UNIVERSITÉ DE MONTRÉAL

LE PROBLÈME DE TOURNÉES DE VÉHICULES AVEC CUEILLETTES, LIVRAISONS, FENÊTRES DE TEMPS ET CONTRAINTES DE MANUTENTION

MARILÈNE CHERKESLY
DÉPARTEMENT DE MATHÉMATIQUES ET DE GÉNIE INDUSTRIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION
DU DIPLÔME DE PHILOSOPHIÆ DOCTOR
(MATHÉMATIQUES DE L'INGÉNIEUR)
JUIN 2015

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée :

LE PROBLÈME DE TOURNÉES DE VÉHICULES AVEC CUEILLETTES, LIVRAISONS, FENÊTRES DE TEMPS ET CONTRAINTES DE MANUTENTION

présentée par : CHERKESLY Marilène
en vue de l'obtention du diplôme de : Philosophiæ Doctor
a été dûment acceptée par le jury d'examen constitué de :

DÉDICACE

À mes parents, merci de m'avoir encouragée.

REMERCIEMENTS

Aux moments d'écrire ces lignes, plusieurs sentiments m'habitent. Je suis à la fois contente de terminer quatre ans de recherche, mais je suis triste de partir de mon confort quotidien du GERAD. Ce doctorat représente quatre années hautes en émotion. La première année a probablement été la plus difficile : j'ai du rattraper toutes les bases qu'ils me manquaient en mathématiques. Aujourd'hui, je peux dire que j'ai appris énormément. Je m'apprête maintenant à relever un défi additionnel : remercier les gens qui ont été une source d'inspiration et de motivation pendant ces quatre dernières années. Je me lance !

Tout d'abord, j'aimerais remercier mes directeurs Guy Desaulniers et Gilbert Laporte qui ont cru en moi et qui ont fait de moi la chercheuse que je suis. Merci de m'avoir appris les rudiments de la recherche. Merci de m'avoir donné l'occasion de présenter mes résultats lors de plusieurs congrès scientifiques internationaux. Merci de m'avoir mise en contact avec Stefan Irnich et de m'avoir encouragée dans le projet avec Marjolein Veenstra.

J'aimerais remercier François Lessard qui a implémenté une grosse partie du code pour deux de mes articles. Sans ton travail assidu, je n'aurais probablement pas terminé ma thèse en quatre ans. Tu as été très efficace et surtout super sympathique ! C'était toujours agréable de discuter avec toi... même via Skype depuis Mainz, en Allemagne.

Dankeschön à Stefan Irnich et à tous les membres de la chaire en gestion de la logistique : Angela, Christian, David, Timo G. et Timo H. Vous m'avez donné l'occasion de faire de la recherche en Allemagne et d'apprendre quelques mots d'allemand. J'ai été très heureuse de vous rencontrer et de travailler avec vous. *Tschüss !*

J'aimerais également remercier les membres de mon jury : Louis-Martin Rousseau, Jacques Desrosiers et Dominique Feillet. Merci d'avoir accepté de faire partie de mon jury de thèse.

Un merci tout spécial à Mélisende Brazeau. Merci d'avoir été là pendant mes deux premières années de doctorat (et de ta maîtrise). Sans toi, je ne sais pas si j'aurais eu la détermination de passer au travers de ces années. Merci de m'avoir soutenue dans les moments plus difficiles et d'avoir rendu les moments joyeux aussi amusants et rigolos !

Merci à tous les étudiants qui sont passés par mon bureau et qui ont su m'endurer. Merci en particulier à Atoosa qui a été là du premier au dernier jour de mon doctorat. Merci à Lê avec qui j'ai beaucoup discuté. Je crois que sans toi mes journées au GERAD auraient semblé plus longues. Tu es maintenant au MIT et j'espère que tu as pu trouver Marilène no. 2. Merci à Camille avec qui nous avons fait une course pour savoir qui finirait par rédiger en premier sa thèse (son mémoire pour Camille). Je n'ai malheureusement pas gagné, mais j'ai réussi à soumettre le tout une semaine après toi ! Atoosa, Lê et Camille, vous avez été des collègues de bureau incroyables !

Merci également aux *gars* : Antoine, Charles, Jean-Bertrand, Jérémy, Samuel, Sébastien et Thibault. Vous avez été une source d'inspiration et de détermination. Surtout, ne doutez pas de vos capacités car vous êtes tous doués. J'ai eu plusieurs fous rires avec vous ! Vous allez me manquer maintenant que je quitte pour un postdoctorat, mais sachez qu'on se reverra et que vous allez être *pognés* avec moi pour toute votre vie. Merci également aux filles. Vous n'êtes pas nombreuses ! En particulier, merci Mathilde d'avoir cru en moi quand je ne savais pas coder et que je ne connaissais rien aux mathématiques. Merci de m'avoir expliqué en me donnant l'impression de comprendre et d'être la meilleure. Merci Hélène-Sarah d'être qui tu es ! Tu dégages une énergie débordante, continue comme ça ! Merci à ceux qui ont évolué et qui sont rendus ailleurs, en particulier, Vincent et Martin. Merci à tous les autres étudiants du GERAD que je n'ai pas nommés ici, vous avez fait de mes midis des occasions d'apprentissage et de partage.

Merci à tous mes enseignants avec qui j'ai appris énormément. Surtout ne changez pas !

Finalement, un merci à ma famille. Merci Olivier de m'avoir supporté pendant ces quatre années hautes en émotion. Merci à mes parents et à ma soeur de m'avoir encouragée à poursuivre mes études supérieures et à postuler à des bourses du CRSNG. Merci à mes cousins, à mes oncles et tantes qui ont su me motiver à toutes les semaines malgré les obstacles que je rencontrais.

RÉSUMÉ

Les problèmes de tounées de véhicules avec cueillettes et livraisons consistent à trouver des tournées réalisables minimisant le nombre de véhicules utilisés et la distante totale parcourue, et permettant de compléter toutes les requêtes. Une requête est définie par un point de cueillette et un point de livraison, et une quantité de marchandise à transporter du point de cueillette au point de livraison. Ce faisant, une tournée est dite réalisable si la charge du véhicule ne dépasse pas sa capacité et si, pour chaque requête, on visite le point de cueillette avant le point de livraison avec le même véhicule. Dans la dernière décennie, la communauté de recherche opérationnelle s'est attaquée à des problèmes de plus en plus complexes qui tiennent compte de contraintes opérationnelles difficiles à traiter. Cette thèse s'insère dans cette tendance.

Cette thèse propose des modèles et des algorithmes pour résoudre deux variantes du problème de tournées de véhicules avec cueillettes et livraisons : le problème de tournées de véhicules avec cueillettes, livraisons, fenêtres de temps et contrainte de chargement dernier entré premier sorti (*last-in-first-out* – LIFO) (*pickup and delivery problem with time windows and LIFO loading* – PDPTWL) et le problème de tournées de véhicules avec fenêtres de temps et plusieurs piles (*pickup and delivery problem with time windows and multiple stacks* – PDPTWMS). Dans le PDPTWL, la contrainte de chargement dernier entré premier sorti stipule qu'aucune manutention non nécessaire n'est faite lors de la livraison d'un item : un item peut seulement être livré s'il est situé sur le dessus de la pile. Dans le PDPTWMS, chaque véhicule contient plusieurs piles qui sont gérées selon une politique de chargement dernier entré premier sorti.

Afin de résoudre le PDPTWL, trois algorithmes de génération de colonnes avec plans coupants et un algorithme heuristique sont proposés. Le premier algorithme de génération de colonnes incorpore la contrainte de chargement dans le problème maître, alors que le second l'incorpore dans le sous-problème. Pour ce faire, un algorithme d'étiquetage et un critère de dominance spécialisés sont proposés. Le troisième algorithme de génération de colonnes est une combinaison des deux premiers algorithmes. Des inégalités valides connues sont adaptées pour le PDPTWL. Des instances ayant jusqu'à 75 requêtes sont résolues par ces trois algorithmes exacts en une heure de temps de calcul.

L'algorithme heuristique, quant à lui, permet de traiter plus rapidement des instances de plus grande taille. D'abord, un ensemble de solutions initiales est construit avec un algorithme glouton. Puis, pour chaque solution, un algorithme de recherche locale est utilisé afin de diminuer en priorité le nombre de véhicules et ensuite la distance totale parcourue. Puis, deux stratégies sont utilisées pour créer des solutions enfants. La première choisit aléatoirement des tournées de l'ensemble de solutions alors que la deuxième utilise un opérateur de croisement. Pour les deux stratégies, un algorithme de recherche locale est ensuite utilisé. Finalement, les enfants sont ajoutés à l'ensemble de solutions et les meilleurs survivants sont conservés. L'ensemble de solutions est géré afin de garder uniquement les solutions variées de meilleure qualité par rapport au coût total. Des instances ayant jusqu'à 300 requêtes sont résolues par cette heuristique en deux heures de temps de calcul.

Afin de résoudre le PDPTWMS, deux algorithmes de génération de colonnes avec plans coupants sont proposés. Le premier algorithme de génération de colonnes incorpore la contrainte de chargement avec plusieurs piles dans le sous-problème. Pour ce faire, un algorithme d'étiquetage et un critère de dominance spécialisés sont proposés. Le deuxième algorithme incorpore partiellement la contrainte de chargement avec plusieurs piles dans le sous-problème et ajoute, au besoin, des contraintes au problème maître lorsque la solution trouvée ne respecte pas la contrainte de chargement avec plusieurs piles. Des instances avec une, deux et trois piles et ayant jusqu'à 75 requêtes sont résolues par ces deux algorithmes exacts en deux heures de temps de calcul.

ABSTRACT

In the pickup and delivery problem, vehicles based at a depot are used to satisfy a set of requests which consists of transporting goods (or items) from a specific pickup location to a specific delivery location. We consider an unlimited fleet of identical vehicles with multiple homogeneous compartments of limited capacity. A vehicle route is feasible if the load in each compartment of the vehicle does not exceed its capacity and each completed request is first picked up at its pickup location and then delivered at its corresponding delivery location. The pickup and delivery problem consists of determining a set of least-cost feasible routes in which the number of vehicles is first minimized. In the last decade, the operations research community has tackled more complex problems that consider real-life constraints. This thesis follows this trend.

This thesis proposes models and algorithms for two variants of the pickup and delivery problem: the pickup and delivery problem with time windows and last-in-first-out (LIFO) loading constraints (PDPTWL) and the pickup and delivery problem with time windows and multiple stacks (PDPTWMS). In the first problem, the LIFO loading rule ensures that no handling is required prior to unloading an item from a vehicle: an item can only be delivered if it is the last one in the stack. In the second problem, each vehicle contains multiple stacks that are operated in a LIFO fashion.

To solve the PDPTWL, three exact branch-price-and-cut algorithms and one metaheuristic algorithm are developed. The first branch-price-and-cut algorithm incorporates the LIFO constraints in the master problem. The second branch-price-and-cut algorithm handles the LIFO constraints directly in the shortest path pricing problem and applies a dynamic programming algorithm relying on an ad hoc dominance criterion. The third branch-price-and-cut algorithm is a hybrid between the first two. Known valid inequalities are adapted to the PDPTWL. Instances with up to 75 requests are solved within one hour of computational time.

The metaheuristic is capable of handling larger instances much faster. First, a set of initial solutions is generated with a greedy randomized adaptive search procedure. For each of these solutions, local search is applied in order to first decrease the total number of vehicles and then the total traveled distance. Two different strategies are used to create offspring. The first selects vehicle routes from the solution pool. The second selects two parents to create an offspring with a crossover operator. For both strategies, local search is then performed on the child solution. Finally, the offspring is added to the population and the best survivors are kept. The population is managed so as to maintain good quality solutions with respect to total cost and population diversity. Instances with up to 300 requests are solved within two hours of computational time.

To solve the PDPTWMS, two exact branch-price-and-cut algorithms are proposed. The first branch-price-and-cut algorithm handles the multiple stacks policy in the shortest path pricing problem and applies a dynamic programming algorithm relying on an ad hoc dominance criterion. The second branch-price-and-cut algorithm incorporates the multiple stacks policy partly in the shortest path pricing problem and adds additional inequalities to the master problem when infeasible LIFO multiple stacks are encountered. Instances with one, two and three stacks involving up to 75 requests are solved within two hours of computational time.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

DTSPMS	Double traveling salesman problem with multiple stacks
DVRPMS	Double vehicle routing problem with multiple stacks
LIFO	Last-in-first-out
PDPL	Pickup and delivery problem with LIFO loading
PDPTW	Pickup and delivery problem with time windows
PDPTWL	Pickup and delivery problem with time windows and LIFO loading
PDPTWMS	Pickup and delivery problem with time windows and multiple stacks
PDTSPMS	Pickup and delivery traveling salesman problem with multiple stacks
TSPPDL	Traveling salesman problem with pickup and delivery and LIFO loading
VRP	Vehicle routing problem

CHAPITRE 1 INTRODUCTION

Les problèmes de tournées de véhicules se posent dans le secteur du transport par camion. Selon un rapport publié par Statistique Canada (2012), l'industrie du transport et de l'entreposage représentait 4,2% du PIB global du Canada en 2012 se situant à 64,7 millards de dollars. En 2011, le transport par camion constituait la première composante du PIB du transport représentant 28,4% ou 17,0 milliards de dollars. Considérant l'importance du secteur du camionnage pour le PIB global du Canada, il semble intéressant de pouvoir diminuer ses coûts afférents.

Dans cette thèse, nous nous pencherons sur deux variantes du problème de tournées de véhicules : le problème de tournées de véhicules avec cueillettes et livraisons, fenêtres de temps et contrainte de chargement dernier entré premier sorti (*last-in-first-out* – LIFO) (*pickup and delivery problem with time windows and LIFO loading* – PDPTWL) et le problème de tournées de véhicules avec cueillettes et livraisons, fenêtres de temps et plusieurs piles (*pickup and delivery problem with time windows and multiple stacks* – PDPTMS). Le PDPTWL et le PDPTWMS sont des variantes du problème de tournées de véhicules avec cueillettes et livraisons pour le transport de marchandises (voir Battara *et al.*, 2014). Nous modéliserons et proposerons des algorithmes spécifiques pour chaque problème.

Décrivons d'abord le problème de tournées de véhicules avec cueillettes et livraisons et fenêtres de temps (*pickup and delivery problem with time windows* – PDPTW). Considérant une flotte de véhicules identiques ayant une capacité limitée, le PDPTW consiste à trouver des routes réalisables minimisant les coûts et permettant de compléter toutes les requêtes. Une requête est définie par un point de cueillette et un point de livraison et une quantité de marchandise à transporter du point de cueillette au point de livraison. Ce faisant, le même véhicule doit nécessairement visiter le point de cueillette avant le point de livraison associé à une requête. Chaque point de cueillette ou de livraison spécifie une plage horaire de visite pendant laquelle on peut recueillir ou livrer la marchandise. Il s'agit d'une fenêtre de temps. Une route est dite réalisable si la charge du véhicule en tout temps ne dépasse pas sa capacité, si pour une requête on visite le point de cueillette avant le point de livraison avec le même véhicule et si chaque point est visité à l'intérieur de la fenêtre de temps spécifiée. Puisqu'il y a souvent des coûts élevés d'utilisation des véhicules, les coûts sont associés au nombre de véhicules utilisés et à la distance totale parcourue. En général, il est donc intéressant de minimiser le nombre de véhicules, puis de minimiser la distance.

Le PDPTWL est une variante du PDPTW. Dans cette variante, chaque véhicule contient

$$\begin{tikzcd}
(a) & 0 \arrow[r] & 1^+ \arrow[r] & 2^+ \arrow[r] & 2^- \arrow[r] & 1^- \arrow[r] & 0 \\
(b) & 0 \arrow[r] & 1^+ \arrow[r] & 2^+ \arrow[r] & 1^- \arrow[r] & 2^- \arrow[r] & 0
\end{tikzcd}$$

Figure 1.1 – Tournées de véhicules pour lesquelles (a) la politique LIFO est respectée et (b) la politique LIFO n'est pas respectée parce que le colis chargé au noeud 1^+ ne peut pas être livré au noeud 1^- sans préalablement décharger le colis chargé au noeud 2^+.

un seul compartiment qui est chargé par l'arrière. Ce compartiment est alors géré comme une pile et la politique LIFO doit être respectée. Cette politique stipule que lorsqu'un point de cueillette est visité, la quantité chargée est mise sur le dessus de la pile. On pourra alors visiter un point de livraison si et seulement si la marchandise à livrer se trouve sur le dessus de la pile. Ce problème survient lors du transport de marchandises dangereuses, fragiles ou lourdes. Dénotons par 0, i^+ et i^- le dépôt ainsi que le point de cueillette et de livraison de la requête i. La Figure 1.1 représente une tournée pour laquelle la contrainte LIFO est respectée et une tournée pour laquelle la contrainte LIFO n'est pas respectée.

Le PDPTWMS est une variante du PDPTWL. Chaque véhicule contient plusieurs compartiments homogènes de capacité limitée. Chaque compartiment est chargé par l'arrière du véhicule et est géré comme une pile. Dans ce cadre, la politique LIFO doit être respectée pour chaque compartiement. Cette dernière stipule que lorsqu'un point de cueillette est visité, la quantité chargée est mise sur le dessus d'une pile. On pourra alors visiter un point de livraison si et seulement si la marchandise à livrer se trouve sur le dessus d'une des piles. Ce problème survient lors du transport de voitures entre des concessionnaires automobiles à l'aide de véhicules à plusieurs niveaux ou lors du transport d'animaux vers des abattoirs à l'aide de véhicules avec plusieurs compartiments. La Figure 1.2 présente une tournée pour laquelle la contrainte LIFO est respectée pour un véhicule contenant deux compartiments.

À notre connaissance, il n'existe aucune méthode permettant de résoudre le PDPTWL. Quelques méthodes exactes basées sur la méthode d'énumération implicite (voir Cordeau *et al.* (2010) et Carrabs *et al.* (2007a)) ont été adaptées pour un problème similaire avec un seul véhicule : le problème de voyageur de commerce avec cueillettes, livraisons et contrainte de chargement LIFO (*traveling salesman problem with pickup and delivery and LIFO loading* – TSPPDL). De plus, plusieurs méthodes heuristiques basées sur la recherche à grand voisinage ont trouvé des résultats pour le TSPPDL (voir Carrabs *et al.* (2007b)) et le problème de tournées de véhicules avec cueillettes, livraisons et contrainte de chargement LIFO (*pickup and delivery problem with LIFO loading* – PDPL) (voir Gao *et al.* (2011) et Cheang *et al.*

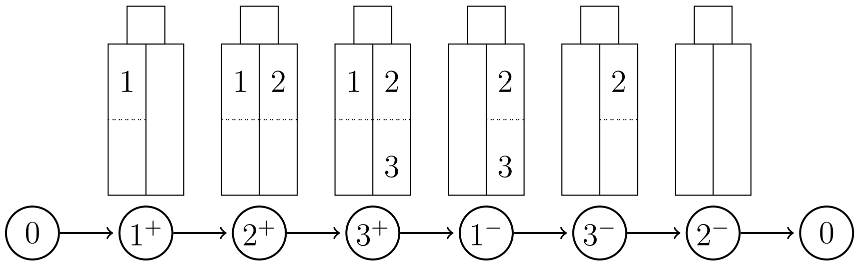


Figure 1.2 – Tournée respectant les contraintes de capacité et la politique LIFO pour chaque compartiment. Le véhicule a deux compartiments ayant chacun une capacité de 2. Toutes les requêtes ont une demande unitaire.

(2012)). Plus récemment, Benavent *et al.* (2015) ont proposé une méthode exacte et une méthode heuristique pour résoudre une variante du PDPL.

À notre connaissance, il n'existe aucune méthode permettant de résoudre le PDPTWMS. Quelques méthodes exactes et heuristiques ont été développées pour des variantes du PDPTWMS. En effet, quelques auteurs ont proposé des méthodes de résolution pour le problème de voyageur de commerce double avec plusieurs piles (*double traveling salesman problem with multiple stacks* – DTSPMS) dans lequel les noeuds de cueillette et les noeuds de livraison sont dans deux régions géographiques différentes ; les cueillettes doivent donc être complétées avant les livraisons. Petersen et Madsen (2009) ont résolu ce problème avec des heuristiques et Alba Martínez *et al.* (2013); Lusby *et al.* (2010); Petersen *et al.* (2010) avec des algorithmes exacts. Plus récemment, Iori et Riera-Ledesma (2015) ont suggéré des algorithmes exacts pour le problème avec plusieurs véhicules (*double vehicle routing problem with multiple stacks* – DVRPMS). Côté *et al.* (2012a,b) ont également développé un algorithme d'énumération implicite avec plans coupants et un algorithme heuristique afin de résoudre un cas particulier avec un véhicule et sans fenêtre de temps du PDPTWMS, le problème de voyageur de commerce avec plusieurs piles (*pickup and delivery traveling salesman problem with multiple stacks* – PDTSPMS).

Compte tenu de l'absence d'algorithmes pour résoudre le PDPTWL et le PDPTWMS, nous formulerons et résolverons dans cette thèse le PDPTWL et le PDPTWMS. Pour ce faire, nous développerons une méthode exacte basée sur la génération de colonnes avec l'ajout de plans coupants ainsi qu'une méthode heuristique basée sur une combinaison d'algorithmes génétiques et d'algorithmes de recherche locale pour le PDPTWL. Nous adapterons ensuite la méthode exacte basée sur la génération de colonnes avec l'ajout de plans coupants pour le PDPTWL au PDPTWMS.

Le présent document est structuré comme suit. Le chapitre 2 permet de faire une brève revue de la littérature sur des variantes des problèmes étudiés. Le chapitre 3 introduit le corps de l'ouvrage qui est constitué des chapitres 4, 5 et 6. Le chapitre 4 contient un article publié dans *Transportation Science* où trois méthodes de génération de colonnes avec plans coupants sont proposées pour le PDPTWL. Le chapitre 5 est publié dans *Computers & Operations Research* et introduit une métaheuristique pour résoudre le PDPTWL. Le chapitre 6 est un article soumis à *European Journal of Operational Research* qui développe un algorithme de génération de colonnes avec plans coupants pour le PDPTWMS. Finalement, une discussion générale est présentée au chapitre 7 et des conclusions sont apportées au chapitre 8.

CHAPITRE 2 REVUE DE LITTÉRATURE

Les problèmes de tournées de véhicules ont été largement étudiés dans la littérature. Ils font d'ailleurs l'objet de plusieurs livres dont ceux de Toth et Vigo (2002; 2014). Cette revue de littérature présente les meilleures algorithmes exacts et heuristiques pour résoudre des variantes du problème de tournées de véhicules, soit la variante avec cueillettes et livraisons, la variante avec cueillettes, livraisons et contrainte de chargement et la variante avec cueillettes, livraisons et plusieurs piles.

