. Transportation Science 36, 55-62, 2002.

[152] Tarantilis C. D., Zachariadis E. E., Kiranoudis C.T., A hybrid metaheuristic algorithm for the integrated vehicle routing and three-dimensional container-loading problem. IEEE Transactions on Intelligent Traportation Systems 10, 255-271, 2009.

[153] Tillman F., The multiple terminal delivery problem with probabilistic demands. Transportation Science 3, 192-204, 1969.

[154] Toulouse S., Wolfer Calvo R., On the complexity of the multiple stack TSP, kSTSP. Dans : Theory and Applications of Models of Computation, Lecture Notes in Computer Science 5532, Chen J., Cooper S.B., Springer-Verlag, Berlin, 360-369, 2009.

[155] Toussaint H., Algorithmique rapide pour les problèmes de tournées et d'ordonnancement. Thèse de Doctorat, Université Blaise Pascal, Clermond-Ferrand II, 2010.

[156] Toth P., Vigo D., The vehicle routing problem. SIAM monographs on discrete mathematics and applications. SIAM, Philadelphia, 2002.

[157] Tricoire, F., Doerner K., Hartl, R., Iori, M., Heuristic and exact algorithms for the multi-pile vehicle routing problem, OR Spectrum 33, 931-959, 2011.

[158] Van Slyke R.M., Wets R., L-shaped linear programs with applications to optimal control and stochastic programming, SIAM Journal on Applied Mathematics 17, 638-663, 1969.

[159] Voudouris C., Guided local search for combinatorial problems. Ph.D. Dissertation, University of Essex, UK, 1997.

[160] Wang L., Guo S., Chen S., Zhu W., Lim A., Two natural heuristics for 3D packing with practical loading constraints. Dans : PRICAI 2010 : Trends in Artificial Intelligence, Lecture Notes in Computer Science 6230, Zhang B.-T., Orgun M.A., Springer-Verlag, Berlin, 256-267, 2010.

[161] Wäscher G., Haussner H., Schumann H., An improved typology of cutting and packing problems. European Journal of Operational Research 183, 1109-1130, 2007.

[162] Weyland, D., Stochastic vehicle routing - From theory to practice, Doctoral dissertation, Faculty of Informatics, UniversitÃă della Svizzera Italiana, Switzerland, 2013.

[163] Wolsey L.A., Valid inequalities for 0-1 knapsacks and MIPS with generalised upper bound constraints. Discrete Applied Mathematics 29, 251-261, 1990.

[164] Zachariadis E. E., Kiranoudis C.T., A strategy for reducing the computational complexity of local search-based methods for the vehicle routing problem. Computers & Operations Research 37, 2089-2105, 2010.

[165] Zachariadis E. E., Tarantilis C. D., Kiranoudis C.T., A guided tabu search for the vehicle routing problem with two dimensional loading constraints. European Journal of Operational Research 3, 729-743, 2009.

[166] Zachariadis E. E., Tarantilis C. D., Kiranoudis C.T., The pallet packing vehicle routing problem, Transportation Science 46, 341-358, 2012.