La revue de littérature est structurée comme suit. Dans la section 2.1, nous décrivons les meilleurs algorithmes pour le problème de tournées de véhicules avec et sans fenêtres de temps. Dans la section 2.2, nous présentons les meilleurs algorithmes pour le problème de tournées de véhicules avec cueillettes, livraisons et fenêtres de temps. Dans la section 2.3, nous détaillerons les algorithmes existants pour le problème de tournées de véhicules avec cueillettes, livraisons et contrainte de chargement LIFO. Dans la section 2.4, nous expliquerons les algorithmes connus à ce jour pour le problème voyageur de commerce avec cueillettes, livraisons et plusieurs piles. Finalement, un tableau sommaire est présenté à la section 2.5.

2.1 Problème de tournées de véhicules

Le problème de tournées de véhicules a été largement étudié. Plusieurs articles synthèses portent sur le sujet, notamment, Laporte (2007, 2009); Irnich *et al.* (2014). Bräysy and Gendreau (2005a,b); Vidal *et al.* (2013a); Desaulniers *et al.* (2014) ont également présenté des articles synthèses portant sur les divers algorithmes pour résoudre le problème de tournées de véhicules avec fenêtres de temps.

Parmi l'ensemble des algorithmes exacts connus pour résoudre le problème de tournées de véhicules, Baldacci *et al.* (2010, 2011b) proposent un algorithme qui combine la génération de colonnes, une procédure d'énumération de tournées et un algorithme d'énumération implicte. De plus, ils exploitent une relaxation du problème de plus court chemin élémentaire, appelé *ng-path*, qui permet certains cycles. Ils arrivent à résoudre des instances contenant jusqu'à 233 noeuds et présentent les meilleurs résultats connus jusqu'à présent.

Parmi les algorithmes heuristiques proposés pour résoudre le problème de tournées de véhicules, l'algorithme heuristique de Vidal *et al.* (2013b) est l'un des plus performants. Il s'agit d'un algorithme génétique où la gestion de l'ensemble des solutions permet de conserver des solutions diversifiées de bonne qualité. Des instances contenant jusqu'à 1000 noeuds sont résolues. À ce jour, leurs résultats sont les meilleurs connus pour résoudre de nombreuses variantes du problème de tournées de véhicules, en particulier, le VRPTW.

2.2 Problème de tournées de véhicules avec cueillettes, livraisons et fenêtres de temps

Le problème de tournées de véhicules avec cueillettes et livraisons a été largement étudié. Berbeglia *et al.* (2007) ont suggéré une norme de classification pour les diverses variantes de ce problème. Parragh *et al.* (2008a,b) ont synthétisé la littérature scientifique sur les variantes existantes du problème. Finalement, Battara *et al.* (2014) et Doerner et Salazar-González (2014) ont rédigé, respectivement, une synthèse portant sur les divers algorithmes pour le problème de cueillettes et livraisons pour le transport de marchandises et pour le transport de personnes. Dans cette section, nous présentons les meilleurs algorithmes exacts et heuristiques pour le PDPTW.

Algorithmes exacts

Ropke *et al.* (2007) proposent deux formulations mathématiques pour le PDPTW et le problème de transport à la demande (*dial-a-ride problem*). La seconde formulation contient moins de variables et formule certaines contraintes par des inégalités de capacité arrondie (*rounded capacity inequalities*) et des inégalités de chemin irréalisable (*infeasible path inequalities*). Afin de résoudre les deux problèmes, ils adaptent un algorithme d'énumération implicite avec plans coupants (*branch-and-cut*). Ils arrivent à résoudre à l'optimalité des instances ayant jusqu'à 96 requêtes. Leurs résultats démontrent que la deuxième formulation mathématique permet de résoudre plus rapidement les instances proposées.

Ropke et Cordeau (2009) développent un algorithme de génération de colonnes avec plans coupants (*branch-and-cut-and-price*) pour le PDPTW. Leur fonction objectif minimise le nombre de véhicules utilisés ainsi que les coûts de transport. Les auteurs décrivent un algorithme d'étiquetage (*labeling algorithm*) ainsi qu'un critère de dominance pour résoudre le sous-problème. La majorité des instances allant jusqu'à 100 requêtes et quelques instances avec 500 requêtes sont résolues.

Baldacci *et al.* (2011a) combinent un algorithme de génération de colonnes pour calculer une borne inférieure, une procédure d'énumération de tournées et un algorithme d'énumération implicite pour résoudre de manière exacte le PDPTW. Ils considèrent deux fonctions objectifs différentes : (1) minimiser les coûts de transport et (2) minimiser le nombre de véhicules et les coûts de transport. Ils arrivent à résoudre la majorité des instances allant jusqu'à 100 requêtes et quelques instances allant jusqu'à 500 requêtes. De plus, ils résolvent plus rapidement les instances solutionnées par Ropke et Cordeau (2009) et arrivent à résoudre 15 instances plus difficiles non résolues par ces derniers.

Algorithmes heuristiques

Li et Lim (2003) mettent au point le premier algorithme heuristique permettant de résoudre efficacement des instances de grande taille du PDPTW. Leur algorithme repose sur la recherche tabou avec recuit simulé (*tabu-embedded simulated annealing*). Afin de définir le voisinage, ils suggèrent trois opérateurs de recherche locale. Les auteurs élaborent 56 instances avec 50 requêtes découlant des instances de Solomon pour le VRPTW.

Bent et Van Hentenryck (2006) conçoivent une heuristique en deux phases. La première phase se sert d'un algorithme de recuit simulé pour diminuer le nombre de véhicules utilisés alors que la deuxième phase diminue le coût total de transport avec un algorithme de recherche à grand voisinage. Leur algorithme permet de trouver la meilleure solution connue pour plusieurs instances allant jusqu'à 300 requêtes. Pour les instances de 50 requêtes, il est très compétitif avec celui de Li et Lim (2003).

Ropke et Pisinger (2006) proposent un algorithme adaptif de recherche à grand voisinage (*adaptive large neighborhood search*). Leur heuristique utilise plusieurs opérateurs d'insertion et de retrait de requêtes. La procédure autoadaptive de l'heuristique permet de choisir l'opérateur d'insertion ou de retrait utilisé à l'itération courante. La probabilité de choisir chaque opérateur dépend d'un poids qui varie d'une itération à l'autre selon la performance passée de chaque opérateur. Afin de réduire le nombre de véhicules et les coûts de transport, l'heuristique est implanté en deux phases où la première phase consiste à diminuer le nombre de véhicules et la deuxième phase la distance totale parcourue. L'heuristique a été testée sur plusieurs instances allant jusqu'à 500 requêtes.

2.3 Problème de tournées de véhicules avec cueillettes, livraisons et contrainte de dernier entré premier sorti

Le problème de tournées de véhicules avec cueillettes, livraisons et contrainte de dernier entré premier sorti est un problème récent de la littérature. Peu d'algorithmes exacts et heuristiques existent. Toutefois, Iori et Martello (2010) ont synthétisé la littérature sur les diverses contraintes de chargement (2 dimensions, 3 dimensions, plusieurs piles, contraintes de chargement et autres variantes). Dans cette section, nous décrivons les algorithmes existants pour résoudre la variante avec un véhicule et la variante avec plusieurs véhicules.

2.3.1 Un seul véhicule

Le TSPPDL consiste à trouver une tournée de coût minimum pour un seul véhicule débutant au dépôt d'origine, visitant une et une seule fois chaque noeud et se terminant au dépôt d'arrivée. La tournée doit respecter les relations de préséance qui requièrent que pour chaque requête, le noeud de cueillette soit visité avant le noeud de livraison. La tournée doit également respecter la politique LIFO. Dans cette section, nous voyons les algorithmes exacts et heuristiques développés pour le TSPPDL.

Algorithmes exacts

Carrabs *et al.* (2007a) présentent un algorithme additif d'énumération implicite (*additive branch-and-bound*) où le calcul de la borne inférieure se fait de manière additive avec des relaxations différentes. Ils arrivent à résoudre à l'optimalité des instances allant jusqu'à 21 requêtes.

Cordeau *et al.* (2010) adaptent un algorithme d'énumération implicite avec plans coupants. Ils formulent de nouvelles inégalités valides pour le TSPPDL dont les inégalités de prédécesseurs et de successeurs incompatibles (*incompatible predecessor and successor inequalities*). Des instances contenant jusqu'à 17 requêtes sont résolues à l'optimalité en 10 minutes et la plus grande instance résolue contient 25 requêtes.

Algorithmes heuristiques

Cassani (2004) met au point un algorithme de recherche locale pour résoudre le TSPPDL. Il élabore quatre opérateurs de recherche locale. Des instances allant jusqu'à 23 requêtes sont résolues. Cet algorithme, est ensuite amélioré par Carrabs *et al.* (2007b). Ils introduisent trois opérateurs de recherche locale supplémentaires. Ils arrivent à résoudre des instances allant jusqu'à 375 requêtes en moins de 40 minutes.

Li *et al.* (2011) conçoivent un algorithme de recherche locale pour résoudre le TSPPDL qui repose sur une nouvelle structure informatique permettant de représenter un chemin respectant la politique LIFO. Cette structure possède plusieurs avantages : les contraintes de chargement et de préséance sont toujours respectées. Ils ont adapté les opérateurs proposés par Cassani (2004) et Carrabs *et al.* (2007b) à cette nouvelle structure. Ils réussissent à résoudre des instances allant jusqu'à 500 requêtes en moins d'une heure.

2.3.2 Plusieurs véhicules

Le TSPPDL peut être généralisé au cas avec plusieurs véhicules. À ce jour, il existe un seul algorithme exact et trois algorithmes heuristiques pour résoudre ce problème.

Algorithme exact

Benavent *et al.* (2015) développent un algorithme d'énumération implicite avec plans coupants pour résoudre le PDPL avec contrainte de temps maximal sur les tournées. Toutes les instances proposées allant jusqu'à 40 requêtes et quelques instances avec 50 et 60 requêtes sont résolues en moins d'une heure.

Algorithmes heuristiques

Ambrosini *et al.* (2004) introduisent le premier algorithme heuristique pour le PDPL. Il s'agit d'un algorithme de recherche avec une adaptation gloutonne (*GRASP : greedy randomized adaptive search procedure*) qui procède en deux phases : la première phase construit une tournée réalisable et la deuxième phase améliore la solution trouvée à l'aide de deux opérateurs de recherche locale. Ils solutionnent des instances allant jusqu'à 100 requêtes en 5 minutes.

Gao *et al.* (2011) reprennent la structure informatique proposée par Li *et al.* (2011) afin de l'intégrer dans une heuristique de recherche à voisinage variable pour le PDPL avec contrainte de distance maximale. Cinq opérateurs inter-tournées de recherche locale sont implantés. Ils résolvent des instances impliquant jusqu'à 375 requêtes en moins de 15 minutes. Cet algorithme est amélioré par Cheang *et al.* (2012). Ces derniers ajoutent deux opérateurs inter-tournées. De plus, l'algorithme est structuré en deux étapes. Dans la première étape, on tente de réduire le nombre de véhicules à l'aide d'un algorithme de recuit simulé et à l'aide d'un algorithme avec éjections (*ejection pool*). La deuxième étape consiste à réduire la distance totale à l'aide d'un algorithme de recherche à voisinage variable et un algorithme de recherche tabou. Parmi les différentes variantes proposées, la combinaison de l'algorithme avec éjections pour la première étape et de la recherche à grand voisinage pour la deuxième étape donne les meilleurs résultats sur leurs instances. Ils résolvent des instances comportant jusqu'à 375 requêtes. Leurs résultats s'avèrent meilleurs que ceux trouvés par Gao *et al.* (2011), bien que leurs temps de calculs soient plus élevés.

Benavent *et al.* (2015) élaborent un algorithme de recherche avec tabous avec démarrage multiple (*multi-start*) pour résoudre le PDPL avec contrainte de temps maximal sur les tournées. Ils arrivent à résoudre des instances allant jusqu'à 211 requêtes en moins de 30 minutes.

2.4 Problème de tournées de véhicules avec cueillettes, livraisons et plusieurs piles

Le problème de tournées de véhicules avec cueillettes, livraisons et plusieurs piles est une généralisation du PDPL. Il s'agit d'un problème récent de la littérature. Dans cette section, nous présentons les algorithmes exacts et heuristiques pour le PDTSPMS, le DTSPMS et le DVRPMS.

2.4.1 Un seul véhicule

Le PDTSPMS consiste à trouver une tournée pour un seul véhicule à coût minimum débutant au dépôt d'origine, visitant une et une seule fois chaque noeud et se terminant au dépôt d'arrivée. La tournée doit respecter les relations de préséance qui requièrent que, pour chaque requête, le noeud de cueillette soit visité avant le noeud de livraison. La tournée doit également respecter la politique LIFO pour chacune des piles du véhicule.

Algorithmes exacts

Lusby *et al.* (2010) conçoivent un algorithme constructif pour le DTSPMS qui consiste à coupler les k-meilleures tournées pour le problème de voyageur de commerce contenant uniquement les noeuds de cueillette et pour le problème de voyageur de commerce contenant uniquement les noeuds de livraison. Des instances comportant jusqu'à 18 requêtes et ayant jusqu'à trois piles sont résolues à l'optimalité en moins de trois heures.

Petersen *et al.* (2010) formulent le DTSPMS de trois façons : la première utilise des variables de précédence, la seconde se base sur un modèle de flot et la troisième utilise des inégalités de chemin irréalisable. Les auteurs adaptent un algorithme d'énumération implicite avec plans coupants approprié pour chacune des formulations. Des instances allant jusqu'à 21 requêtes et comportant jusqu'à 18 piles sont résolues à l'optimalité en moins d'une heure. Leurs résultats démontrent que la troisième formulation est la plus performante.

Côté *et al.* (2012a) proposent une formulation mathématique pour le PDTSPMS où la politique LIFO pour chacune des piles est implantée avec des inégalités de chemin irréalisable. Pour résoudre le problème, ils suggèrent un algorithme d'énumération implicite avec plans coupants. Leur algorithme consiste ensuite à résoudre leur formulation mathématique sans les inégalités de chemin irréalisable. Puis, lorsqu'une solution est trouvée, ils vérifient si elle respecte la politique LIFO pour chacune des piles en résolvant un problème de sac à dos. Des instances allant jusqu'à 25 requêtes et quatre piles sont résolues en moins d'une heure.

Alba Martínez *et al.* (2013) développent un algorithme d'énumération implicite avec plans coupants pour résoudre le DTSPMS. Des instances allant jusqu'à 28 requêtes et quatre piles sont résolues en moins de trois heures. De plus, ils arrivent à résoudre certaines instances non résolues par Lusby *et al.* (2010).

Algorithmes heuristiques

Petersen et Madsen (2009) sont les premiers à formuler mathématiquement le DTSPMS. De plus, ils élaborent trois heuristiques pour résoudre ce problème : un algorithme avec recherche tabou, un algorithme avec recuit simulé et un algorithme de recherche à grand voisinage. Ils arrivent à résoudre des instances comportant jusqu'à 66 requêtes et avec exactement trois piles en moins de trois minutes. Leurs résultats démontrent que l'algorithme de recherche à grand voisinage est le plus performant.

Côté *et al.* (2012b) implantant un algorithme de recherche à grand voisinage pour résoudre le PDTSPMS. Pour ce faire, ils définissent plusieurs opérateurs de recherche locale. Ils réussissent ainsi à résoudre des instances allant jusqu'à 375 requêtes et avec quatre piles en moins de 10 minutes.

2.4.2 Plusieurs véhicules

Le TSPPDMS peut être généralisé au cas avec plusieurs véhicules. À notre connaissance, il n'existe pas d'algorithme pour résoudre ce problème. Toutefois, Iori et Riera-Ledesma (2015) ont étudié la généralisation du DTSPMS avec plusieurs véhicules, le DVRPMS. Ils proposent différentes formulations pour le DVRPMS et conçoivent trois algorithmes exacts : un algorithme d'énumération implicite avec plans coupants, un algorithme de génération de colonnes et un algorithme de génération de colonnes avec plans coupants. Ils arrivent à résoudre des instances allant jusqu'à 25 requêtes et trois piles en une heure. Pour leurs instances avec peu de véhicules, leur algorithme d'énumération implicite avec plans coupants est le plus performant, mais pour les instances avec plus de véhicules, les deux autres algorithmes donnent de meilleurs résultats.

2.5 Sommaire

Suite à cette revue de littérature, nous présentons deux tableaux récapitulatifs.

Le tableau 2.1 présente les algorithmes exacts récemment proposés pour résoudre chacun des problèmes. Les différents algorithmes sont nommés : *B&B* – algorithme d'énumération implicite (*branch-and-bound*) ; *B&C* – algorithme d'énumération implicite avec plans coupants (*branch-and-cut*) ; *B&P* – algorithme de génération de colonnes (*branch-and-price*) ; et *B&P&C* – algorithme de génération de colonnes avec plans coupants (*branch-and-price-and-cut*). Nous pouvons remarquer que les algorithmes avec plans coupants sont d'actualité pour l'ensemble des problèmes. De plus, les algorithmes de génération de colonnes semblent bien fonctionner pour le PDPTW.

Les algorithmes heuristiques récents sont présentés dans le tableau 2.2. Les différents algorithmes sont nommés : *Génétique* – algorithme génétique ; *Tabou* – algorithme de recherche avec tabous ; *Recuit simulé* – algorithme avec recuit simulé ; *LNS* – algorithme de recherche à grand voisinage (*large neighborhood search*) ; et *GRASP* – algorithme de recherche avec une adaptation gloutonne (*greedy randomized adaptive search procedure*). On remarque que la recherche à grand voisinage est utilisé dans les algorithmes de l'état de l'art pour résoudre plusieurs variantes des problèmes de tournées de véhicules.

Tableau 2.1 – Algorithmes exacts

	Algorithme				Problème						
	B&B	*B&C*	*B&P*	*B&P&C*	VRP	PDPTW	TSPPDL	PDPL	DTSPMS	PDTSPMS	DVRPMS
Carrabs *et al.* (2007a)	●						●				
Ropke *et al.* (2007)		●				●					
Ropke et Cordeau (2009)				●		●					
Baldacci *et al.* (2010)			●		●						
Cordeau *et al.* (2010)		●					●				
Lusby *et al.* (2010)									●		
Petersen *et al.* (2010)		●							●		
Baldacci *et al.* (2011b)			●		●						
Baldacci *et al.* (2011a)				●	●						
Côté *et al.* (2012a)		●								●	
Alba Martínez *et al.* (2013)		●							●		
Benavent *et al.* (2015)		●						●			
Iori et Riera-Ledesma (2015)		●	●	●							●

Tableau 2.2 – Algorithmes heuristiques

	Algorithme					Problème					
	Génétique	Tabou	Recuit simulé	LNS	GRASP	VRP	PDPTW	TSPPDL	PDPL	DTSPMS	PDTSPMS
Li et Lim (2003)		●	●				●				
Ambrosini *et al.* (2004)					●				●		
Cassani (2004)				●				●			
Bent et Van Hentenryck (2006)			●	●		●	●				
Ropke et Pisinger (2006)				●		●					
Carrabs *et al.* (2007b)				●			●				
Petersen et Madsen (2009)		●	●	●						●	
Gao *et al.* (2011)				●					●		
Li *et al.* (2011)				●			●				
Cheang *et al.* (2012)				●				●			
Côté *et al.* (2012b)				●							●
Vidal *et al.* (2013b)	●					●					
Benavent *et al.* (2015)		●							●		

CHAPITRE 3 ORGANISATION DE LA THÈSE

Cette thèse a pour objectif de formuler mathématiquement et résoudre deux variantes du problème de tournées de véhicules avec cueillettes, livraisons et fenêtres de temps, le PDPTWL et le PDPTWMS. Dans le chapitre 2, nous avons identifié les meilleurs algorithmes exacts et heuristiques qui existent pour résoudre plusieurs variantes du problèmes de tournées de véhicules, soit le PDPTW, le TSPPDL, le PDPL, le DTSPMS, le PDTSPMS et le DVRPMS. Au meilleur de nos connaissances, il n'existe pas d'algorithme exact ou heuristique pour résoudre le PDPTWL et le PDPTWMS.

En premier lieu, nous formulerons le PDPTWL. Nous développerons ensuite trois algorithmes de génération de colonnes avec plan coupants pour résoudre ce problème. Le premier algorithme consiste à incorporer la contrainte LIFO dans le problème maître alors que le second l'incorpore dans le sous-problème. Le troisième algorithme, quant à lui, est une combinaison des deux premiers algorithmes. Chaque algorithme comporte des avantages et des inconvénients : le premier permet de résoudre le sous-problème le plus facile, mais fournit les bornes inférieures les plus faibles ; le second fournit les meilleures bornes inférieures, mais le sous-problème est le plus difficile à résoudre ; et le troisième permet un équilibre entre la difficulté du sous-problème et la qualité de la borne inférieure. Nous présentons et comparons les résultats obtenus avec chaque algorithme dans le chapitre 4. Nous comparons également le coût total de la solution optimale obtenue avec et sans la contrainte LIFO. Des instances comportant jusqu'à 75 requêtes sont résolues avec ces algorithmes. Ces trois algorithmes font partie d'un article publié dans *Transportation Science*. Suite à la publication de l'article, nous avons amélioré notre algorithme d'étiquetage. Neuf instances supplémentaires sont résolues suite à cette amélioration et, en moyenne, les temps de calcul sont réduits.

En deuxième lieu, nous développerons une méthode heuristique pour résoudre des instances de plus grande taille pour le PDPTWL. Cette dernière s'inspire des meilleurs algorithmes heuristiques pour résoudre des variantes du problème de tournées de véhicules, i.e., ceux de Vidal *et al.* (2013b) et de Ropke et Pisinger (2006). Pour ce faire, nous proposons un algorithme génétique hybride. Cet algorithme permet de générer rapidement des tournées initiales avec un GRASP. Puis, les solutions passent au travers d'une phase de recherche locale et les solutions résultantes sont ajoutées à l'ensemble des solutions. Par la suite, nous proposons deux façons de créer des enfants à partir de ces solutions. Pour ce qui est de l'ensemble des solutions, nous nous basons sur le critère de diversification proposé par Vidal *et al.* (2012) afin de conserver de bonnes solutions diversifiées. Le chapitre 5 présente les résultats obtenus avec cette heuristique. Des instances comportant jusqu'à 300 requêtes sont résolues avec cet algorithme. De plus, pour les plus petites instances, celles avec moins de 75 requêtes, l'écart par rapport aux solutions optimales connues est en moyenne de 0,17%.

En troisième lieu, nous formulerons un modèle mathématique pour le PDPTWMS et développerons deux algorithmes de génération de colonnes pour résoudre ce problème. Ces algorithmes sont une extension des algorithmes développés au chapitre 4 pour le PDPTWL. Nous avons adapté la notation de pile pour permettre la représentation de plusieurs piles dans un véhicule. Le premier algorithme consiste à incorporer la contrainte de chargement à plusieurs piles dans le sous-problème alors que le second l'incorpore dans le problème maître. Le premier algorithme permet ainsi d'obtenir de meilleures bornes inférieures, alors que la résolution du sous-problème est plus facile dans le deuxième algorithme. Dans le chapitre 6, nous présentons et comparons les résultats obtenus avec chaque algorithme. Nous comparons également les résultats obtenus lorsque le nombre de piles varie. Des instances comportant jusqu'à 75 requêtes sont résolues.

Finalement, une discussion générale est apportée dans le chapitre 7 et une conclusion est présentée au chapitre 8.

CHAPITRE 4 ARTICLE 1 : BRANCH-PRICE-AND-CUT ALGORITHMS FOR THE PICKUP AND DELIVERY PROBLEM WITH TIME WINDOWS AND LIFO LOADING

Recopié avec permission, M. Cherkesly, G. Desaulniers et G. Laporte, (2014), Branch-Price-and-Cut Algorithms for the Pickup and Delivery Problem with Time Windows and LIFO Loading. *Transportation Science*, publié dans *Articles in Advance*, 30 juin 2014. Copyright (2015), the Institute for Operations Research and the Management Sciences, 5521 Research Park Drive, Suite 200, Catonsville, Maryland 21228 USA.

4.1 Introduction

In pickup and delivery problems (PDP), a set of vehicles based at a depot must carry out pickup and delivery requests for goods (here called items) or passengers between geographically scattered locations, subject to side constraints. This paper focuses on the first case. One-to-one PDPs, which are the topic of this paper, arise whenever each item must be transported between a specified origin and a specified destination (see, e.g., Berbeglia *et al.*, 2007; Parragh *et al.*, 2008a,b). Two common side constraints are time windows (Desaulniers *et al.*, 2002; Kallehauge *et al.*, 2005; Ropke *et al.*, 2007) and delivery priorities (Carrabs *et al.*, 2007a; Cordeau *et al.*, 2010). A time window is an interval of time during which the service at a customer must start. If a vehicle arrives before the beginning of time window, it must then wait until its opening time in order to start the service. If a vehicle arrives after the closing of the time window, then the service cannot take place. Loading priorities impose rules on the order in which items can be loaded in and unloaded from the vehicles.

This paper focuses on the pickup and delivery problem with time windows and LIFO (last-in-first-out) constraints (PDPTWL). The LIFO policy means that when a pickup point is visited, the collected item is put on top of a stack and can be delivered only when it is in this position. To illustrate, let 0, i^+ and i^- denote respectively the depot, as well as the pickup and the delivery points associated with request i. Figure 4.1a depicts a path that respects the LIFO policy, whereas Figure 4.1b depicts a path that does not. We consider an unlimited fleet of identical capacitated vehicles and a set of requests. A request is defined by the transportation of an item having an associated load, from a pickup point to a delivery point, and has a specified time window. A vehicle route is feasible if (i) the load onboard a vehicle never exceeds its capacity, (ii) the time windows are respected, (iii) every pickup point is visited before its corresponding delivery point and (iv) the LIFO policy is respected.

Two types of cost are considered : a fixed cost for each vehicle used in the solution and a distance-related variable cost. The PDPTWL consists of determining a set of least cost feasible routes.

The PDPTWL arises in the transportation of heavy or dangerous material where handling should be avoided, or when transporting livestock. To our knowledge, the PDPTWL has not been previously studied, apart from a distance-constrained pickup and delivery problem with LIFO constraints which was solved heuristically (Cheang *et al.*, 2012). However, algorithms have been proposed for related problems, namely the pickup and delivery problem with time windows (PDPTW) and the traveling salesman problem with pickup, delivery and LIFO constraints (TSPPDL).

Ropke *et al.* (2007) have developed a branch-and-cut algorithm for the PDPTW. Several families of valid inequalities were proposed and tested, two of which seem to perform better to reduce the integrality gap : the fork inequalities and the reachability inequalities. Ropke and Cordeau (2009) have proposed a branch-price-and-cut algorithm for the same problem in which the objective first minimizes the number of vehicles and then the total traveled distance. It makes use of several families of valid inequalities : infeasible path inequalities, rounded capacity inequalities, 2-path inequalities, precedence inequalities and strenghtened precedence inequalities. The 2-path inequalities seem to exhibit the best performance. Finally, Baldacci *et al.* (2011a) have designed two exact algorithms relying on column generation and variable fixing based on reduced cost for the same problem. Their algorithms yield the best known results for it.

Two exact algorithms were proposed for the TSPPDL. Carrabs *et al.* (2007a) first developed an additive branch-and-bound algorithm where the additive process computes lower bounds. Cordeau *et al.* (2010) later proposed a branch-and-cut algorithm. These authors introduced valid inequalities for this problem : incompatible predecessor and successor inequalities, hamburger inequalities, and incompatible path inequalities. They concluded that for the tested instances, hamburger inequalities and incompatible path inequalities produce better lower

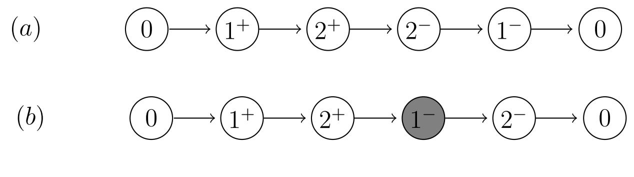


Figure 4.1 – Vehicle routes in which (a) the LIFO policy is respected, (b) the LIFO policy is not respected, because the item picked up at 1^+ cannot be delivered to 1^- without first removing from the vehicle the item picked up at 2^+.

bounds than the incompatible predecessor and successor inequalities.

Battarra *et al.* (2010) present models and exact algorithms based on branch-and-cut for a version of the TSPPDL in which the LIFO rule can be violated, but rehandling costs are imposed according to three different handling policies. Erdoğan *et al.* (2012) have later developed powerful tabu search metaheuristics for the same problem.

The main objective of this paper is to develop, for the first time, exact algorithms for the PDPTWL. More specifically, we propose three related branch-price-and-cut algorithms. These are branch-and-cut algorithms in which the linear relaxations are solved by means of column generation, a decomposition algorithm that alternates between the solution of a master problem and that of a subproblem called the pricing problem. Column generation has been widely applied to various constrained vehicle routing and crew scheduling problems (Desaulniers *et al.*, 2005) and provides some of the best known results for the PDPTW (Ropke and Cordeau, 2009; Baldacci *et al.*, 2011a). The first algorithm developed in this paper adds the LIFO constraints to the master problem, whereas the second incorporates them into the pricing problem. Finally, the third algorithm combines the best features of the first two algorithms. Computational results on instances derived from known PDPTW instances are reported and show that the second and third algorithms perform the best, with a slight advantage for the hybrid algorithm for certain instance classes.

The remainder of this paper is structured as follows. Section 4.2 proposes a three-index formulation for the PDPTWL. Section 4.3 presents a branch-price-and-cut algorithm in which the LIFO constraints are added inside the master problem. Different pricing problems that can be adapted for all the algorithms are discussed. Section 4.4 describes the algorithm incorporating the LIFO constraints into the pricing problem and Section 4.5 presents the hybrid algorithm. Computational results are reported in Section 4.6. This is followed by conclusions in Section 4.7.

4.2 Mathematical Formulation

We now introduce the notation used in our models. A three-index formulation is then introduced, including two ways of formulating the LIFO constraints.

4.2.1 Notation

Let n denote the number of requests. The PDPTWL can be defined on a directed graph $G = (N, A)$, where $N = \{0, 1, ..., 2n, 2n + 1\}$ is the set of nodes and A is the set of arcs. Nodes 0 and $2n+1$, called the origin and destination nodes, represent two copies of the depot appearing at the two ends of a path. The subsets $P = \{1, ..., n\}$ and $D = \{n+1, ..., 2n\}$ represent the sets of pickup and delivery nodes, respectively. Each request i is associated with a pickup node $i \in P$ and a delivery node $n + i \in D$ (denoted i^+ and i^- in Section 4.1). The set of arcs A can be described by four types of arcs (i, j) : (i) from the origin depot to the pickup nodes, i.e. $i = 0$ and $j \in P$, (ii) from the delivery nodes to the destination depot, i.e. $i \in D$ and $j = 2n + 1$, (iii) from a pickup node to its corresponding delivery node or to another pickup node, i.e. $i \in P$ and $j \in P \cup \{n+i\}$, and (iv) from a delivery node to another node except its corresponding pickup node, i.e. $i \in D$ and $j \in D \cup P \backslash \{i-n\}$. Note that there are no arcs $(i, n + j), i \in P, j \in P \backslash \{i\}$, because their use would violate the LIFO policy.

For each node $i \in N$, q_i represents the load picked up or delivered at this node, with $q_i = 0$ if $i \in \{0, 2n+1\}$, $q_i > 0$ if $i \in P$ and $q_i = -q_{i-n}$ if $i \in D$. Let s_i be the service duration at node i, with $s_i > 0$ if $i \in P \cup D$ and $s_i = 0$ if $i \in \{0, 2n + 1\}$. A time window $[\underline{w}_i, \bar{w}_i]$ is associated with each node $i \in P \cup D$, where $\underline{w}_i$ and $\bar{w}_i$ represent respectively the earliest and the lastest time at which service at node i must begin. Unconstraining time windows are also imposed on the origin and destination nodes 0 and $2n + 1$. An unrestricted set K of identical vehicles with capacity Q is available. A nonnegative travel cost c_{ij} and a nonnegative travel time t_{ij} are associated with each arc $(i, j) \in A$. The cost of each arc leaving the origin node, i.e. an arc $(0, j)$ such that $j \in P$, also includes a vehicle fixed cost. This cost is assumed to be large, leading to first minimizing the number of vehicles, and then the total traveled distance. The triangle inequality is assumed to be respected for travel costs and travel times.

4.2.2 A Three-Index Formulation

The PDPTWL can be formulated as a three-index model similar to that proposed by Ropke and Cordeau (2009). For each arc $(i, j) \in A$ and each vehicle $k \in K$, let x_{ij}^k be a binary variable equal to 1 if and only if vehicle k uses arc (i, j). For each node $i \in N$ and each vehicle $k \in K$, let T_i^k represent the time at which vehicle k begins service at node i, and let Q_i^k be the load of vehicle k upon leaving node i. To enforce the LIFO policy, it is necessary to define, for each request $i \in P$, the set Φ_i of subsets $S \subset N$ such that $0, 2n+1, i, n+i \notin S$ and there exists a request j such that $j \notin S, n + j \in S$ or $j \in S, n + j \notin S$. The PDPTWL can then be formulated as follows :

$$\text{minimize} \qquad \sum_{k \in K} \sum_{(i,j) \in A} c_{ij} x_{ij}^k \tag{4.1}$$

$$\text{subject to} \qquad \sum_{k \in K} \sum_{j \in N} x_{ij}^k = 1, \quad \forall i \in P, \tag{4.2}$$

$$\sum_{j\in N} x_{ij}^k - \sum_{j\in N} x_{n+i,j}^k = 0, \quad \forall i \in P, k \in K, \tag{4.3}$$

$$\sum_{j\in N} x_{0j}^k = 1, \quad \forall k \in K, \tag{4.4}$$

$$\sum_{j\in N} x_{ji}^k - \sum_{j\in N} x_{ij}^k = 0, \quad \forall i \in P \cup D, k \in K, \tag{4.5}$$

$$\sum_{i\in N} x_{i,2n+1}^k = 1, \quad \forall k \in K, \tag{4.6}$$

$$\sum_{(i,j)\in A|j\in S} x_{ij}^k + \sum_{(l,j)\in A|l,j\in S} x_{lj}^k + \sum_{(j,n+i)\in A|j\in S} x_{j,n+i}^k \leq |S|,$$
$$\forall i \in P, S \in \Phi_i, k \in K, \tag{4.7}$$

$$Q_j^k \geq (Q_i^k + q_j)x_{ij}^k, \quad \forall (i,j) \in A, k \in K, \tag{4.8}$$

$$\max\{0, q_i\} \leq Q_i^k \leq \min\{Q, Q + q_i\}, \quad \forall i \in N, k \in K, \tag{4.9}$$

$$T_j^k \geq (T_i^k + s_i + t_{ij})x_{ij}^k, \quad \forall (i,j) \in A, k \in K, \tag{4.10}$$

$$\underline{w}_i \leq T_i^k \leq \bar{w}_i, \quad \forall i \in N, k \in K, \tag{4.11}$$

$$T_i^k + t_{i,n+i} + s_i \leq T_{n+i}^k, \quad \forall i \in P, \tag{4.12}$$

$$x_{ij}^k \in \{0,1\}, \quad \forall (i,j) \in A, k \in K. \tag{4.13}$$

The objective function (4.1) minimizes the total cost. Constraints (4.2) and (4.3) ensure that each pickup node is visited exactly once and that for every request the pickup and delivery nodes are visited by the same vehicle. Constraints (4.4)–(4.6) define a path structure for every vehicle : constraints (4.4) and (4.6) ensure that each route starts and ends at the depot, while constraints (4.5) are flow conservation constraints for each node $i \in P \cup D$. The LIFO policy is imposed through constraints (4.7) which were initially proposed by Cordeau *et al.* (2010) for the TSPPDL. Constraints (4.8) and (4.9) compute the load variables according to the arcs used in the solution and ensure that the vehicle capacity is respected. Similarly, constraints (4.10) and (4.11) compute the time variables according to the active arcs and ensure that the time windows are respected. Constraints (4.12) impose that for each request i the pickup node is visited before its corresponding delivery node. The model is non-linear because of constraints (4.8) and (4.10) but can easily be linearized (Ropke *et al.*, 2007).

It is interesting to note that constraints (4.7) can be reformulated with LIFO-infeasible path inequalities. These inequalities are the infeasible path inequalities proposed for the PDPTW (Cordeau, 2006; Ropke *et al.*, 2007; Ropke and Cordeau, 2009) but are applied to LIFO-infeasible paths. Let $\mathcal{R}$ be the set of infeasible paths with respect to the LIFO constraints that do not begin at the origin node nor end at the destination node, and let $N(R)$ be the set of nodes in path $R \in \mathcal{R}$. The LIFO-infeasible path cuts are then expressed as

$$\sum_{k\in K}\left(\sum_{\mu=1}^{\rho-1} x^k_{i_\mu,i_{\mu+1}}\right) \leq |N(R)| - 2, \ \ \forall R \in \mathcal{R}, \tag{4.14}$$

where $R = (i_1, ..., i_\rho)$ is a LIFO-infeasible path. Let $\mathcal{R}' \subseteq \mathcal{R}$ be a subset of LIFO-incompatible paths such that (i) the first node is a pickup node and the last node is a delivery node i.e. $i_1 \in P$, $i_\rho \in D$, (ii) for each request $j \in P \cap \{i_2, ..., i_{\rho-1}\}$, $n + j \in \{i_2, ..., i_{\rho-1}\}$, and (iii) the LIFO policy is respected on the path $\bar{R} = (i_2, ..., i_{\rho-1})$. Figure 4.2 depicts a path $R \in \mathcal{R}'$ where $i_1 = 2^+$ and $i_\rho = 1^-$.

Proposition 4.2.1. *For a given LIFO-infeasible solution, there always exists a LIFO-infeasible path R such that $R \in \mathcal{R}'$.*

Proof. By definition, the LIFO policy implies that when a pickup node is visited, the collected items are put on top of a stack. These items can be delivered only when they are in this position. A path R is LIFO-infeasible if there exists a request $i \in P$ such that node $n + i$ is visited on the path when the item corresponding to another request $j \in P$ such that $j \neq i$ is on top of the stack. Now consider the path R going from j to $n + i$. This path is LIFO-infeasible and has the following property : $R = \{i_1 = j, ..., i_\rho = n+i\}$, $i, n+j \notin R$ and all the requests visited between j and $n + i$ are completed. As a result, for a given LIFO-infeasible solution, there always exists a LIFO-infeasible path $R \in \mathcal{R}'$. □

Consequently, the LIFO policy can be modeled by considering inequalities (4.14) only for the paths in $\mathcal{R}'$. The following proposition states how these inequalities can be strenghtened. It relies on the set $A(R) = \{(i, j) \in A | i, j \in N(R), i \neq i_\rho, j \neq i_1\}$ which contains all the arcs connecting the nodes of path R, except those ending in i_1 or beginning in i_ρ. Note that all the arcs $(i_\mu, i_{\mu+1})$, $\mu = 1, ..., \rho - 1$ used in R belong to $A(R)$.

Proposition 4.2.2. *The inequalities*

$$\sum_{k\in K}\sum_{(l,j)\in A(R)} x^k_{lj} \leq |N(R)| - 2, \ \ \forall R \in \mathcal{R}', \tag{4.15}$$

are valid for the feasible solution set of model (4.1)–(4.13).

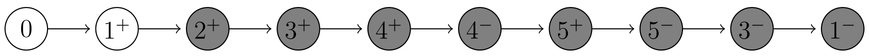


Figure 4.2 – The shaded nodes form a path $R = \{i_1 = 2^+, i_2 = 3^+, ..., i_{\rho-1} = 3^-, i_\rho = 1^-\}$ such that $R \in \mathcal{R}'$.

Proof. By contradiction, suppose that the LIFO policy is satisfied for a feasible solution, but that at least one constraint (4.15) is violated, namely, for the path $R = \{i_1, ..., i_\rho\} \in \mathcal{R}'$. Because exactly one vehicle enters and exits each node in $N(R)$ in a feasible solution, it follows that $\sum_{k\in K} \sum_{(l,j)\in A(\bar{R})} x^k_{lj} \leq |N(\bar{R})| - 1 = |N(R)| - 3$, $\sum_{k\in K} \sum_{(i_1,j)\in A|j\in N(\bar{R})} x^k_{i_1,j} \leq 1$, and $\sum_{k\in K} \sum_{(j,i_\rho)\in A|j\in N(\bar{R})\cup\{i_1\}} x^k_{j,i_\rho} \leq 1$ where $\bar{R} = R\backslash\{i_1, i_\rho\}$. If one of the constraints is violated, then $\sum_{k\in K} \sum_{(l,j)\in A(\bar{R})} x^k_{lj} = |N(R)| - 3$ and $\sum_{k\in K} \sum_{(i_1,j)\in A|j\in N(\bar{R})} x^k_{i_1,j} = \sum_{k\in K} \sum_{(j,i_\rho)\in A|j\in N(\bar{R})\cup\{i_1\}} x^k_{j,i_\rho} = 1$. As a result, this solution contains a path starting at i_1, visiting every node in $\bar{R}$, and ending at i_ρ. But this is impossible because path $R \in \mathcal{R}'$ would then be LIFO-infeasible. Therefore, for this solution, there exists a path containing a path $\hat{R} \neq R$ starting at i_1, ending at i_ρ, and visiting every node in $\bar{R}$. Because $\hat{R} \in \mathcal{R}'$, this path is LIFO-infeasible, contradicting the assumption that the LIFO policy is satisfied in the feasible solution. □

Ropke and Cordeau (2009) have shown that for the PDPTW, the three-index formulation yields a weaker linear relaxation lower bound than a set partitioning formulation. Furthermore, a branch-price-and-cut algorithm based on the set partitioning formulation can solve more instances than a branch-and-cut algorithm based on the three-index formulation. As a result, we have decided to investigate different set partitioning formulations for the PDPTWL and to develop ad hoc branch-price-and-cut algorithms.

In this paper, one of the branch-price-and-cut algorithms uses LIFO constraints imposed for every vehicle similarly to constraints (4.7), while one of the other algorithms enforces the LIFO policy with an equivalent form of constraints (4.15). Using constraints (4.15) instead of constraints (4.7) yields an exact, but weaker formulation.

4.3 A Branch-Price-and-Cut Algorithm with LIFO-infeasible Paths

Our first branch-price-and-cut algorithm enforces the LIFO policy in the master problem. This is implemented by solving the pricing problem as for the PDPTW and adding LIFO inequalities in the master problem. In this case, the pricing problem can generate LIFO-infeasible paths. These paths will be discarded from the final solution by adding cuts. In this section, we first present a set partitioning formulation for the PDPTWL. Path relaxations and labeling algorithms for the corresponding pricing problem are then discussed. We finally describe valid inequalities for the PDPTWL, as well as branching strategies.

4.3.1 Set Partitioning Formulation

Let Ω denote the set of all feasible paths satisfying constraints (4.3)–(4.6) and (4.8)–(4.13). Let c_r be the cost of path $r \in \Omega$, let a_{ir} be a constant indicating the number of times node $i \in P$ is visited in this path, and let b^r_{ij} be a constant equal to the number of times arc $(i,j) \in A$ is used in this path. Defining y_r as a binary variable equal to 1 if and only if path $r \in \Omega$ is used in the solution, the PDPTWL can be formulated as follows :

$$\text{minimize} \qquad \sum_{r\in\Omega} c_r y_r \tag{4.16}$$

$$\text{subject to} \qquad \sum_{r\in\Omega} a_{ir} y_r = 1, \qquad \forall i \in P, \tag{4.17}$$

$$\sum_{r\in\Omega} \sum_{(l,j)\in A(R)} b^r_{lj} y_r \leq |N(R)| - 2, \qquad \forall R \in \mathcal{R}', \tag{4.18}$$

$$y_r \in \{0,1\}, \qquad \forall r \in \Omega. \tag{4.19}$$

The objective function (4.16) minimizes the total cost. Constraints (4.17) ensure that every request is completed exactly once, while constraints (4.18), which are equivalent to (4.15), ensure that the LIFO policy is enforced. In practice, the model defined by (4.16)–(4.19) contains a very large number of variables. Consequently, we use column generation to solve its linear relaxation which is called the master problem in this context. The restricted master problem contains a subset of the variables (columns) and is solved by linear programming to yield a primal and a dual solution. The pricing problem is then solved to identify columns with a negative reduced cost with respect to the dual solution of the restricted master problem. Whenever such columns are identified, they are added to the restricted master problem before starting a new iteration. Otherwise, the process stops with an optimal solution to the master problem.

Associating the dual variables α_i and β_R with constraints (4.17) and (4.18), respectively, the pricing problem for the PDPTWL is defined as

$$\text{minimize} \qquad \sum_{(i,j)\in A} \bar{c}_{ij} x_{ij} \tag{4.20}$$

$$\text{subject to} \qquad \text{constraints (4.3)–(4.6), (4.8)–(4.13)}, \tag{4.21}$$

where index k is dropped because all the vehicles are identical, and therefore the pricing problem is the same for every vehicle. For each path $R \in \mathcal{R}'$, let γ^R_{ij} be a parameter taking the value 1 if arc $(i,j) \in A(R)$ and 0 otherwise. The reduced cost $\bar{c}_{ij}$ of arc $(i,j) \in A$ is then defined as

$$\bar{c}_{ij} = \begin{cases} c_{ij} - \alpha_i - \sum_{R \in \mathcal{R}'} \gamma_{ij}^R \beta_R, & \forall i \in P, \\ c_{ij} - \sum_{R \in \mathcal{R}'} \gamma_{ij}^R \beta_R, & \forall i \in N \backslash P. \end{cases} \tag{4.22}$$

The pricing problem thus corresponds to an elementary shortest path problem with pickup and delivery, time window and capacity constraints. It can be solved by dynamic programming as will be discussed in Section 4.3.2. In the proposed branch-price-and-cut algorithm, the master problem is not solved with constraints (4.18) because adding these to the master problem slows down the computational process. In fact, we solve the linear programming relaxation of formulation (4.16)–(4.17) and (4.19), which may produce LIFO-infeasible solutions. To enforce the LIFO policy, we first add incompatible predecessor and successor inequalities. Even though these inequalities are, in general, not sufficient to restore LIFO-feasibility, they are added first because they are easy to separate. When no more of these inequalities can be found i.e. when the solution respects all these inequalities but remains LIFO-infeasible, we add violated constraints (4.18) which are separated through an exact enumeration procedure. For every path in a given optimal solution of the master problem, several of these constraints can be violated. A sequential search is then carried out on each path, and the first identified violated inequality is added to the master problem.

Incompatible Successor and Predecessor Cuts.

The incompatible predecessor and successor cuts were introduced by Cordeau *et al.* (2010) in the context of the TSPPDL. Let $i \prec j$ denote the fact that node i precedes node j in a path. For each pair of nodes $i, j \in P$, if arc $(i, j) \in A$ is used in a feasible solution, the LIFO policy is respected if and only if $0 \prec i, j \prec n + j \prec n + i \prec 2n + 1$. This implies that the successor of node $n + j$ will either be node $n + i$, or another pickup node different from i and j. In this case, the set of successors of node $n + j$ given that arc $(i, j) \in A$ is used is $\sigma_{n+j}(i, j) = \{n + i\} \cup (P \backslash \{i, j\})$, and the incompatible successor inequalities are written as

$$\sum_{r \in \Omega} \left(b_{ij}^r + \sum_{l \notin \sigma_{n+j}(i,j)} b_{n+j,l}^r \right) y_r \leq 1, \quad \forall i, j \in P | (i, j) \in A. \tag{4.23}$$

These inequalities imply that if an arc $(i, j) \in A$ with $i, j \in P$ is used in a feasible solution, then no arc $(n + j, l) \in A$ such that $l \notin \sigma_{n+j}(i, j)$ can be used.

In a symmetric way, for each $i, j \in P$ such that arc $(n + i, n + j) \in A$ is used in a feasible solution, the path must satisfy $0 \prec j \prec i \prec n + i, n + j \prec 2n + 1$. The predecessor of node i will either be node j or a delivery node different from $n + i$ and $n + j$. The set of compatible predecessors at node i given that arc $(n+i, n+j) \in A$ is used can be expressed as $\pi_i(n+i, n+j) = \{j\} \cup (D \backslash \{n+i, n+j\})$. If arc $(n+i, n+j) \in A$ with $i, j \in P$ is used in a feasible solution, then no arcs $(l, i) \in A$ such that $l \notin \pi_i(n+i, n+j)$ can be used. The incompatible predecessor inequalities are given by

$$\sum_{r \in \Omega} \left(b^r_{n+i,n+j} + \sum_{l \notin \pi_i(n+i,n+j)} b^r_{li} \right) y_r \leq 1, \quad \forall i, j \in P | (i,j) \in A. \tag{4.24}$$

Because there exists only a quadratic number of inequalities (4.23) and (4.24), their separation can easily be achieved by enumeration. For a given optimal solution of the master problem, the flow on each arc $(i, j) \in A$ is computed as $\sum_{r \in \Omega} b^r_{ij} y_r$. Then, all the inequalities (4.23) and (4.24) violated by this solution are added to the master problem, thus introducing dual variables in the objective function (4.20) of the pricing problem (for more details, see Desaulniers *et al.*, 2011).

4.3.2 Path Relaxations and Labeling Algorithms

The pricing problem is an elementary shortest path problem with pickup and delivery, time window and capacity constraints, which is known to be NP-hard (Sol, 1994). A labeling algorithm can be used for its solution (see Irnich and Desaulniers, 2005). A label is a vector representing a partial path starting at the origin node and ending at a given node η. It stores information about the partial path such as its cumulated reduced cost, and the start of service time in the last node. Each element stored is called a component. Starting from an initial label E_0 at the origin node, a labeling algorithm propagates labels toward the destination node with extension functions. To avoid enumerating all feasible paths, some labels are eliminated through dominance tests.

To speed up the algorithm, the pricing problem can be relaxed by allowing cycles in paths, that is, a request can be completed more than once. These relaxations usually yield weaker lower bounds. Because paths with cycles cannot be part of a feasible integer solution, branching ensures that the final solution contains only elementary paths. Many relaxations of the elementary shortest path problem with pickup and delivery, time windows, and capacity exist. We will present two such relaxations. The first allows many cycles to occur. The second is the ng-path relaxation introduced by Baldacci *et al.* (2010, 2011b) which allows only some cycles to occur. Sections 4.3.2, 4.3.2, and 4.3.2 focus on the elementary version of the pricing problem, on the complete relaxation of the elementarity constraints, and on the ng-path relaxation, respectively. We describe a labeling algorithm for each of these problems.

Elementary Shortest Path Problem with Pickup and Delivery, Time Windows, and Capacity.

The first version of the constrained shortest path problem respects the elementarity constraints. The ideas presented in this section were proposed by Ropke and Cordeau (2009) for the PDPTW. Each label stores the following components :

— η : the node of the label,
— t : start of service time at node η,
— ι : load of vehicle after visiting node η,
— c : cumulated reduced cost,
— $\mathcal{O}$: the set of onboard requests,
— U : the set of unreachable requests.

A request i is said to be onboard if it is still in the vehicle, i.e. its corresponding pickup node has been visited but not its corresponding delivery node. A request $i \in P$ is said to be unreachable if i has already been visited on the partial path, or if traveling directly from η to i violates the time window at node $i \in P$. For a given label E, let $R(E)$ represent its correponding partial path. Then

$$U(E) = \{i \in P | i \in R(E)\} \cup \{i \in P | t(E) + s_{\eta(E)} + t_{\eta(E),i} > \bar{w}_i\}. \tag{4.25}$$

Given a label E, its extension along arc $(\eta(E), j) \in A$ is allowed only if it satisfies one of the following three conditions :

$$0 < j \leq n \quad \text{and} \quad j \notin U(E), \tag{4.26}$$
$$n < j \leq 2n \quad \text{and} \quad j - n \in \mathcal{O}(E), \tag{4.27}$$
$$j = 2n + 1 \quad \text{and} \quad \mathcal{O}(E) = \emptyset. \tag{4.28}$$

Condition (4.26) ensures that if j is a pickup node, then it must not have been previously visited and it must be reachable with respect to the time windows. Condition (4.27) stipulates that if j is a delivery node, then it must be associated with an onboard request. Finally, condition (4.28) ensures that if j is the destination node, then all the visited requests on the path must be completed. These conditions ensure that each request will be completed at most once on any given path. When these conditions are respected, a new label E' is created and the components are set as follows :

$$\eta(E') = j, \tag{4.29}$$
$$t(E') = \max\{\underline{w}_j, t(E) + s_{\eta(E)} + t_{\eta(E),j}\}, \tag{4.30}$$

$$\iota(E') = \iota(E) + q_j, \tag{4.31}$$

$$c(E') = c(E) + \bar{c}_{\eta(E),j}, \tag{4.32}$$

$$\mathcal{O}(E') = \begin{cases} \mathcal{O}(E) \cup \{j\} & \text{if } j \in P, \\ \mathcal{O}(E) \backslash \{j-n\} & \text{if } j \in D, \end{cases} \tag{4.33}$$

$$U(E') = \begin{cases} U(E) \cup \{j\} \cup \{i \in P | t(E') + s_{\eta(E')} + t_{\eta(E'),i} > \bar{w}_i\} & \text{if } j \in P, \\ U(E) \cup \{i \in P | t(E') + s_{\eta(E')} + t_{\eta(E'),i} > \bar{w}_i\} & \text{if } j \in D. \end{cases} \tag{4.34}$$

Label E' is kept if it respects the time windows and capacity constraints, that is, if

$$t(E') \leq \bar{w}_{\eta(E')}, \tag{4.35}$$

$$\iota(E') \leq Q. \tag{4.36}$$

A label E_1 dominates a label E_2 if

$$\eta(E_1) = \eta(E_2), \tag{4.37}$$

$$t(E_1) \leq t(E_2), \tag{4.38}$$

$$c(E_1) \leq c(E_2), \tag{4.39}$$

$$\mathcal{O}(E_1) \subseteq \mathcal{O}(E_2), \tag{4.40}$$

$$U(E_1) \subseteq U(E_2). \tag{4.41}$$

All dominated labels are removed except when two labels dominate each other, in which case one of them is kept. This dominance condition was proposed by Ropke and Cordeau (2009) in the context of the PDPTW. These authors showed that it constitutes a valid dominance criterion when the triangle inequality is respected by the reduced costs of the arcs at the delivery nodes. However, the definition of $\bar{c}_{ij}$ in formula (4.22) does not necessarily ensure that $\bar{c}_{ij} + \bar{c}_{jk} \geq \bar{c}_{ik}$ if j is a delivery node. In this situation, the authors propose a procedure to transform an arbitrary cost matrix into a cost matrix satisfying the delivery triangle inequality. Here we apply the same procedure.

Shortest Path Problem with Pickup and Delivery, Time Windows, and Capacity.

The second version of the constrained shortest path problem allows paths to contain cycles under the following two conditions :

1. a pickup cannot be performed again before the corresponding delivery has been completed ;
2. the precedence constraints for every request must be respected.

In this version of the algorithm, the label components are η, t, ι, c and $\mathcal{O}$, and component U is unnecessary. The extension of label E along arc $(\eta(E), j)$ proceeds as follows : if E and j satisfy condition (4.27), (4.28) or

$$0 < j \leq n \quad \text{and} \quad j \notin \mathcal{O}(E), \tag{4.42}$$

then a label E' is created using the extension functions (4.29)–(4.33). Condition (4.42) replaces condition (4.26), and allows cycles to occur while forbidding to pickup the same request twice without delivering it in the meantime. The resulting label E' is kept if it satisfies conditions (4.35) and (4.36). If the delivery triangle inequality holds, then the dominance criterion is as follows : a label E_1 dominates a label E_2 if conditions (4.37)–(4.40) are respected.

Shortest ng-Path Problem with Pickup and Delivery, Time Windows, and Capacity.

Baldacci *et al.* (2010, 2011b) have introduced another path relaxation, called ng-path, which allows some cycles. Let N_i represent a set of neighbor requests for each request $i \in P$. If $i \in D$, we define $N_i = N_{i-n}$. Then a cycle $(0, ..., j, ..., n+j, ..., i, ..., j)$ can occur if there exists a request $j \in P$ such that $i \in P \cup D$ and $j \notin N_i$. The neighborhood of each request i can contain a maximum of λ neighbors, i.e. $|N_i| \leq \lambda$. For our problem, we define the neighborhood of each request $i \in P$ as follows. We first compute the distance, $\text{dist}(i, j)$, from request i to request j :

$$\text{dist}(i, j) = \min\{t_{ij} + s_i, t_{i,n+j} + s_i, t_{n+i,j} + s_{n+i}, t_{n+i,n+j} + s_{n+i}\}. \tag{4.43}$$

Then, for each request $i \in P$, the λ nearest requests such that the time windows and capacity constraints could be respected if both requests were completed by the same vehicle are added to the set N_i. Request $i \in P$ is also added to its own neighborhood, i.e. $i \in N_i$.

The idea behind this relaxation is that when request $j \in P$ does not belong to the neighborhood of request $i \in P$, then returning to j after visiting i creates a detour. Therefore, a path containing such a cycle should not be part of a master problem solution.

In a labeling algorithm, the ng-paths are handled as follows. For a given label E, let $R(E) = (0, i_1, i_2, ..., i_\rho = \eta(E))$ represent the partial path corresponding to this label. Then, let $ng(E)$

be the set of requests such that a cycle is not allowed on the extension of the label, i.e.,

$$ng(E) = \left\{ i \in P \,\middle|\, \exists \mu = 1, ..., \rho \text{ such that } n + i = i_\mu \text{ and } i \in \bigcap_{\nu=\mu}^{\rho} N_{i_\nu} \right\}. \tag{4.44}$$

The dominance condition and the extension functions are identical to those of the elementary shortest path problem with pickup and delivery, time window, and capacity constraints, except that the set of unreachable requests U is defined differently. Let U_{ng} be the set of unreachable requests according to the ng-path relaxation. Then for a label E

$$U_{ng}(E) = ng(E) \cup \{i \in P | t(E) + s_{\eta(E)} + t_{\eta(E),i} > \bar{w}_i\}. \tag{4.45}$$

With this version, the components of a label are η, t, ι, c, $\mathcal{O}$ and U_{ng}. The extension of a label E along arc $(\eta(E), j) \in A$ is allowed if one of the three following conditions is respected : (4.27), (4.28) or

$$0 < j \leq n \quad \text{and} \quad j \notin U_{ng}(E) \quad \text{and} \quad j \notin \mathcal{O}(E). \tag{4.46}$$

If the previous conditions are satisfied, the components of the new label E' are set according to equations (4.29)–(4.33), and

$$U_{ng}(E') = ng(E') \cup \{i \in P | t(E') + s_{\eta(E')} + t_{\eta(E'),i} > \bar{w}_i\}. \tag{4.47}$$

Label E' is kept if conditions (4.35) and (4.36) are satisfied.

If the delivery triangle inequality holds, then the dominance criterion is : a label E_1 dominates a label E_2 if it respects conditions (4.37)–(4.40) and

$$U_{ng}(E_1) \subseteq U_{ng}(E_2). \tag{4.48}$$

It is now clear that $U_{ng}(E) \subseteq U(E)$ for a given label E, which leads to the following result. If $\lambda < |P|$, the ng-path relaxation allows for more dominance but if $\lambda = |P|$, the ng-path will solve the shortest path problem with elementarity constraints.

4.3.3 Valid Inequalities

We now present valid inequalities commonly used to solve the PDPTW and applicable to the PDPTWL. These are 2-path cut inequalities, rounded capacity inequalities, and subset-row inequalities. We also present a family of cuts based on the branching on the number of vehicles. These inequalities are added within the master problem. For the sake of conciseness, we omit the discussion about the impact on the reduced cost of adding such inequalities (see Desaulniers *et al.*, 2011, for details). These inequalities will be used in each of our three proposed branch-price-and-cut algorithms.

To impose integrality on the number of vehicles, we use a family of inequalities inspired from the branching on the number of vehicles (Desrochers *et al.*, 1992). If the number of vehicles is fractional, two branches are created : $\sum_{r\in\Omega} y_r \leq \lfloor \sum_{r\in\Omega} \tilde{y}_r \rfloor$, and $\sum_{r\in\Omega} y_r \geq \lceil \sum_{r\in\Omega} \tilde{y}_r \rceil$, where $(\tilde{y}_1, \ldots, \tilde{y}_{|\Omega|})$ is the computed fractional-valued solution of the master problem. Because we first minimize the number of vehicles, the number of vehicles used in the solution of the master problem is a lower bound on the number of vehicles used in the optimal integer solution. In this case, the inequality

$$\sum_{r\in\Omega} y_r \geq \left\lceil \sum_{r\in\Omega} \tilde{y}_r \right\rceil \tag{4.49}$$

is added to the master problem and replaces the branching on the number of vehicles.

Kohl *et al.* (1999) have introduced 2-path cuts to solve the VRPTW. These were later shown to be valid for the PDPTW (Ropke and Cordeau, 2009). Let $S \subseteq P \cup D$ be a subset of nodes that cannot be served by a single vehicle. Then the inequality

$$\sum_{r\in\Omega} \sum_{(i,j)\in A | i\in S, j\notin S} b^r_{ij} y_r \geq 2 \tag{4.50}$$

is valid. Identifying a subset of nodes that cannot be served by a single vehicle means determining whether the corresponding traveling salesman problem with pickup and delivery, and time windows is feasible, which is an NP-complete problem. In practice, the separation of this class of inequalities can often be achieved by means of a greedy heuristic (Ropke and Cordeau, 2009).

The rounded capacity inequalities are often used for the VRP, the VRPTW, and the PDPTW (Naddef and Rinaldi, 2002; Cordeau, 2006; Ropke *et al.*, 2007). Let $S \subseteq P \cup D$ be a subset of nodes and let $\xi(S)$ be a lower bound on the number of vehicles needed to visit all nodes in S. The inequality

$$\sum_{r\in\Omega} \sum_{(i,j)\in A | i\in S, j\notin S} b^r_{ij} y_r \geq \xi(S) \tag{4.51}$$

is valid whenever $\xi(S) = \max\{1, \lceil q(\pi(S))/Q \rceil, \lceil -q(\sigma(S))/Q \rceil\}$, where $\pi(S) = \{i \in P | i \notin S, n+i \in S\}$ and $\sigma(S) = \{n+i \in D | i \in S, n+i \notin S\}$ respectively denote the set of predecessors and the set of successors of S. The lower bound on the load of the vehicles entering

S is set as $q(\pi(S)) = \sum_{i\in\pi(S)} q_i$, and the lower bound on the load of the vehicles leaving S is $q(\sigma(S)) = \sum_{n+i\in\sigma(S)} q_i$. These inequalities are separated by means of an enumerative procedure.

The subset-row inequalities were introduced by Jepsen *et al.* (2008) for the VRPTW and are a special case of the clique inequalities. These inequalities are the rank-1 Chvátal-Gomory inequalities defined as

$$\sum_{r\in\Omega} \left\lfloor \frac{1}{\chi} \sum_{i\in S} a_{ir} \right\rfloor y_r = \left\lfloor \frac{|S|}{\chi} \right\rfloor, \ \ \forall S \subseteq P, 2 \le \chi \le |S|, \tag{4.52}$$

where S is a subset of pickup nodes. As in Jepsen *et al.* (2008) and Desaulniers *et al.* (2008), we focus on the cuts defined for subsets of three customers because these are easy to separate. These cuts can be rewritten as

$$\sum_{r\in\Omega_S} y_r \le 1, \ \ \forall S \subseteq P \text{ such that } |S| = 3, \tag{4.53}$$

where $\Omega_S \subseteq \Omega$ is the subset of paths completing at least two requests in S. These inequalities can again be separated by enumeration.

Note that handling the dual variables of the subset-row cuts (4.53) in the pricing problem can be highly time-consuming. Consequently, we limit their usage by generating them only in the first two levels of the branching tree and adding at most 50 cuts at once.

4.3.4 Branching

In a branch-price-and-cut algorithm, branching is used to obtain integer feasible solutions and must be compatible with the column generation process, especially with the algorithm used to solve the pricing problem. With the dominance criterion (4.37)–(4.41), the removal of arcs must preserve the triangle inequality (Ropke and Cordeau, 2008). Consequently, we propose to branch on the outflow of node subsets as for the VRP (Naddef and Rinaldi, 2002). This branching strategy adds constraints to the master problem, yielding additional dual variables incorporated in the objective function of the pricing problem (Desaulniers *et al.*, 2011). In this branching strategy, a subset of nodes S is selected such that $\sum_{r\in\Omega}\sum_{(i,j)\in A|i\in S,j\notin S} b^r_{ij}\tilde{y}_r$ is as far as possible from the nearest integer. Two branches are then created by adding the following constraints to the master problem associated with each branch :

$$\sum_{r\in\Omega}\sum_{(i,j)\in A|i\in S,j\notin S} b^r_{ij} y_r \le \left\lfloor \sum_{r\in\Omega}\sum_{(i,j)\in A|i\in S,j\notin S} b^r_{ij}\tilde{y}_r \right\rfloor$$

and

$$\sum_{r\in\Omega}\sum_{(i,j)\in A|i\in S,j\notin S} b_{ij}^r y_r \geq \left\lceil \sum_{r\in\Omega}\sum_{(i,j)\in A|i\in S,j\notin S} b_{ij}^r \tilde{y}_r \right\rceil .$$

The exploration of the enumeration tree is achieved through a best-first strategy.

4.4 Branch-Price-and-Cut Algorithm with LIFO Paths

This second branch-price-and-cut algorithm deals with the LIFO policy in the pricing problem. This suggests a stronger formulation yielding better lower bounds. We present a labeling algorithm for the elementary shortest path problem with pickup and delivery, time windows, capacity and LIFO constraints.

4.4.1 Set Partitioning Formulation

Let Ω^L denote the set of all feasible paths satisfying pickup and delivery, time window, capacity and LIFO constraints. Using the notation of Section 4.3, the PDPTWL is formulated as

$$\begin{aligned}
&\text{minimize} && \sum_{r\in\Omega^L} c_r y_r && && (4.54)\\
&\text{subject to} && \sum_{r\in\Omega^L} a_{ir} y_r = 1, && \forall i \in P, && (4.55)\\
& && y_r \in \{0,1\}, && \forall r \in \Omega^L. && (4.56)
\end{aligned}$$

Again, we resort to column generation to solve the linear relaxation of model (4.54)–(4.56). Because the pricing problem is solved with LIFO constraints, inequalities equivalent to (4.17), (4.23), and (4.24) are not used in this algorithm.

Constraints (4.55) are associated with dual variables $\alpha_i, i \in P$. The pricing problem is

$$\begin{aligned}
&\text{minimize} && \sum_{(i,j)\in A} \bar{c}_{ij} x_{ij} && (4.57)\\
&\text{subject to} && \text{constraints (4.3)-(4.13)}, && (4.58)
\end{aligned}$$

where index k is dropped again in (4.3)–(4.13). The reduced cost $\bar{c}_{ij}$ for each arc $(i,j) \in A$ can be defined as

$$\bar{c}_{ij} = \begin{cases} c_{ij} - \alpha_i, & \forall i \in P, \\ c_{ij}, & \forall i \in N\backslash P. \end{cases} \tag{4.59}$$

All the valid inequalities and the branching strategies presented in Sections 4.3.3 and 4.3.4 are used in the branch-price-and-cut algorithm with LIFO paths.

4.4.2 Labeling Algorithm

In this section, we adapt the three labeling algorithms presented in Section 4.3 to account for the LIFO policy. This means that the labels have to store information about the order in which past requests have been visited.

To solve the elementary version of the shortest path problem with pickup and delivery, time window, capacity and LIFO constraints, new components $\mathcal{H}_i,\ i \in P$, are introduced. For each onboard request $i \in P$, $\mathcal{H}_i$ indicates its position in the stack : $\mathcal{H}_i > 0$ if request i is onboard, and $\mathcal{H}_i = 0$ otherwise. For each $i, j \in P$, if the position of the onboard request j is larger than the position of the onboard request i, then the pickup node associated with request i was visited before the pickup node associated with request j, i.e. if $0 < \mathcal{H}_i < \mathcal{H}_j$, then $0 \prec i \prec j$. For each label, the components η, t, ι, c, U and $\mathcal{H}_i, \forall i \in P$ are stored.

Because the LIFO policy is imposed, the extension of a label E along an arc $(\eta(E), j) \in A$ must satisfy one of the following four conditions :

$$0 < j \leq n \quad \text{and} \quad j \notin U(E), \tag{4.60}$$

$$0 < \eta(E) \leq n \quad \text{and} \quad n < j \leq 2n \quad \text{and} \quad j = \eta(E) + n, \tag{4.61}$$

$$n < \eta(E) \leq 2n \quad \text{and} \quad n < j \leq 2n \quad \text{and} \quad \mathcal{H}_{j-n}(E) > \mathcal{H}_i(E),\ \forall i \in P\backslash\{j-n\}, \tag{4.62}$$

$$j = 2n+1 \quad \text{and} \quad \mathcal{H}_i(E) = 0,\ \forall i \in P. \tag{4.63}$$

Condition (4.60) ensures that if j is a pickup node, then it must not have been previously visited and it must be reachable with respect to the time windows. Condition (4.61) states that if j is a delivery node and $\eta(E)$ is a pickup node, then j must be the delivery node associated with request $\eta(E)$. Note that this condition is always respected in graph G because the arcs (i, j) going from pickup nodes to delivery nodes are only created if $i \in P$ and $j = n + i$. Condition (4.62) stipulates that if j and $\eta(E)$ are delivery nodes, then j must be the delivery node associated with the last visited onboard request. Condition (4.63) ensures that all requests serviced along the path are completed when reaching the destination node. Together these conditions enforce the pairing constraints for the pickup and delivery nodes of each request. They also ensure that all paths are elementary, and that the LIFO policy is respected.

The new label E' corresponding to the extension of a label E is set through equations

(4.29)–(4.32), (4.34), and

$$\mathcal{H}_l(E') = \begin{cases} 1+\max_i\{\mathcal{H}_i(E)\} & \text{if } j = l, \\ 0 & \text{if } j \in D, j-n = l, \\ \mathcal{H}_l(E) & \text{otherwise,} \end{cases} \qquad \forall l \in P. \tag{4.64}$$

Equations (4.64) update the positions of the requests. If j is a pickup node, then the highest position is given to node j; otherwise its request position is set to 0. In both cases, the positions of all the other requests $i \in P, i \neq j$ remain the same. If conditions (4.35) and (4.36) are respected, then E' is kept unchanged.

If the delivery triangle inequality holds, we apply the following dominance condition : a label E_1 dominates a label E_2 if the conditions (4.37)–(4.39), (4.41), and

$$\forall i, j \text{ such that } 0 < \mathcal{H}_i(E_1) \leq \mathcal{H}_j(E_1) \text{ then } 0 < \mathcal{H}_i(E_2) \leq \mathcal{H}_j(E_2) \tag{4.65}$$

hold. Note that this relaxation can be satisfied even if $\mathcal{O}(E_1) \subset \mathcal{O}(E_2)$.

Let $R(E)$ represent the path corresponding to label E and (r_1, r_2) the path obtained by concatenating paths r_1 and r_2.

Proposition 4.4.1. *Conditions* (4.37)–(4.39)*,* (4.41)*, and* (4.65) *constitute a valid dominance criterion whenever $\bar{c}_{ij}$ satisfies the delivery triangle inequality.*

Proof. The proof follows from those of Propositions 1 and 3 of Ropke and Cordeau (2009). Let r be a LIFO-feasible path extending $R(E_2)$ to node $2n + 1$. If no such path exists, then clearly one can remove label E_2. Let r' be the path obtained from r by removing the deliveries corresponding to each request $i \in P$ such that $\mathcal{H}_i(E_1) = 0$ and $\mathcal{H}_i(E_2) > 0$. Because $(R(E_2), r)$ is feasible with respect to time windows, capacity constraints, elementarity constraints, and pickup and delivery constraints, then so is $(R(E_1), r')$. The LIFO constraints are not violated because the order in which the deliveries are performed on $(R(E_1), r')$ is the same as the order on $(R(E_2), r)$. Because $(R(E_2), r)$ is feasible, then so is $(R(E_1), r')$. Because the extension function (4.32) is non-decreasing for the reduced cost component c, and the delivery triangle inequality is assumed, the cost of r' does not exceed that of r. Because $c(E_1) \leq c(E_2)$, the cost of $(R(E_1), r')$ is at most equal to that of $(R(E_2), r)$. As a result, the best extension of label E_1 to $2n+1$ cannot be worse than the best extension of E_2 to $2n+1$. Hence, label E_1 dominates label E_2. $\square$

At this point, it should be noted that the arguments of this proof cannot be adapted to the ng-path relaxation because deleting a delivery node from r can yield a cycle that is not allowed in $(R(E_1), r')$, implying that labels could be incorrectly dominated. However, this case only arises in non-elementary paths, which means that no integer solution can be discarded. Thus, the lower bound obtained at a branching node and the proposed algorithms remain valid.

4.5 A Hybrid Branch-Price-and-Cut Algorithm

The two algorithms just described have advantages and drawbacks. The first one solves shortest paths problems without the LIFO constraints, leading to a less restrictive dominance criterion than for the second algorithm (i.e. more labels can be dominated). The pricing problem is then easier and faster to solve. On the other hand, the second algorithm solves shortest path problems under the LIFO policy, yielding a stronger relaxation than for the first algorithm. In fact, the potential number of variables that can be generated is never larger ($\Omega^L \subseteq \Omega$) and all these variables respect the LIFO policy.

In the third algorithm that we will now present, the idea is to combine the respective advantages of the first two algorithms. This will be achieved by solving shortest path problems under hybrid-LIFO constraints, i.e., they must respect the LIFO policy for at most κ requests at the same time. An ejection stack process will therefore be needed : when a pickup node is visited, it is put on top of the stack ; if the height of the stack exceeds κ, the request at the bottom of the stack is then ejected. The extension of the partial path then needs to respect the LIFO policy for the onboard requests in the stack, but all remaining requests can be visited without respecting the LIFO policy. Figure 4.3 depicts the cases that may arise when $\kappa = 2$. The shaded nodes represent the requests that are in the ejection stack and for which the extension of the path must respect the LIFO policy. Figure 4.3a depicts an initial path containing two nodes in the stack. Figure 4.3b shows what happens when the stack reaches its maximal size. Figures 4.3c, 4.3d and 4.3e depict the possible extensions of the LIFO-feasible path for each node in the stack. As shown in Figure 4.3e, the solution of the linear relaxation can contain LIFO-infeasible paths. If this is the case, incompatible predecessor and successor inequalities, and inequalities (4.18) are added.

4.5.1 Labeling Algorithm

We now describe the labeling algorithm we have implemented to solve the shortest paths with hybrid-LIFO constraints. The valid inequalities and the branching decisions used are those of Sections 4.3.3 and 4.3.4. We have adapted the elementary version of the labeling

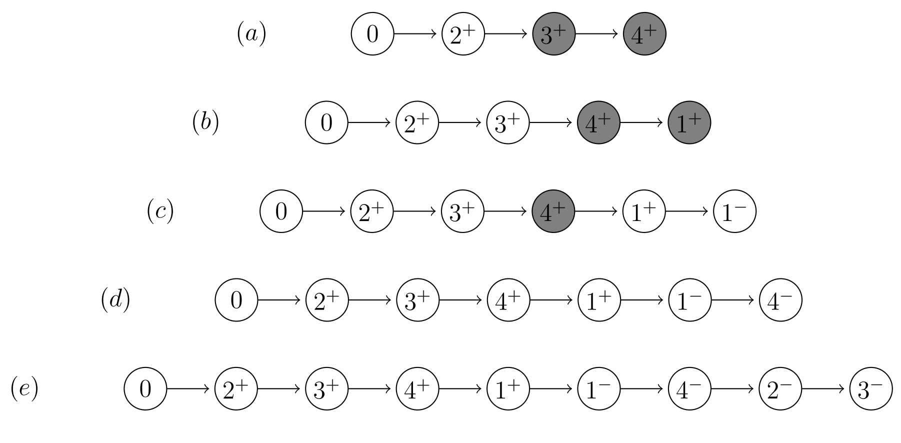


Figure 4.3 – (a) The LIFO stack with $\kappa = 2$ contains nodes 3^+ and 4^+. (b) The LIFO stack contains nodes 4^+ and 1^+ : node 3^+ has been ejected. (c) The LIFO stack contains node 4^+ : request 1 is not onboard. (d) The LIFO stack is empty once requests 1 and 4 have been completed. (e) Requests 2 and 3 can then be completed without respecting the LIFO policy.

algorithm described in Section 4.3 to shortest path problems with pickup and delivery, time window, capacity and hybrid-LIFO constraints. It will be necessary to consider the order in which at most κ requests have already been visited on the partial path.

In the elementary version of the problem, every label stores the following components : η, t, c, $\mathcal{O}$, U, and $\mathcal{H}_i^{ES}$, $\forall i \in P$. $\mathcal{H}_i^{ES}$ represents the position of a request in the ejection stack : for each request $i \in P$ that is not onboard, $\mathcal{H}_i^{ES} = 0$. Otherwise, if request i is onboard, $\mathcal{H}_i^{ES}$ can take multiple values. If $\mathcal{H}_i^{ES} = 0$, request i is not in the ejection stack i.e. the extension of the label may not respect the LIFO policy for this request, and if $\mathcal{H}_i^{ES} > 0$, the extension of the label must respect the LIFO policy for request i as long as i stays in the stack. Consequently, if $0 < \mathcal{H}_i^{ES} < \mathcal{H}_j^{ES}$ for $i, j \in P$, then the extension of the label must complete request j before request i, unless i is ejected from the stack before performing the delivery at j.

The extension of a label E along an arc $(\eta(E), j) \in A$ is feasible if one of the four following conditions is respected :

$$0 < j \leq n \quad \text{and} \quad j \notin U(E), \tag{4.66}$$

$$0 < \eta(E) \leq n \quad \text{and} \quad n < j \leq 2n \quad \text{and} \quad j = \eta(E) + n, \tag{4.67}$$

$$n < \eta(E) \leq 2n \quad \text{and} \quad n < j \leq 2n \quad \text{and} \quad \mathcal{H}_{j-n}^{ES}(E) \geq \mathcal{H}_i^{ES}(E), \quad \forall i \in P,\ j-n \in \mathcal{O}(E), \tag{4.68}$$

$$j = 2n+1 \quad \text{and} \quad \mathcal{O}(E) = \emptyset. \tag{4.69}$$

Condition (4.68) specifies that if j and $\eta(E)$ are delivery nodes, and there exists a request in the ejection stack, then j is the delivery node associated with the request on top of the stack. Otherwise, j is the delivery node of any onboard request.

For a label E, its extension along arc $(\eta(E), j) \in A$ generates a new label E' where the information is set with equations (4.29)–(4.34) and

$$\mathcal{H}_l^{ES}(E') = \begin{cases} \min\{\kappa, 1 + \max_i\{\mathcal{H}_i^{ES}(E)\}\} & \text{if } j = l, \\ 0 & \text{if } j \in D, j - n = l, \\ \mathcal{H}_l^{ES}(E) - 1 & \text{if } j \in P, j \neq l, \mathcal{H}_l^{ES} > 0, \\ & \quad \max_i\{\mathcal{H}_i^{ES}(E)\} = \kappa, \\ \mathcal{H}_l^{ES}(E) & \text{otherwise,} \end{cases} \qquad \forall l \in P. \quad (4.70)$$

Equation (4.70) updates the position of the requests in the ejection stack. If j is a pickup node and the size of the ejection stack is less than κ, then all positions remain the same. Otherwise, if j is a pickup node but the size of the LIFO stack is equal to κ, then the request i such that $\mathcal{H}_i^{ES}(E) = 1$ is removed from the ejection stack and the positions of the other requests are changed accordingly. In both cases, the highest position is given to node j. If j is a delivery node, then its request position becomes 0 and the other positions are unchanged. If the new label created E' satisfies the time windows and capacity constraints (4.35) and (4.36), then it remains unchanged.

If the delivery triangle inequality holds, then the dominance criterion is the following : a label E_1 dominates a label E_2 if (4.37)–(4.41) are satisfied and

$$\forall i, j \text{ such that } 0 < \mathcal{H}_i^{ES}(E_1) \leq \mathcal{H}_j^{ES}(E_1) \text{ then } 0 < \mathcal{H}_i^{ES}(E_2) \leq \mathcal{H}_j^{ES}(E_2). \qquad (4.71)$$

For each request $i \in P$ such that $\mathcal{H}_i^{ES}(E) > 0$, condition (4.40) is implied by condition (4.71), meaning that it can be substituted by

$$\mathcal{O}(E_1)\backslash\{i \in P | \mathcal{H}_i^{ES}(E_1) > 0\} \subseteq \mathcal{O}(E_2)\backslash\{i \in P | \mathcal{H}_i^{ES}(E_1) > 0\}. \qquad (4.72)$$

The reader can easily adapt this procedure to the cyclic and the ng-path relaxations.

4.6 Computational Results

The three branch-price-and-cut algorithms just described were tested on a set of PDPTWL instances derived from the instances proposed by Ropke and Cordeau (2009) for the PDPTW.

In this section, we report the computational results obtained for these PDPTWL instances. We also compare the results obtained for each instance of the PDPTW and of the PDPTWL. All tests were performed on a Linux computer equipped with an Intel(R) Core(TM) i7-3770 processor (3.4 GHz). The algorithms were implemented using the GENCOL library using CPLEX 12.4.0.0 to solve all restricted master problems.

4.6.1 Instances

To test our algorithms, we have used a modified version of the instances proposed by Ropke and Cordeau (2009) for the PDPTW. The coordinates of the depot, and of the pickup and delivery nodes are the same as in their instances. For each request $i \in P$, we have used the original time windows for the pickup nodes, but have delayed the time windows for the delivery nodes as follows $\underline{w}_{n+i} = \underline{w}_{n+i} + \Delta$ and $\bar{w}_{n+i} = \bar{w}_{n+i} + \Delta$, where Δ is a user-defined parameter called the delay. The load q_i of request $i \in P$ was not modified, but the vehicle capacity Q was increased by a factor of 1.5 for the AA and BB groups, and by a factor of 1.25 for the CC and DD groups. We have made these modifications in order to increase the number of requests that may be handled simultaneously by a vehicle. Without such modifications, removing only the arcs (i, j) such that $i \in P$, $j \in D$ and $j \neq n + i$ is often sufficient to find LIFO-feasible solutions.

Table 4.1 summarizes the characteristics of the test instances. For each group, we report the vehicle capacity Q, the width of the time windows W, and the delay Δ applied to the time windows of the delivery nodes. For each group, we have tested 10 instances in which the number of requests ranges from 30 to 75. In all instances, the primary objective is the minimization of the number of vehicles. To this end, we have imposed a fixed cost of 10,000 on each arc $(0, j) \in A$ such that $j \in P$.

Table 4.1 – Characteristics of the PDPTWL instances

Group	Q	W	Δ
AA	22	60	45
BB	30	60	45
CC	18	120	15
DD	25	120	15

4.6.2 Impact of Algorithm on Results

Table 4.2 presents the number of instances solved for each algorithm and each relaxation of the pricing problem. A time limit of 3600 seconds was imposed for the solution of each instance. The algorithms are named as follows : *BPC Non-LIFO* (branch-price-and-cut with LIFO-infeasible paths), *BPC LIFO* (branch-price-and-cut with LIFO paths), and *BPC Hybrid* (hybrid branch-price-and-cut with $\kappa = 2$). The relaxation of the shortest path problems are named as follows : *ESPP* (elementary shortest path problem), *SPP* (shortest path problem without elementarity constraints), and ng_λ (ng-route relaxation, where λ represents the maximal number of neighbors for each request). We can see that, for all three algorithms, the ng-route relaxation with $\lambda = 10$ solves the most instances. In our experiments, we have observed that all instances solved with the other relaxations are also solved with ng_{10}. Therefore, we will henceforth solve all shortest path problems with ng_{10}. Note that all our conclusions should be similar for each relaxation of the shortest path problem because there does not seem to exist any correlation between the performance of the algorithm and the relaxation of the pricing problem.

Table 4.3 presents the results for all three algorithms. The first column indicates the name of the instances corresponding to its group and to its number of requests. In the last column, z^* is the optimal solution value obtained for each instance. For each algorithm, we present the following information : *Sec.*, the CPU time in seconds ; $\underline{z}$, the lower bound at the root node (before adding any cuts) ; *LC*, the number of constraints (4.18), (4.23), and (4.24) added to the master problem to obtain a LIFO-feasible solution ; *OC*, the number of other constraints added to the master problem, i.e. constraints (4.50), (4.51), (4.53) ; and *B*, the number of nodes in the search tree including the root node. Whenever an instance is not solved within the prescribed time limit, but a lower bound has been identified, then the lower bound value is reported. Note that for instances BB55, BB70, BB75, CC45, CC60 to CC75, and DD45 to DD75 no feasible solutions were found within the time limit, and no non-trivial lower bounds could be identified for DD55, DD60, and DD75.

Table 4.2 – Number of instances solved

	BPC Non-LIFO	BPC LIFO	BPC Hybrid
ESPP	15	20	20
SPP	19	24	24
ng_5	18	23	23
ng_{10}	19	25	25
ng_{15}	18	22	22

Table 4.3 indicates that the branch-price-and-cut algorithm with LIFO-infeasible paths does not solve as many instances as the other two algorithms and is generally slower. Because this algorithm needs more branching nodes, the CPU time can be larger. In fact, for the AA group, this time is relatively large and the algorithm cannot solve instances with more than 55 requests. For the other groups, the number of branching nodes is not as large, thus allowing the algorithm to perform well.

For these instances, no LIFO cuts are added in the BPC Hybrid. In fact, these instances are constructed in such a way that a vehicle can contain items corresponding to relatively few requests simultaneously. In the solution of a typical instance, the maximum number of onboard requests at any given time ranges between 2 and 3. This maximum is rarely reached leading to 1.3 onboard requests on average when performing a delivery. Setting $\kappa \geq 3$ yields the same results as $\kappa = 2$. There does not seem to be any significant difference between the performance of BPC Hybrid and that of BPC LIFO in terms of lower bound value (it differs only for instance BB40) and of CPU time. Additional results will be presented in Section 4.6.4 to determine whether the two algorithms perform differently on harder instances. Note that, for the BPC non-LIFO, most LC cuts added in the master problem are incompatible predecessor and successor constraints. Indeed, these inequalities are often sufficient to enforce the LIFO policy for the tested instances. Most of the other cuts (OC) are 2-path cuts and subset-row inequalities.

4.6.3 Impact of LIFO Policy on Results

In Table 4.4 we examine the impact of imposing the LIFO policy by comparing the optimal solution costs and computational times for the PDPTW and the PDPTWL. For each problem, we provide : *Sec.*, the CPU time in seconds required to obtain an optimal solution ; B, the number of nodes in the search tree ; *It.*, the overall number of column generation iterations ; *Col.*, the number of columns generated ; and z^*, the optimal solution cost. We also report the increase in the number of vehicles used (*Veh.*) and in the travel costs (TC) induced by the LIFO policy. For the PDPTWL, we report the results obtained with the branch-price-and-cut with LIFO paths. For the PDPTW, we report the results obtained with a branch-price-and-cut similar to that proposed by Ropke and Cordeau (2009). We do not report any result for the CC and DD groups because no instance except CC30 was solved for the PDPTW. Note that instances AA75, and BB55 to BB75 could not be solved for the PDPTW.

Table 4.3 – Comparative computational results for the three algorithms with ng_{10}

	BPC Non-LIFO					BPC LIFO				BPC Hybrid					
Inst.	Sec.	$\underline{z}$	LC	OC	B	Sec.	$\underline{z}$	OC	B	Sec.	$\underline{z}$	LC	OC	B	z^*
AA30	1.6	31,129.4	0	0	1	2.5	31,129.4	0	1	2.5	31,129.4	0	0	1	31,129.4
AA35	109.5	31,268.3	4	53	17	16.7	31,285.2	16	1	16.7	31,285.2	0	16	1	31,294.1
AA40	139.0	41,338.2	15	20	24	7.1	41,349.2	0	1	7.1	41,349.2	0	0	1	41,349.2
AA45	207.0	41,504.1	4	48	13	21.6	41,521.4	0	1	21.6	41,521.4	0	0	1	41,521.4
AA50	305.6	41,619.6	6	46	11	42.3	41,643.6	0	1	41.0	41,643.6	0	0	1	41,643.6
AA55		46,778.6				60.6	46,803.8	0	2	59.0	46,803.8	0	0	2	51,743.2
AA60		46,972.0				800.4	46,999.2	69	4	789.2	46,999.2	0	69	4	51,949.7
AA65		47,147.4				665.8	47,172.0	60	2	657.5	47,172.0	0	60	2	52,077.4
AA70		47,877.3				1,149.4	47,896.1	27	2	1,094.5	47,896.1	0	27	2	52,219.2
AA75		51,587.3				1,994.2	51,607.2	29	2	1,893.4	51,607.2	0	29	2	52,330.1
BB30	5.0	31,074.5	1	2	1	5.6	31,076.3	2	1	5.4	31,076.3	0	2	1	31,077.5
BB35	13.7	31,311.0	2	0	1	13.9	31,312.4	0	1	13.2	31,312.4	0	0	1	31,312.4
BB40	122.4	35,143.6	3	48	2	142.5	35,695.5	30	2	130.0	35,559.9	0	50	2	41,404.0
BB45	1,627.8	37,516.7	9	65	8	838.8	37,645.1	56	4	667.7	37,645.1	0	65	4	41,537.5
BB50	565.6	41,791.1	0	0	1	720.8	41,791.1	0	1	687.3	41,791.1	0	0	1	41,791.1
BB55		45,637.2					46,391.4				46,391.4				
BB60	887.8	62,296.8	10	44	3	766.8	62,305.5	0	1	468.4	62,305.5	0	0	1	62,305.5
BB65	576.2	62,564.6	0	0	1	1,545.9	62,564.6	0	1	1,015.3	62,564.6	0	0	1	62,564.6
CC30	18.4	23,318.9	0	34	2	17.6	23,318.9	40	2	17.6	23,318.9	0	40	2	31,088.6
CC35	37.4	24,777.2	0	24	2	39.6	24,777.2	24	2	39.4	24,777.2	0	24	2	31,237.4
CC40	45.5	26,024.6	0	0	2	62.5	26,024.6	0	2	62.2	26,024.6	0	0	2	31,340.2
CC45		29,562.7					29,562.7				29,562.7				
CC50	889.4	35,156.6	0	52	12	1,481.0	35,156.6	52	8	1,475.2	35,156.6	0	52	8	41,673.6
CC55	2,817.7	36,778.1	2	54	34	2,733.3	36,779.4	52	32	2,741.1	36,779.4	0	52	32	41,793.5
DD30		19,051.5				3,351.8	19,153.3	58	2	3,307.2	19,153.3	0	58	2	21,103.2
DD35	695.1	21,774.6	7	52	6	794.4	21,854.0	52	2	767.8	21,854.0	0	52	2	31,127.8
DD40	1,820.3	22,932.5	8	41	2	3,468.6	23,024.6	20	2	3,092.7	23,024.6	0	20	2	31,245.3

Table 4.4 – Comparative computational results for the PDPTW and the PDPTWL

	PDPTWL					PDPTW					Increase	
Instance[1]	Sec.	B	It.	Col.	z^*	Sec.	B	It.	Col.	z^*	Veh.	TC (%)
AA30	2.5	1	108	3,100	31,129.4	2.5	1	79	4,663	30,969.4	0	16.5
AA35	16.7	1	180	8,594	31,294.1	20.5	1	159	7,507	31,089.0	0	18.8
AA40	7.1	1	95	4,850	41,349.2	53.5	3	192	7,494	41,241.7	0	8.7
AA45	21.6	1	124	8,143	41,521.4	167.0	3	198	9,460	41,412.2	0	7.7
AA50	42.3	1	130	11,460	41,643.6	1,968.0	13	368	30,221	41,531.6	0	7.3
AA55	60.6	2	172	12,608	51,743.2	775.4	11	364	18,744	41,667.1	1	4.6
AA60	800.4	4	349	25,324	51,949.7	1,682.9	7	800	96,591	41,822.7	1	7.0
AA65	665.8	2	424	30,487	52,077.4	1,029.2	3	540	80,053	42,011.6	1	3.3
AA70	1,149.4	2	400	36,740	52,219.2	2,434.2	2	447	88,691	51,992.8	0	11.4
BB30	5.6	1	78	2,808	31,077.5	5.0	1	67	3,115	31,017.5	0	5.9
BB35	13.9	1	99	4,924	31,312.4	28.9	1	136	5,629	31,211.3	0	8.3
BB40	142.5	2	193	8,358	41,404.0	131.7	1	112	9,796	31,503.2	1	-6.6
BB45	838.8	4	277	12,760	41,537.5	539.5	2	198	11,235	41,386.4	0	10.9
BB50	720.8	1	110	8,428	41,791.1	1,908.4	1	154	11,480	41,564.9	0	14.5

Table 4.5 – Characteristics of the additional PDPTWL instances

Group	Q	W	Δ
AA*	26	60	60
BB*	35	60	60

We first observe that imposing the LIFO policy either requires an additional vehicle or increases the total travel cost by at least 5.9%. In fact, the travel cost increase can reach almost 20%, which is substantial. Note that even when an extra vehicle is used, the travel cost increases, except for instance BB40.

Second, we observe that, beside the fact that some CC and DD instances were solved for the PDPTWL but not for the PDPTW, it takes longer to solve the PDPTW for the AA instances. This can be explained by the larger number of nodes generated in the search tree, yielding a larger number of column generation iterations. The computational times are, however, relatively similar for the BB instances, except for the instance BB50 where the PDPTW problem requires much more time than the PDPTWL. From these results, we conclude that the pricing problems for the PDPTW and the PDPTWL are equivalently difficult to solve. Indeed, even if the dominance rule in the labeling algorithm is more restrictive for the PDPTWL, fewer paths are feasible, reducing the overall number of labels generated. Moreover, the input graph for the PDPTWL contains fewer arcs than that for the PDPTW (it does not contain the arcs (i, j) such that $i \in P$, $j \in D$ and $j \neq n + i$).

4.6.4 Results on Additional Instances

Because the results presented in Table 4.3 do not exhibit significant differences between the two best proposed algorithms, we have decided to run additional experiments for groups AA and BB. To this end, we have created the harder-to-solve instances described in Table 4.5. We present in Table 4.6 the results obtained on these new instances with a time limit of 7200 seconds. These instances allow more requests to be simultaneously present in a vehicle. In the solution of a typical instance, the maximum number of onboard requests at any given time is 3, and the average number of onboard requests when performing a delivery is 1.6 leading to an increase of approximately 20% over the previous average. We observe that, for both BPC Non-LIFO and BPC Hybrid, more LIFO cuts are generated in the solution process. We can see that the BPC Hybrid performs slightly better than the other two algorithms. This algorithm can solve instance BB*60 while the two others could not. For most instances, except AA*35, the BPC Hybrid is the fastest and produces good quality lower bounds. Note

Table 4.6 – Additional comparative computational results for the three algorithms with ng_{10}

	BPC Non-LIFO					BPC LIFO				BPC Hybrid					
Inst.	Sec.	$\underline{z}$	LC	OC	B	Sec.	$\underline{z}$	OC	B	Sec.	$\underline{z}$	LC	OC	B	z^*
AA*30	8.8	31,048.8	1	20	3	6.7	31,051.4	20	1	7.3	31,051.4	0	20	1	31,051.6
AA*35	3,891.9	31,210.3	414	50	12	1,145.7	31,231.7	69	55	2,497.2	31,231.3	4	70	68	31,244.8
AA*40	615.9	36,354.5	66	50	18	61.4	36,364.4	25	2	60.5	36,364.4	0	25	2	41,331.4
AA*45		36,558.4				2,666.4	36,590.3	85	26	1,092.5	36,589.2	9	85	20	41,515.4
AA*50		38,339.6					38,376.4				38,375.0				
AA*55		41,793.8					41,843.5				41,843.4				
AA*60	5,923.8	45,237.3	21	60	36	1,779.0	45,280.4	0	2	1,512.6	45,279.3	0	0	2	51,808.2
BB*30	16.1	32,843.8	9	4	2	18.1	36,144.2	11	2	15.7	36,144.2	0	11	2	41,111.0
BB*35	170.1	36,377.2	24	56	6	109.8	37,122.8	49	2	89.4	37,122.8	0	42	2	41,332.9
BB*40	6,036.6	37,788.0	49	56	24	2,597.0	39,337.7	54	14	1,844.4	39,337.7	0	52	12	41,477.1
BB*45		39,527.1				3,455.7	41,646.1	23	1	2,468.1	41,646.1	0	23	1	41,699.5
BB*50		45,011.2					46,505.4				46,505.4				
BB*55		48,267.2									49,891.9				
BB*60		63,148.6								6,217.0	65,172.6	3	51	2	72,184.3

that for five instances, the lower bound obtained with BPC Hybrid is slightly worse than that obtained with BPC LIFO. These results indicate that the BPC Hybrid should perform better on harder instances.

4.7 Conclusions

We have introduced the PDPTWL and described three column generation algorithms for its solution. We have implemented an adaptation of the labeling algorithm for shortest path problems with LIFO constraints. Instances involving up to 75 requests were solved to optimality. We have shown that on harder instances, the BPC Hybrid outperforms the other two algorithms because it is faster and produces better lower bounds. Our results should serve as benchmarks for future research.

4.8 Nouveaux résultats

Suite à la publication de ce chapitre dans *Transportation Science*, nous avons améliorer l'algorithme d'étiquetage afin d'accélérer le temps de résolution. Pour ce faire, à chaque extension d'une étiquette, on vérifie qu'il est possible d'aller visiter chaque noeud de livraison associé à chaque colis à bord. Si ce n'est pas possible, l'étiquette est éliminée. Cela permet de diminuer les temps de calcul.

Les tableaux 4.7 et 4.8 présentent les nouveaux résultats et devraient, respectivement, remplacer les tableaux 4.3 et 4.6. Dans les tableaux 4.7–4.8, la première colonne indique le nom de l'instance correspondant à son groupe et au nombre de requêtes. La dernière colonne, z^*, présente la valeur optimale. Pour chaque algorithme, on présente l'information suivante : *Sec.*, le temps de résolution en secondes ; $\underline{z}$, la borne inférieure au noeud racine (avant l'ajout de coupes) ; LC, le nombre de contraintes (4.18), (4.23) et (4.24) ajoutées au problème-maître pour obtenir une solution qui respecte la politique LIFO ; OC, le nombre de contraintes (4.50), (4.51) et (4.53) ajoutées au problème-maître ; et B, le nombre de noeuds dans l'arbre de recherche incluant le noeud racine. Pour les instances des groupes AA à DD, le temps limite de calcul est de une heure, alors que, pour les instances additionnelles des groupes AA* et BB*, le temps limite de calcul est de deux heures. Lorsqu'une instance n'est pas résolue dans le temps limite de calcul, aucune borne inférieure n'est rapportée. De plus, aucune solution réalisable n'a été trouvée pour toutes les instances non résolues dans le temps limite.

Nous pouvons maintenant constater que toutes les instances des groupes AA et BB sont résolues à optimalité avec les algorithmes *BPC LIFO* et *BPC Hybrid*. De plus, l'algorithme *BPC Non-LIFO* semble être le plus performant pour les instances du groupe CC. Pour les instances plus difficiles, i.e., les instances des groupes AA* et BB*, elles sont maintenant presque toutes résolues lorsqu'il y a 65 requêtes et moins, sauf pour l'instance BB*55. Pour ces instances, il ne semble pas y avoir de différence significative entre les algorithmes *BPC LIFO* et *BPC Hybrid*, mais l'algorithme *BPC Non-LIFO* demeure le moins performant.

Pour l'algorithme *BPC Non-LIFO*, sept instances supplémentaires sont résolues (AA65, BB70, CC60, DD45, DD50, AA*65 et BB*60). De plus, pour toutes les instances qui étaient résolues auparavant, les temps sont diminués avec cet algorithme. Le temps diminue en moyenne de 839,8 secondes et l'impact le plus important est pour l'instance BB*40 avec une diminution de 5795,0 secondes. Pour l'algorithme *BPC LIFO*, onze instances supplémentaires sont résolues (BB55, BB70, BB75, DD45, DD50, AA*50, AA*55, AA*65, BB*50, BB*60 et BB*65) et une instance n'est plus résolue dans le temps limite (CC55). De plus, pour toutes les instances sauf AA60 et CC30, le temps diminue en moyenne de 866,0 secondes et l'impact le plus important est pour l'instance BB*45, avec une diminution de 3389,0 secondes. Toutefois, pour les instances AA60 et CC30, les temps de résolution augmentent de 99,1 et de 9,9 secondes. Pour l'algorithme *BPC Hybrid*, neuf instances supplémentaires sont résolues (BB55, BB70, BB75, DD45, DD50, AA*50, AA*55, AA*65 et BB*50) et une instance n'est plus résolue dans le temps limite (CC55). De plus, pour toutes les instances sauf AA60 et CC30, le temps diminue en moyenne de 904,1 secondes et l'impact le plus important est pour l'instance BB*60 avec une diminution de 6061,4 secondes. Toutefois, pour les instances AA60 et CC30, les temps de résolution augmentent respectivement de 69,0 et 10,1.

Au total, neuf instances supplémentaires sont résolues (BB50, CC60, DD45, DD50, AA*50, AA*55, AA*65m BB*50 et BB*65) et, en moyenne, les temps sont diminués.

Tableau 4.7 – Nouveaux résultats pour les trois algorithmes avec ng_{10}

	BPC Non-LIFO					BPC LIFO				BPC Hybrid					
Inst.	Sec.	$\underline{z}$	LC	OC	B	Sec.	$\underline{z}$	OC	B	Sec.	$\underline{z}$	LC	OC	B	z^*
AA30	1,1	31129,4	0	0	1	1	31129,4	0	1,0	1,1	31129,4	0	0	1	31 129,4
AA35	68,2	31268,3	2	4	21	8,2	31285,2	4	3	8,6	31285,2	0	4	3	31 294,1
AA40	92,6	41338,2	15	0	33	2,3	41349,2	0	1	2,3	41349,2	0	0	1	41 349,2
AA45	135,8	41504,1	4	5	21	4,2	41521,4	0	1	4,5	41521,4	0	0	1	41 521,4
AA50	157,9	41619,6	6	4	17	10,9	41643,6	0	1	8,3	41643,6	0	0	1	41 643,6
AA55	2952,8	46778,6	190	33	218	13,6	46803,8	0	2	16,4	46803,8	0	0	2	51 743,2
AA60						899,5	46999,2	36	41	858,2	46999,2	0	38	39	51 949,7
AA65	1699,8	47147,4	7	34	63	169,3	47172,0	16	5	238,9	47172,0	0	23	7	52 077,4
AA70						151,9	47896,1	7	4	153,4	47896,1	0	7	4	52 219,2
AA75						240,2	51607,2	9	6	229,4	51607,2	0	9	6	52 330,1
BB30	3,6	31074,5	1	2	3	3,0	31076,3	2	2	2,9	31076,3	0	2	2	31 077,5
BB35	4,5	31311,0	2	0	2	3,1	31312,4	0	1	3,0	31312,4	0	0	1	31 312,4
BB40	37,8	35143,6	3	8	6	37,1	35695,5	10	5	37,3	35559,9	0	10	5	41 404,0
BB45	202,1	37516,7	9	5	16	123,0	37645,1	7	9	143,9	37645,1	0	15	14	41 537,5
BB50	18,3	41791,1	0	0	1	20,5	41791,1	0	1	21,9	41791,1	0	0	1	41 791,1
BB55						548,5	46391,4	5	18	518,2	46391,4	0	10	17	51 911,7
BB60	107,6	62296,8	10	4	9	15,2	62305,5	0	1	15,0	62305,5	0	0	1	62 305,5
BB65	22,4	62564,6	0	0	1	24,0	62564,6	0	1	22,5	62564,6	0	0	1	62 564,6
BB70	584,2	65978,7	14	6	24	429,3	66002,4	3	12	461,8	66002,4	0	7	13	72 535,2
BB75						1660,3	68197,0	9	47	806,7	68197,0	0	13	25	72 656,7
CC30	27,9	23318,9	0	0	7	27,5	23318,9	0	7	27,7	23318,9	0	0	7	31 088,6
CC35	33,1	24777,2	0	0	4	32,7	24777,2	0	4	32,8	24777,2	0	0	4	31 237,4
CC40	30,4	26024,6	0	0	2	32,5	26024,6	0	2	32,3	26024,6	0	0	2	31 340,2
CC45															
CC50	345,6	35156,6	0	2	12	1013,5	35156,6	2	26	1026,1	35156,6	0	2	26	41 673,6
CC55	1416,1	36778,1	2	4	40										41 793,5
CC60	3500,1	38262,6	8	1	52										41 947,3
DD30						790,2	19153,3	13	9	825,3	19153,3	0	13	9	21 103,2
DD35	243,2	21774,6	8	2	13	154,4	21854,0	5	6	160,2	21854,0	0	5	6	31 127,8
DD40	233,9	22932,5	6	1	7	176,1	23024,6	1	4	189,5	23024,6	0	1	4	31 245,3
DD45	562,6	24537,8	6	2	10	482,3	24560,7	2	9	434,2	24560,7	0	1	5	31 350,4
DD50	903,0	25512,7	16	1	17	827,3	25547,8	1	10	914,1	25547,8	0	1	10	31 450,2

Tableau 4.8 – Nouveaux résultats des instances additionnelles pour les trois algorithmes avec ng_{10}

	BPC Non-LIFO					BPC LIFO				BPC Hybrid					
Inst.	Sec.	$\underline{z}$	LC	OC	B	Sec.	$\underline{z}$	OC	B	Sec.	$\underline{z}$	LC	OC	B	z^*
AA*30	4,2	31048,8	3	0	3	3,4	31051,4	0	2	3,5	31051,4	0	0	2	31 051,6
AA*35	1334,2	31210,4	275	20	113	391,7	31231,7	20	62	242,8	31231,3	0	17	37	31 244,8
AA*40	360,4	36354,5	25	43	54	16,5	36364,4	25	3	18,7	36364,4	0	25	3	41 331,4
AA*45						355,9	36590,3	29	23	373,9	36589,2	2	20	24	41 515,4
AA*50						2148,6	38376,4	47	84	1284,0	38375,0	3	22	41	41 637,5
AA*55						356,5	41843,5	28	11	2366,5	41843,4	5	41	70	41 880,2
AA*60	526,4	45237,3	31	25	36	52,4	45280,4	0	2	52,6	45279,3	0	0	2	51 808,2
AA*65	1780,9	47020,8	23	24	36	256,5	47048,8	20	8	242,7	47048,8	0	20	8	51 961,9
BB*30	5,1	32843,9	9	4	5	3,6	36144,2	11	3	3,7	36144,2	0	11	3	41 111,0
BB*35	58,9	36377,3	25	7	18	29,3	37122,8	2	6	26,2	37122,8	0	2	6	41 332,9
BB*40	241,6	37788,0	55	1	48	108,7	39337,7	2	16	131,2	39337,7	0	2	22	41 477,1
BB*45						66,7	41646,1	3	3	50,9	41646,1	0	3	3	41 699,5
BB*50						1245	46505,4	38	69	1701,3	46505,4	56	25	89	51 719,1
BB*55															
BB*60	420,7	63148,6	53	10	25	74,0	65265,4	16	4	155,6	65172,6	3	12	7	72 184,3
BB*65						1719,5	66055,7	52	72						72 394,5

CHAPITRE 5 ARTICLE 2 : A POPULATION-BASED METAHEURISTIC FOR THE PICKUP AND DELIVERY PROBLEM WITH TIME WINDOWS AND LIFO LOADING

Recopié avec permission, M. Cherkesly, G. Desaulniers et G. Laporte, (2015), A Population-Based Metaheuristic for the Pickup and Delivery Problem with Time Windows and LIFO Loading. *Computers & Operations Research*, publié en ligne le 11 avril 2015, `http://dx.doi.org/10.1016/j.cor.2015.04.002`.

5.1 Introduction

This paper proposes a population-based metaheuristic for the pickup and delivery problem with time windows and last-in-first-out (LIFO) loading (PDPTWL). In the pickup and delivery problem (PDP), a set of vehicles is used to complete several requests. A request corresponds to transporting goods (or items) from a pickup node to a delivery node. The LIFO policy means that when a pickup node is visited, its corresponding item is loaded on top of a linear stack, and an item can only be delivered if it is on top of the stack. Figure 5.1 depicts two vehicle routes where 0^+ and 0^- represent the depot at the beginning and the end of the route, 1^+ and 1^- represent the pickup and the delivery nodes for item 1, and 2^+ and 2^- represent the pickup and delivery nodes for item 2. The first route repects the LIFO policy, but not the second one. In route 2, item 1 is delivered when item 2 is on top of the stack, meaning that the LIFO policy is not respected. Each item has a specified load, and each pickup or delivery node has a given service time and a time window during which service must start. We consider an unlimited fleet of identical capacitated vehicles. A vehicle route is feasible if it respects (i) the vehicle capacity, (ii) the time windows, and (iii) the LIFO policy. Note that there is a single depot, and each vehicle route starts and ends at the depot. Travel costs are proportional to the total traveled distance. The PDPTWL consists of first minimizing the number of vehicles used, and then the total distance traveled, subject to the feasibility constraints.

To the best of our knowledge, the PDPTWL has only been studied by Cherkesly *et al.* (2014) who have developed three exact branch-price-and-cut algorithms which can solve instances with up to 75 requests within one hour of computation time. In addition, a number of heuristics have been proposed for variants of the problem, namely the vehicle routing problem with time windows (VRPTW) (see Bräysy and Gendreau (2005a,b); Vidal *et al.* (2013a) for a survey, and Vidal *et al.* (2013b) for a state-of-the-art heuristic), the pickup and

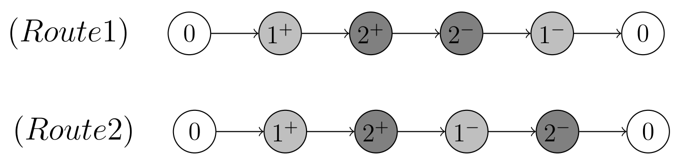


Figure 5.1 – The LIFO policy is respected in route 1, but not in route 2 because item 1 cannot be delivered without first removing item 2 from the vehicle.

delivery problem with time windows (PDPTW) (see Berbeglia *et al.* (2007); Parragh *et al.* (2008a,b) for a survey, and Li and Lim (2003), Bent and Van Hentenryck (2006), and Ropke and Pisinger (2006) for recent heuristics), the traveling salesman problem with pickup and delivery and LIFO loading (see Cassani (2004), Carrabs *et al.* (2007b), and Li *et al.* (2011) for recent heuristics), the pickup and delivery problem with LIFO loading (see Ambrosini *et al.* (2004); Gao *et al.* (2011); Li *et al.* (2011), and Cheang *et al.* (2012) for recent heuristics), and the traveling salesman problem with pickups, deliveries and handling costs (see Battarra *et al.* (2010) for a branch-and-cut algorithm and Erdoğan *et al.* (2012) for a heuristic).

Among the algorithms put forward for the PDPTWL variants, two main heuristic search principles emerge and will constitute the basis of this study. The first is the population-based heuristic of Vidal *et al.* (2013b) which can solve many variants of the VRPTW, namely the periodic VRPTW, the multi-depot VRPTW, and the site-dependent VRPTW. One important feature of this algorithm is the population management strategy which allows to diversify the solution pool. The second is the adaptive large neighborhood search (ALNS) of Ropke and Pisinger (2006) for the PDPTW. The ALNS performs a local search by first removing requests and then reinserting them. The algorithm chooses at each iteration one of three removal operators and one of two reinsertion operators, according to their past performance.

The main objective of this paper is to propose a population-based metaheuristic capable of solving large-sized instances of the PDPTWL. In this algorithm, a set of initial solutions is obtained through the application of a greedy randomized adaptive search procedure (GRASP). Two population-based methods are then used to generate offspring. The first method combines routes from the solution pool, whereas the second applies an adapted order crossover operator. In the second method, a diversification strategy inspired by that of Vidal *et al.* (2012) is used to update the solution pool. Local search based on the ALNS principle is then performed on each solution in order to first minimize the number of vehicles, and then the total traveled distance. Computational results are reported for instances with 30 to 300 requests, and show that the second method used to generate offspring produces better quality solutions.

The remainder of this paper is structured as follows. Section 5.2 presents the mathematical notation used. Section 5.3 describes the construction of initial solutions, the population-based metaheuristic, and the local search operators. Computational results are reported in Section 5.4, and conclusions follow in Section 5.5.

5.2 Problem description

Let $I = \{1, ..., n\}$ denote a set of n items, also called requests, and let $P = \{1^+, ..., n^+\}$ and $D = \{1^-, ..., n^-\}$ represent the sets of pickup and delivery nodes. With each request $i \in I$ are associated a pickup node $i^+ \in P$ and a delivery node $i^- \in D$. The depot is represented by two nodes 0^+ and 0^- which are respectively called the origin and the destination depot. The PDPTWL can be defined on a directed graph $G = (N, A)$, where $N = P \cup D \cup \{0^+, 0^-\}$ is the set of nodes and A is the set of arcs. An arc $(i, j) \in A$ must respect the predecence constraints, and the LIFO policy, i.e., for each request $i \in I$, the arc (i^-, i^+) is not generated, and for each pair of requests $i, j \in I$ where $i \neq j$, the arc (i^+, j^-) is not created. Note that because the algorithm allows intermediate solutions containing infeasible routes with respect to time windows or capacity constraints, we allow arcs that violate the time windows or the capacity constraints.

For each request $i \in I$, q_i represents the load of the items to be picked up at node $i^+ \in P$ and delivered at node $i^- \in D$. We denote by $q_{i^+} > 0$ the load picked up at node $i^+ \in P$, and by $q_{i^-} = -q_{i^+}$ the load delivered at node $i^- \in D$, with $q_i = 0$ if $i \in \{0^+, 0^-\}$. A time window $[a_i, b_i]$ is associated with each node $i \in N$, where a_i and b_i represent the earliest and the latest times at which the service can begin at node i, and waiting before the beginning of the time window is allowed. The time windows of the origin and the destination nodes 0^+ and 0^- are unconstraining. An unrestricted set of K identical vehicles of capacity Q is available. With each arc $(i, j) \in A$ are associated a nonnegative travel distance d_{ij}, and a nonnegative travel time t_{ij} which includes the service time at node i if any. We assume that the triangle inequality holds for travel distances and travel times.

Let $\mathcal{R}$ be the set of routes in a solution, and let $R \in \mathcal{R}$ be a route that can be denoted as $R = (i_0 = 0^+, i_1, i_2, ..., i_m = 0^-)$, where i_ρ is the ρ^{th} node visited in R. For each visited node $i_\rho \in R$, $1 \leq \rho \leq m$, we compute the total load of the vehicle after visiting it as $l(i_\rho) = l(i_{\rho-1}) + q_{i_\rho}$, with $l(i_0) = 0$. We define $t(i_\rho) = \max\{t(i_{\rho-1}) + t_{i_{\rho-1}, i_\rho}, a_{i_\rho}\}$ as the time at which service starts at node i_ρ, $1 \leq \rho \leq m$ with $t(i_0) = 0$. Note that because we accept infeasible intermediate solutions, it is possible that $t(i_\rho) > b_{i_\rho}$, i.e., the service at node i_ρ could start after the end of its time window. Thus, we define $\tau(i_\rho) = \min\left\{\max\{\tau(i_{\rho-1}) + t_{i_{\rho-1}, i_\rho}, a_{i_\rho}\}, b_{i_\rho}\right\}$ the service start time with time-warp at node i_ρ, $1 \leq \rho \leq m$, with $\tau(i_0) = 0$, and $w(i_\rho) =$

$\max\{0, \tau(i_{\rho-1}) + t_{i_{\rho-1},i_\rho} - b_{i_\rho}\}$ the time-warp needed at this node to respect the time windows, with $\tau(i_0) = 0$. This concept was introduced by Nagata *et al.* (2010) and extended by Vidal *et al.* (2013b) to be applied to all operators used in local search algorithms for routing problems, and is used when a vehicle arrives after the time window of a customer in order to reach the end of the time window. Each route R is associated with a set $I_R = \{i \in I | i \in R\}$ of completed requests, where $i \in R$ indicates that request i is served in route R. We denote by $d(R) = \sum_{\rho=0}^{m-1} d_{i_\rho,i_{\rho+1}}$ the total distance of route R, by $q(R) = \sum_{\rho=0}^{m} \max\{0, l(i_\rho) - Q\}$ the excess capacity of route R, and by $w(R) = \sum_{\rho=0}^{m} w(i_\rho)$ the time window violation of route R. If a route is feasible with respect to capacity, then $q(R) = 0$; otherwise $q(R) > 0$. Similarly if a route is feasible with respect to time windows, then $w(R) = 0$; otherwise $w(R) > 0$. Each route R has a cost

$$c(R) = d(R) + \alpha q(R) + \beta w(R), \tag{5.1}$$

where α and β are positive user-defined parameters.

Let $\mathcal{S}_f$ denote the set of feasible solutions, $\mathcal{S}_i$ the set of infeasible solutions, and $\mathcal{S} = \mathcal{S}_f \cup \mathcal{S}_i$ the solution pool. For each solution $S \in \mathcal{S}$, we denote by $\mathcal{R}_S$ the set of all routes in S. Each solution S has a cost $c(S) = \kappa|\mathcal{R}_S| + \sum_{R\in\mathcal{R}_S} c(R)$, where κ is a positive user-defined parameter, and $|\mathcal{R}_S|$ is the number of vehicles used.

The PDPTWL consists of determining a set of feasible routes covering exactly once each request with respect to capacity constraints, time windows, and the LIFO policy such that the number of vehicles is first minimized, and then the total traveled distance is minimized.

5.3 Description of the metaheuristic

We now describe the population-based metaheuristic we have designed for the PDPTWL. It proceeds in three phases. The first phase consists of creating an initial solution pool by means of a GRASP, a concept introduced by Feo and Resende (1989, 1995). The cost evaluation for each request is based on a savings criterion. Each solution goes through a local search phase to first minimize the number of vehicles, and then the total traveled distance. The GRASP generates feasible solutions only, but infeasibility is allowed in the local search phase. The second phase consists of creating additional solutions by selecting routes from different solutions and creating an offspring. Local search is applied to the offspring. Finally, the third phase consists of selecting two parents, and creating offspring by means of an adapted crossover operator. Each offspring is educated through local search. Note that the first phase of the algorithm is essential because it generates the initial solution pool, but the second and third phases are not. Thus, several variants of the algorithm are possible. For example one could start with the first phase and continue directly to the third one.

In the local search phase, a move, i.e., defined by one of the operators described in Section 5.3.4, can produce an infeasible solution with respect to capacity constraints or time windows. If such a move is accepted and increases the excess capacity or the time window violation, the values of α or β in formula (5.1) are increased as follows : $\alpha_{new} = \alpha_{old}(1 + \alpha_{old})$, and $\beta_{new} = \beta_{old}(1 + \beta_{old})$, where α_{old} and β_{old} are the old values of α and β, and α_{new} and β_{new} are the new values. When starting a new iteration in one of the three phases, α and β are reset to their original values. The adapted order crossover can select feasible and infeasible solutions to create offspring.

Here follows a description of the three phases of the algorithm and of the local search operators.

5.3.1 Phase I : Building the set of initial solutions

The set of initial solutions is created with a GRASP, which consists of a greedy randomized construction phase followed by a local search phase (see Resende and Ribeiro (2010)). In our GRASP, the construction phase creates initial feasible solutions by sequentially adding feasible routes to a solution. For each route, requests are added sequentially according to a saving criterion. Let $(i_0 = 0^+, i_1, ..., i_m = 0^-)$ be the current route. The best insertion position of each unvisited node i_u in the route is the one yielding

$$\rho^* \in \underset{\rho \in \{0,\ldots,m-1\}}{\operatorname{argmin}} \{d_{i_\rho,i_u} + d_{i_u,i_{\rho+1}} - d_{i_\rho,i_{\rho+1}}\}. \tag{5.2}$$

If formula (5.2) returns several arguments, the one with the lowest value of ρ is kept. The best insertion position of each unvisited request consists in finding a pair (ρ^+, ρ^-) that determines the insertion position of both its pickup and delivery nodes as long as it respects the time windows, the capacity constraint, and the LIFO policy. This can be computed in $O(n^3)$ operations. In order to accelerate the computation time, we apply an insertion strategy that can be executed in $O(n^2)$ operations. First, for each unvisited request u, the feasible positions for the pickup node are computed and sorted in non-decreasing order of position according to formula (5.2). The best insertion position for node u^+ is determined as long as it respects the capacity constraint and the time windows, and the best feasible insertion for u^- is determined as long as it respects the capacity constraint, the time windows, and the LIFO policy. If no such position exists, the process is reiterated with the second best position of u^+, and so on until a feasible position is found for u^- or no more feasible position exists for u^+. This can also be achieved by first finding the best position for u^-, and then finding the best position for u^+. A similar idea was used by Cordeau and Laporte (2003) in the context of the dial-a-ride problem. After the insertion cost has been computed for each unvisited request u, we create a restricted candidate list containing at most the η requests with the best savings criterion. A request is then chosen at random from the restricted candidate list and is inserted in the current route. When no more requests can be added to a route and some requests remain unvisited, a new route is created. Each solution goes through a local search phase and the ensuing solution is added to the solution pool $\mathcal{S}$.

5.3.2 Phase II : Creating new solutions from existing routes

In the second phase of the algorithm, offspring are created by selecting existing routes from the solution pool. This algorithm is similar to that of Rochat and Taillard (1995) who developed a diversification and intensification strategy for the vehicle routing problem (VRP). This procedure is implemented as follows. For each solution $S \in \mathcal{S}$, and each route $R \in S$ we compute

$$c'(R) = c(S) + \gamma v(R)c(S) - \psi|I_R|, \tag{5.3}$$

where $v(R)$ is the number of times route R has been chosen to generate an offspring, and γ and ψ are positive user-defined parameters. Offspring are then created by selecting routes sequentially according to a certain probability. More specifically, if $\mathcal{R}^*$ is the ordered set of routes R in decreasing order of $c'(R)$, the route with position ω in the set $\mathcal{R}^*$ is chosen with probability

$$p_{insert} = \frac{2\omega}{|\mathcal{R}^*|(|\mathcal{R}^*|+1)}. \tag{5.4}$$

When a route is added to the current solution, $\mathcal{R}^*$ is updated by eliminating all routes $R \in \mathcal{R}^*$ having requests in common with the added route or having the same cost. Routes are added until the solution visits all the requests or there are no more routes in $\mathcal{R}^*$. If some requests remain unvisited, an attempt is made to reinsert them into existing routes with the request insertion operators to be described in Section 5.3.4. If this is impossible, additional routes are created with a GRASP. Each solution goes through a local search phase and the resulting solution is added to the solution pool $\mathcal{S}$.

5.3.3 Phase III : An adapted order crossover operator

The third phase of the algorithm consists of creating an offspring from two parent solutions with an adaptation to the LIFO policy of the crossover operator proposed by Prins (2004). In order to diversify the choice of parent solutions, we apply a procedure based on that proposed by Vidal *et al.* (2012). All solutions created in Phases I and II are added to the solution pool. Before the first iteration of Phase III, minimal and maximal population sizes, denoted $\mathcal{S}_{min}$ and $\mathcal{S}_{max}$, are defined. The diversification strategy is then launched when the solution pool reaches its maximal size, and the solutions with the highest biased fitness value are eliminated from the pool until it reaches its minimal size. For a solution $S \in \mathcal{S}$, the biased fitness value is defined as

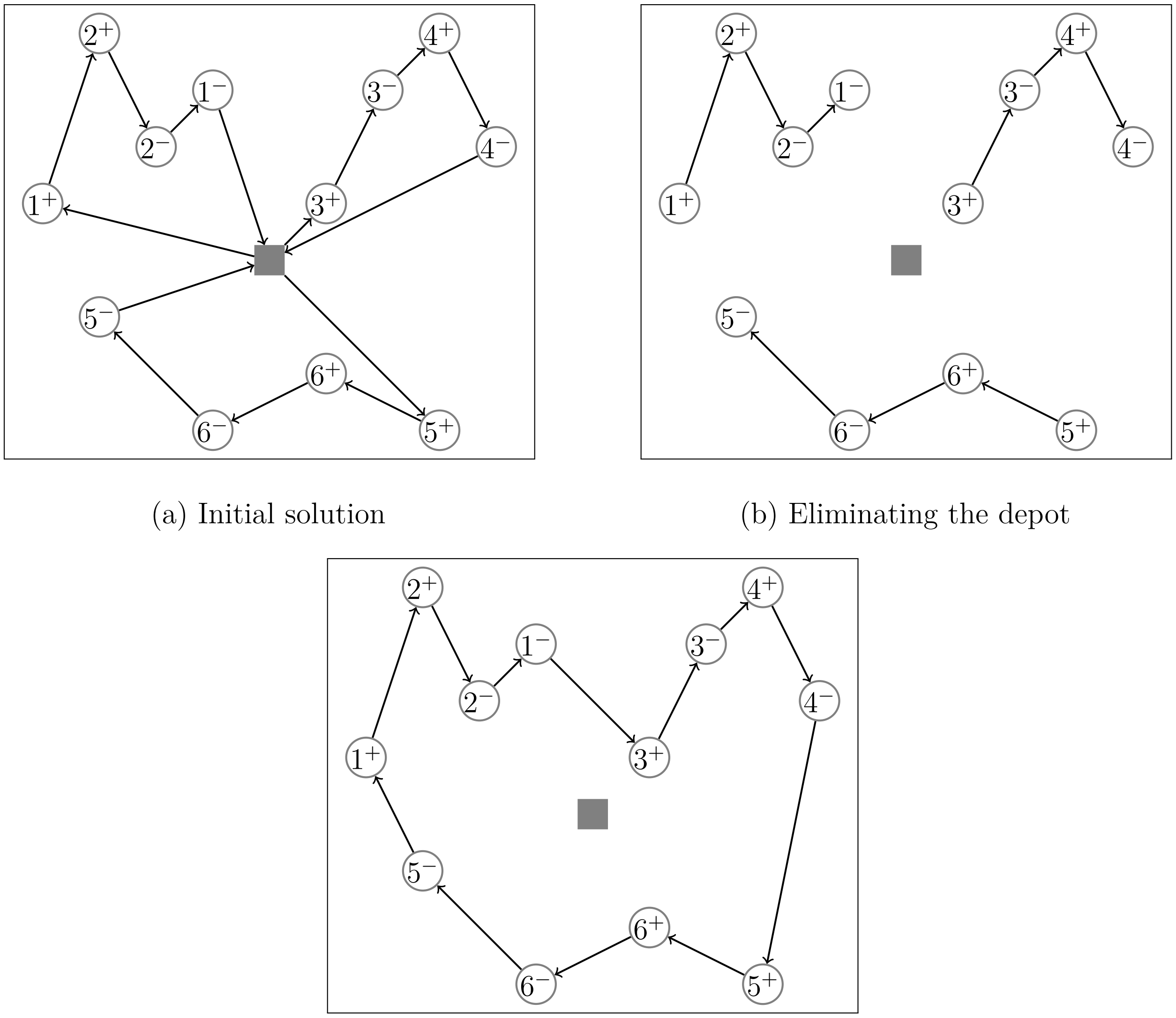
$$BF(S) = fit(S) + (1 - \zeta)dc(S), \tag{5.5}$$

where $0 \leq \zeta \leq 1$ is a positive user-defined parameter, and $fit(S)$ and $dc(S)$ are, respectively, the rank of a solution S with respect to its cost $c(S)$ and its similarity with the other solutions. The similarity of two solutions is measured by the numbers of arcs they have in common. The idea is to keep in the population solutions that have low costs and are sufficiently different from each other.

A giant route is created as in Prins (2004) by first removing the depot from the solution and then merging all routes into a single route while keeping the node sequence of each route intact (Figure 5.2). The giant route representation of an initial feasible solution respects the capacity constraints and the LIFO policy, but may not respect the time windows. Note that this giant route will contain $2n$ nodes.

To create the giant route representation of an offspring, two parent solutions are first randomly selected from the diversified solution pool, and two break-points x and y with $x < y \leq 2n$ are then randomly selected to create it. Figure 5.3 shows the giant route representations of two offspring created with the adapted order crossover when the break-points are at $x = 2$ and $y = 6$, and illustrates how the giant route representation of the first offspring is created. The following procedure explains the creation of the giant route representation of an offspring in six steps :

- Step 1 (Path from parent 1) : Copy the path from positions x to y from parent 1 to their corresponding positions in the offspring.
- Step 2 (Node deletions) : Delete the nodes for which the delivery node has been visited but not the pickup node, i.e., node 1^-. All the following nodes are shifted to the left.
- Step 3 (Node completions from parent 1) : If there remain unfilled positions between



(a) Initial solution

(b) Eliminating the depot

(c) Creating a single route

Figure 5.2 – Creation of a giant route

x and y in the offspring, fill them by taking nodes from parent 1, beginning at position $y + 1$. If necessary, repeat Steps 2 and 3.

— Step 4 (Request completions) : Once all the positions between x and y are filled, add the delivery nodes of uncompleted requests, i.e., node 2^- in Figure 5.3.

— Step 5 (Completions from parent 2) : Sweep parent 2 circularly from $y + 1$ to fill the last positions of the offspring with unvisited nodes. Repeat step 2.

— Step 6 (Completions and shifting) : If there are uncompleted requests, then visit the corresponding delivery nodes, and shift the preceding nodes to the left. If there remain unfilled positions from 1 to x, fill them with respect to parent 2.

Position	1	2	3	4	5	6	7	8	9	10
Parent 1	1^+	5^+	5^-	3^+	3^-	1^-	2^+	4^+	4^-	2^-
Parent 2	3^+	4^+	5^+	5^-	4^-	3^-	1^+	1^-	2^+	2^-
Step 1 : Path from parent 1		5^+	5^-	3^+	3^-	1^-				
Step 2 : Node deletions		5^+	5^-	3^+	3^-					
Step 3 : Node completions from parent 1		5^+	5^-	3^+	3^-	2^+				
Step 4 : Request completions		5^+	5^-	3^+	3^-	2^+	2^-			
Step 5 : Completions from parent 2		5^+	5^-	3^+	3^-	2^+	2^-	1^+	1^-	4^+
Step 6 : Completions and shifting	5^+	5^-	3^+	3^-	2^+	2^-	1^+	1^-	4^+	4^-
Offspring 1	5^+	5^-	3^+	3^-	2^+	2^-	1^+	1^-	4^+	4^-
Offspring 2	4^+	5^+	5^-	4^-	1^+	1^-	2^+	2^-	3^+	3^-

Figure 5.3 – Creation of the giant route representation of an offspring with the adapted order crossover

In order to create the giant route representation for the second offspring, we apply the same procedure by swapping the roles of parents 1 and 2.

An offspring is then created by splitting its corresponding permutation of nodes, i.e., its giant route representation. The first route is created by selecting the node in the first position of the giant route representation. Nodes are added sequentially until the number of requests in the current route reaches the minimum number of requests, computed as the average number of requests per vehicle in the best feasible solution found so far. All uncompleted requests are completed and the process is repeated to create additional routes until all the requests are performed in the current solution.

5.3.4 Local search

Our local search has been implemented in two steps. The first step decreases the number of vehicles, whereas the second one decreases the total traveled distance. Infeasible solutions are not accepted in the first step, but they are in the second step. To decrease the number of vehicles, the route with the lowest number of visited requests (or one of them if there are several) is eliminated from the current solution, and an attempt is first made to reinsert the unvisited requests into existing routes with the request insertion operators to be described in Section 5.3.4. If no reinsertion is possible, local search is then applied to the current solution, i.e., the solution obtained after removing a route. Evaluating the cost of a move is achieved according to its impact on the total traveled distance. When no more improving moves are found and no reinsertion is possible, a new route is created to accomodate the unvisited requests. This step is repeated until no more feasible solution with fewer vehicles can be found. To decrease the total traveled distance, local search is applied to the current solution. Evaluating the cost of a move is achieved according to the impact on the total traveled distance, the excess capacity and the time window violation, as in formula (5.1). When no more local search operators can find a better solution, an infeasible solution is repaired according to a user-defined probability p_{repair}. For each infeasible route, the repair procedure consists of sequentially eliminating the requests for which the time windows or the capacity constraint are first violated. An attempt is then made to reinsert the eliminated requests with the request insertion operators to be described in Section 5.3.4. If no insertion is possible, local search is applied to the current solution. When no more improving moves are found and no reinsertion is possible, a new route is created to accomodate the unvisited requests.

This section describes four inter-route operators, four intra-route operators and one perturbation operator. We have adapted and extended the inter-route operators proposed by Cheang *et al.* (2012), and the intra-route operators of Cassani (2004). Finally, the perturbation operator corresponds to a restricted version of the ALNS heuristic of Ropke and Pisinger (2006) for the PDPTW. The operators have been implemented in a variable neighborhood descent fashion, except for the perturbation operator which is used when the current solution cannot be improved with the other operators.

Inter-route operators

The four inter-route operators consist of exchanging requests (request exchange and multiple request exchange), and of relocating requests from one route to another (request relocate and multiple request relocate).

Inter-route request exchange

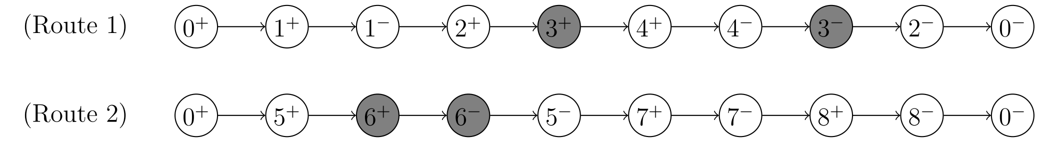
The inter-route request exchange selects two requests i and j served by different vehicles. Three types of exchanges are possible for requests i and j. The first exchange, denoted by type 1, consists of exchanging the position of nodes i^+ and j^+, and the position of nodes i^- and j^-. Another one, denoted by type 2, replaces i^+ with (j^+, j^-), j^+ with i^+, and j^- with i^-. The third one, denoted by type 3, replaces i^- with (j^+, j^-), j^+ with i^+, and j^- with i^-. Figure 5.4 depicts these exchanges for requests $i = 3$ and $j = 6$.

Inter-route request relocate

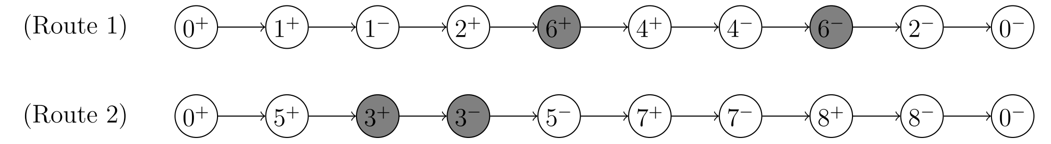
The inter-route request relocate operator consists of moving a request i in another route. Figure 5.5 depicts the relocation of request 3 in route 2. If node 3^+ is inserted before 5^+, four feasible insertions of 3^- are possible. The first possible insertion relocates node 3^- before node 5^+. The second possible insertion relocates node 3^- after node 5^-. The third possible insertion relocates node 3^- after node 7^-. The fourth possible insertion relocates node 3^- after node 8^-. Node 3^+ could also be positioned elsewhere in the route. All moves compatible with the LIFO policy are explored.

Inter-route multiple request exchange

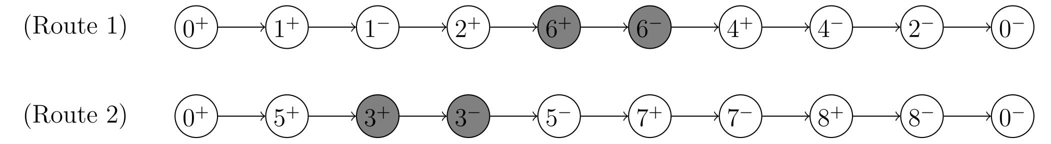
For each request $i \in I$, we denote by $\mathcal{H}(i)$ a path starting at node $i^+ \in P$, and ending at node $i^- \in D$. In Figure 5.6a, we denote by $\mathcal{H}(3)$ the path $(3^+, 4^+, 4^-, 3^-)$. The inter-route multiple request exchange selects two requests i and j performed by two different vehicles and their corresponding paths $\mathcal{H}(i)$ and $\mathcal{H}(j)$. It then exchanges $\mathcal{H}(i)$ with $\mathcal{H}(j)$. Figure 5.6 depicts an example where the positions of paths $\mathcal{H}(3)$ and $\mathcal{H}(7)$ are exchanged. The exchange of paths starting and ending at different requests is also allowed. We restrict ourselves to the following


(Route 1)
$0^+ \to 1^+ \to 1^- \to 2^+ \to 3^+ \to 4^+ \to 4^- \to 3^- \to 2^- \to 0^-$
(Route 2)
$0^+ \to 5^+ \to 6^+ \to 6^- \to 5^- \to 7^+ \to 7^- \to 8^+ \to 8^- \to 0^-$

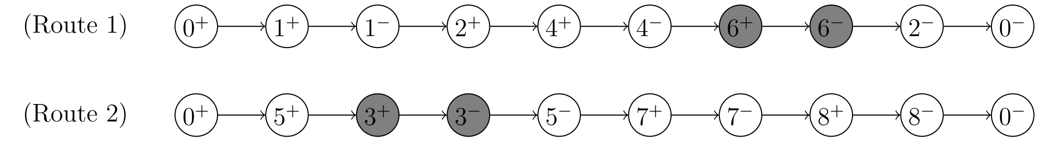
(a) Initial solution


(Route 1)
$0^+ \to 1^+ \to 1^- \to 2^+ \to 6^+ \to 4^+ \to 4^- \to 6^- \to 2^- \to 0^-$
(Route 2)
$0^+ \to 5^+ \to 3^+ \to 3^- \to 5^- \to 7^+ \to 7^- \to 8^+ \to 8^- \to 0^-$

(b) Type 1 : positions of nodes 3^+ and 6^+ are exchanged and positions of nodes 3^- and 6^- are exchanged


(Route 1)
$0^+ \to 1^+ \to 1^- \to 2^+ \to 6^+ \to 6^- \to 4^+ \to 4^- \to 2^- \to 0^-$
(Route 2)
$0^+ \to 5^+ \to 3^+ \to 3^- \to 5^- \to 7^+ \to 7^- \to 8^+ \to 8^- \to 0^-$

(c) Type 2 : nodes 6^+ and 6^- take the position of node 3^+


(Route 1)
$0^+ \to 1^+ \to 1^- \to 2^+ \to 4^+ \to 4^- \to 6^+ \to 6^- \to 2^- \to 0^-$
(Route 2)
$0^+ \to 5^+ \to 3^+ \to 3^- \to 5^- \to 7^+ \to 7^- \to 8^+ \to 8^- \to 0^-$

(d) Type 3 : nodes 6^+ and 6^- take the position of node 3^-

Figure 5.4 – Inter-route request exchange operator : possible exchanges for requests $i = 3$ and $j = 6$

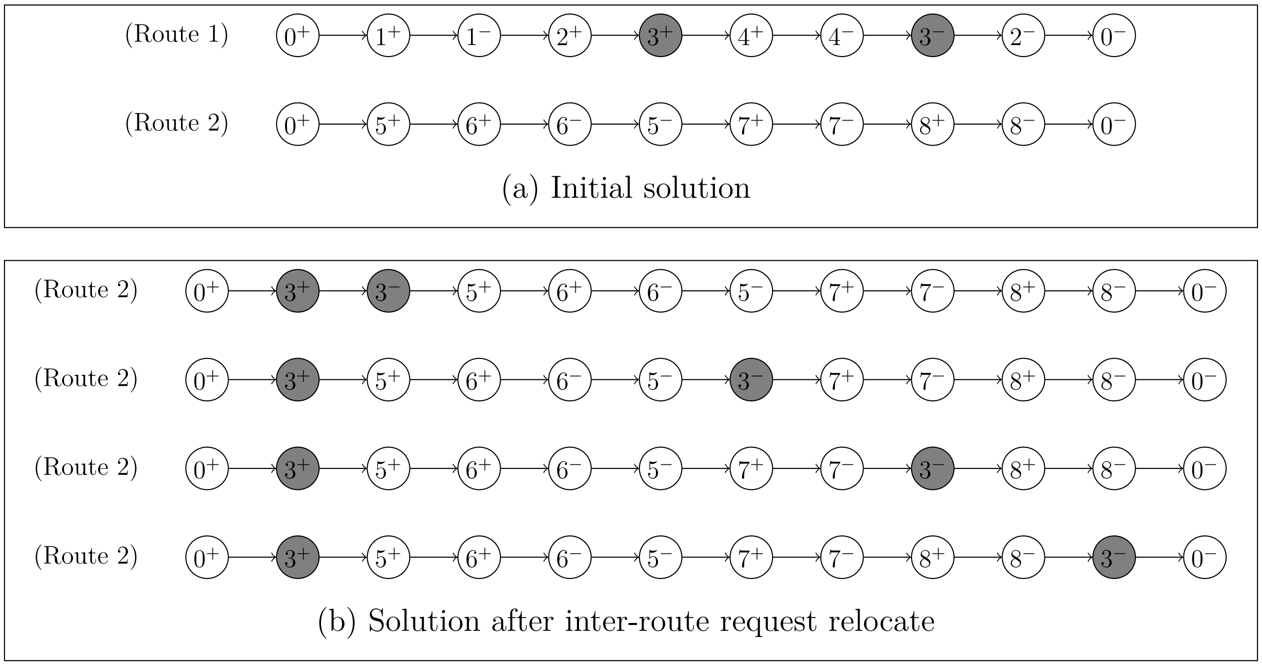


Figure 5.5 – Inter-route request relocate operator : inserting request 3 in route 2

two cases : (1) a path $(i^+, ..., k^-)$ can be exchanged with $\mathcal{H}(j)$ if the vehicle is empty before visiting node i^+ and the successor of node k^- in the current solution is 0^- ; and (2) such a path can also be exchanged with a path $(j^+, ..., h^-)$ if the vehicle is empty before visiting node j^+ and the successor of node h^- in the current solution is 0^-. Figure 5.7 provides an example where the positions of path $(7^+, 7^-, 8^+, 8^-)$ and of path $\mathcal{H}(3)$ are exchanged.

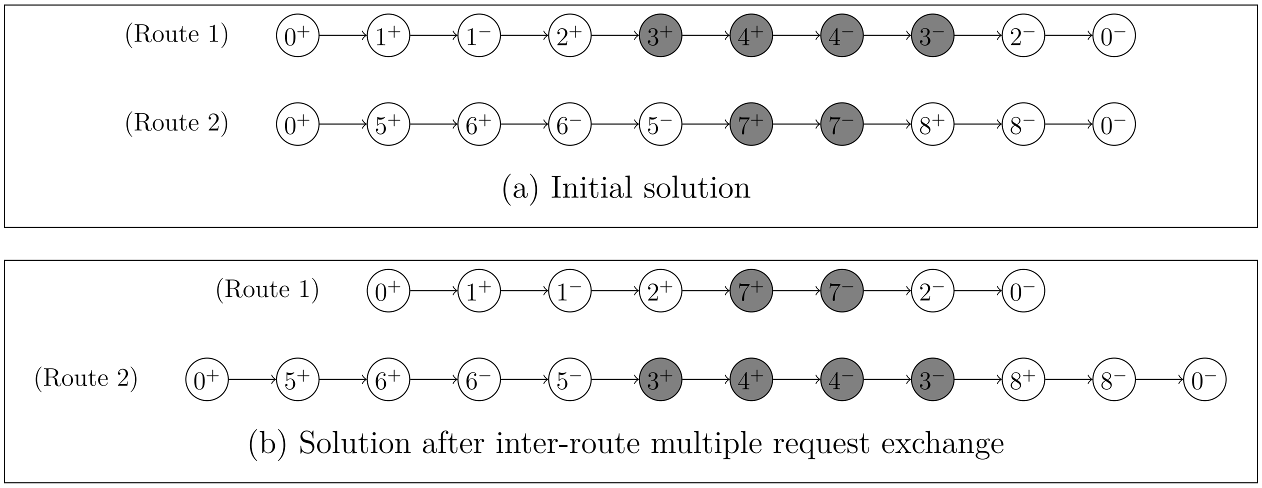
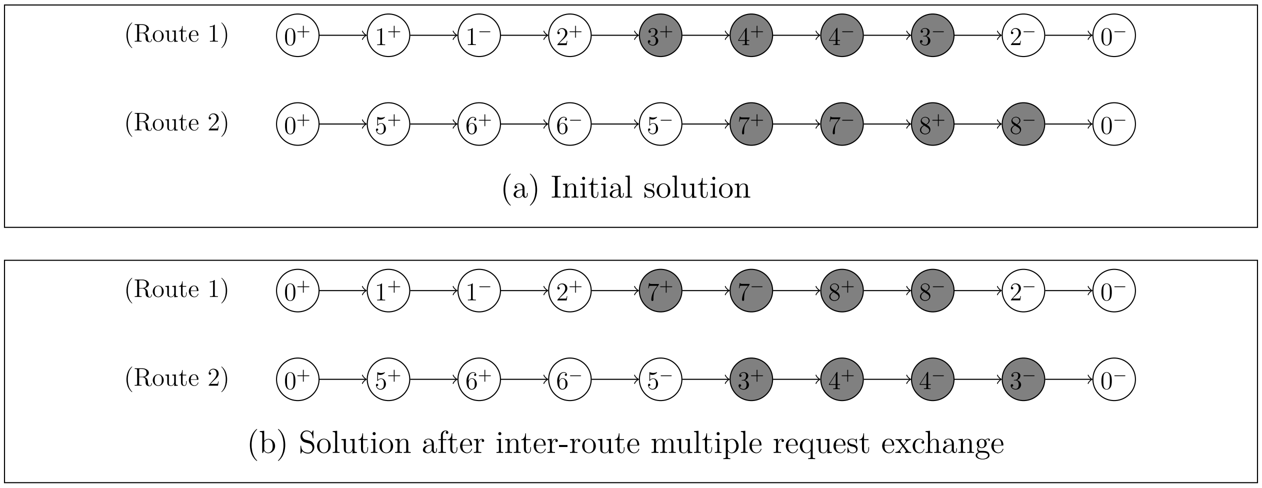


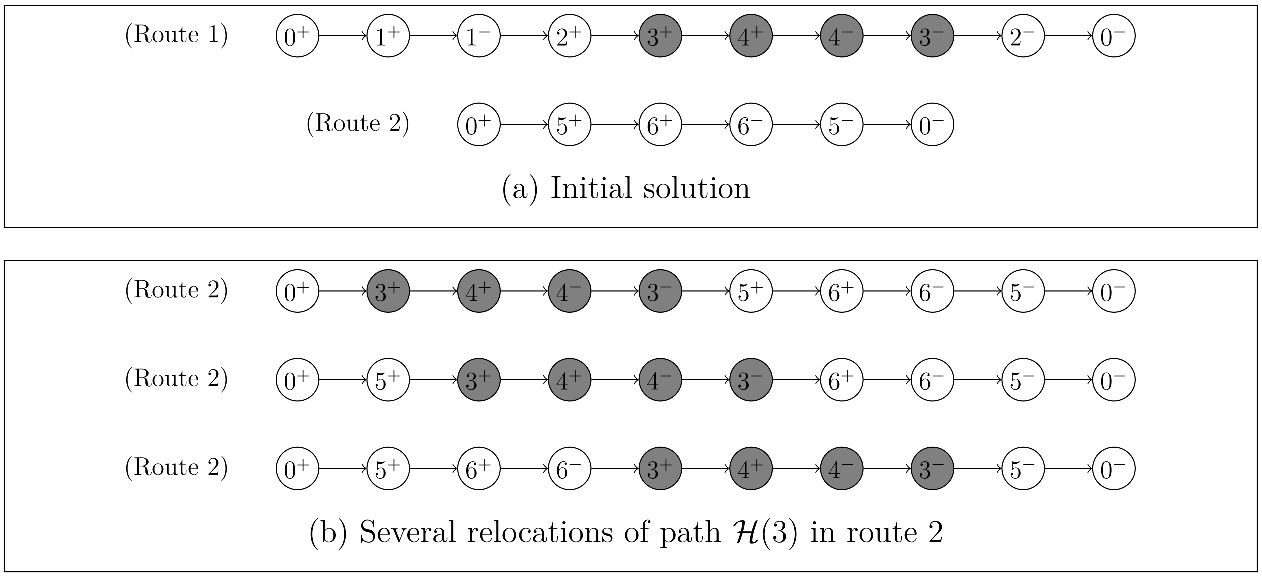
Figure 5.6 – Inter-route multiple request exchange operator : we wish to exchange paths $\mathcal{H}(3)$ and $\mathcal{H}(7)$



(a) Initial solution

(b) Solution after inter-route multiple request exchange

Figure 5.7 – Inter-route multiple request exchange operator : we wish to exchange paths $(3^+, 4^+, 4^-, 3^-)$ and $(7^+, 7^+, 8^+, 8^-)$



(a) Initial solution

(b) Several relocations of path $\mathcal{H}(3)$ in route 2

Figure 5.8 – Inter-route multiple request relocate operator : examples of possible moves for path $\mathcal{H}(3)$ in route 2

Inter-route multiple request relocate

The inter-route multiple request relocate selects a path $\mathcal{H}(i)$, and moves it in another route of the current solution. Figure 5.8 illustrates several ways of relocating path $\mathcal{H}(3)$ in route 2 with respect to the position of request 5. The first example relocates $\mathcal{H}(3)$ before node 5^+, the second relocates $\mathcal{H}(3)$ between nodes 5^+ and 6^+, and the third relocates $\mathcal{H}(3)$ before node 5^-. All moves compatible with the LIFO policy are explored.

Intra-route operators

The four intra-route operators consist of exchanging positions of requests (request exchange and multiple request exchange), and of relocating requests served by the same vehicle (request relocate and multiple request relocate).

Intra-route request exchange

The intra-route request exchange operator selects two requests i and j in a given route, and exchanges the position of nodes i^+ and j^+, and the position of nodes i^- and j^-. Figure 5.9 provides an example where the positions of requests 1 and 3 are exchanged.

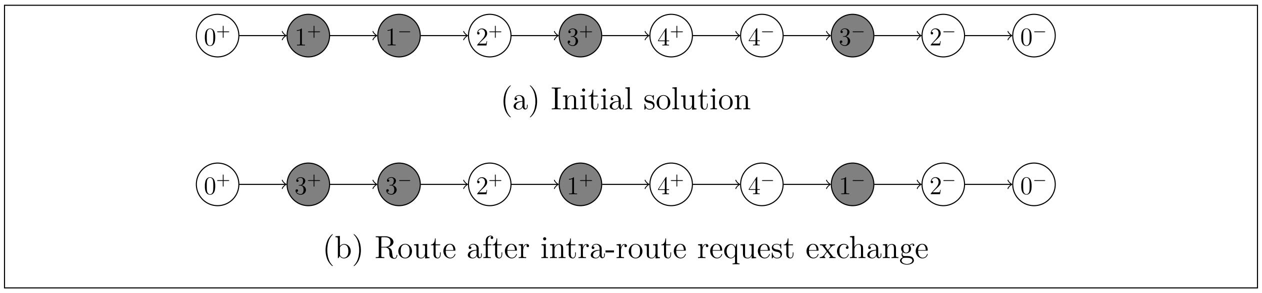


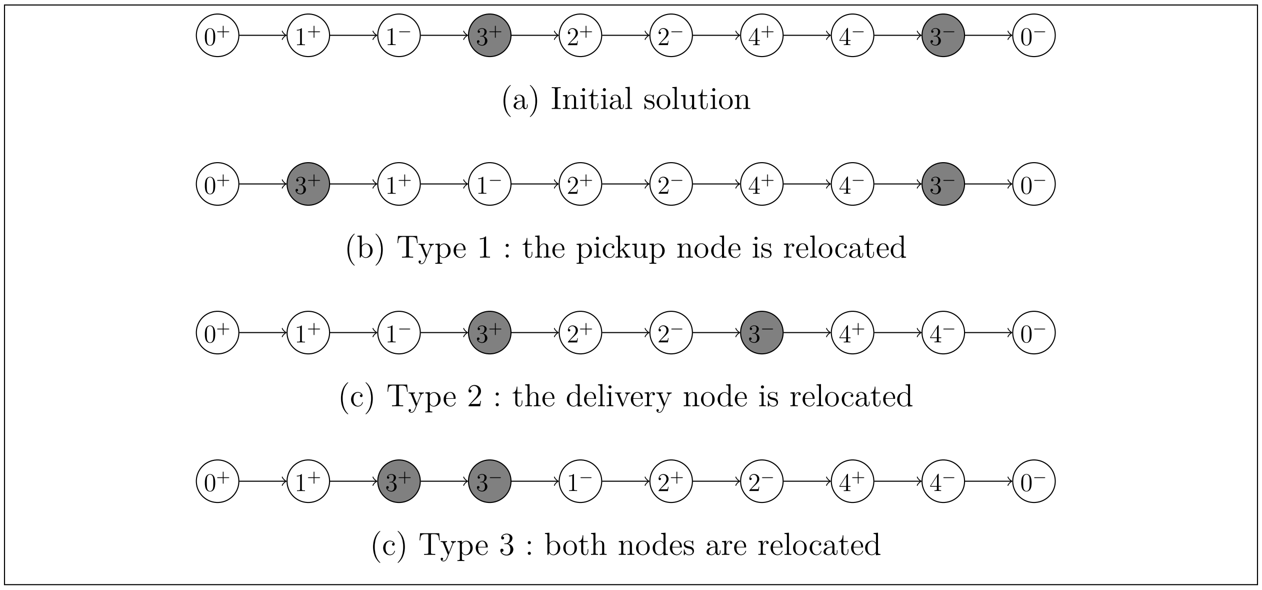
Figure 5.9 – Intra-route request exchange operator : exchanging the positions of requests 1 and 3

Intra-route request relocate

The intra-route request relocate operator selects a request i and tries to find a better position for it in the same route. This can be time consuming because a request is composed of a pickup node and a delivery node, and the complexity of this operation is $O\left(n^2\right)$. We have therefore implemented a linear-time procedure that sequentially relocates the pickup node and the delivery node. Three types of relocation moves are possible for request i. The first move, denoted by type 1, relocates only the pickup node i^+. The second one, denoted by type 2, relocates only the delivery node i^-. The third one, denoted by type 3, relocates both the pickup and delivery nodes, but inserts the delivery node directly after visiting the pickup node, i.e., arc (i^+, i^-) is used in the new route. Figure 5.10 illustrates these moves for the relocation of request 3.

Intra-route multiple request exchange

The intra-route multiple request exchange operator finds two non-embedded paths $\mathcal{H}(i)$ and $\mathcal{H}(j)$ and exchanges their respective positions. Figure 5.11 provides an example where the paths $\mathcal{H}(1)$ and $\mathcal{H}(3)$ are exchanged.



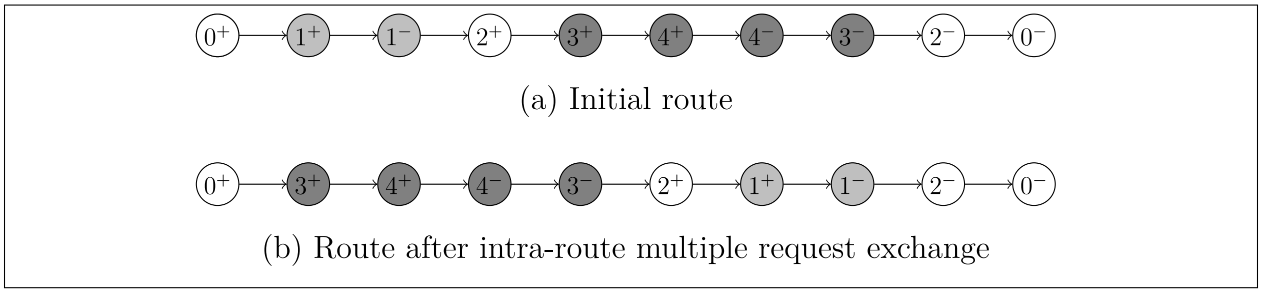
(a) Initial solution

(b) Type 1 : the pickup node is relocated

(c) Type 2 : the delivery node is relocated

(c) Type 3 : both nodes are relocated

Figure 5.10 – Intra-route request relocate operator : possible relocations of request 3



(a) Initial route

(b) Route after intra-route multiple request exchange

Figure 5.11 – Intra-route multiple request exchange operator : exchange of paths $\mathcal{H}(1)$ and $\mathcal{H}(3)$

Intra-route multiple request relocate

The intra-route multiple request relocate operator finds a path $\mathcal{H}(i)$, in a route and tries to find a better position for this path in the same route. Figure 5.12 provides an example where the path $\mathcal{H}(3)$ is relocated between nodes 1^+ and 1^-.

Perturbation operator

Implementing the above intra-route and inter-route operators often yields a local optimum. It is therefore necessary to perturb the current solution to achieve a better solution. Our perturbation operator is based on that of Ropke and Pisinger (2006) except that it only considers improving solutions. Our operator selects λ requests to be perturbed by first removing them from the current solution, and then reinserting them. The choice of a request removal ope-

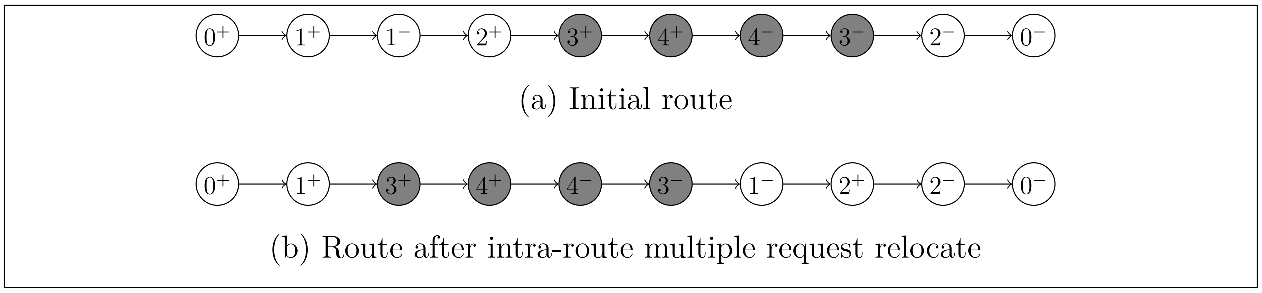


Figure 5.12 – Intra-route multiple request relocate operator : possibility to relocate path $\mathcal{H}(3)$

rator and of a request insertion operator is made through a roulette wheel procedure based on the past performance of the operators. This perturbation operator is implemented in a descent fashion, and the weights of the operators are reset when starting a new iteration. The weights of the operators are computed as follows :

$$\pi_{l,new} = \pi_{l,old}(1-r) + r\left(\frac{s_l}{\phi_l}\right), \tag{5.6}$$

where $\pi_{l,new}$ and $\pi_{l,old}$ are respectively the new and the old weights of operator l, r is a reaction factor, s_l is the score of operator l, and ϕ_l is the number of times operator l has been used in the current major iteration of the metaheuristic. The metaheuristic corresponds to three phases each with a user-defined number of major iterations ; doing 25 iterations of phase I and 25 iterations of phase II, implies that there are 50 major iterations. These parameters are reset at each major iteration of the metaheuristic. As explained in Ropke and Pisinger (2006), the reaction factor r determines the speed at which the weights are adjusted according to their performance. If $r = 0$, no adjustment is done and the weights remain at their original values. If $r = 1$, each weight will depend on its score obtained at the last iteration. The score s_l of an operator l is increased by σ_1 if the new solution is better than the old one, and by σ_2 if the new solution is worse than the previous one. In the latter case, the new solution is not accepted.

The following section describes the request removal operators, the request insertion operators and the choice of operator.

Request removal operators

We have implemented three request removal operators. The first is the Shaw (1997) removal operator where the relatedness of two requests is computed as

$$R(i,j) = (d_{i^+,j^+} + d_{i^-,j^-}) + (|t(i^+) - t(j^+)| + |t(i^-) - t(j^-)|) + (|q_i - q_j|). \tag{5.7}$$

A request is first randomly selected and eliminated from the current solution. Its $\lambda - 1$ closest requests according to the relatedness measure are then eliminated from the current solution. The second is a random removal : λ requests are randomly removed from a solution. The third is a worst removal heuristic. For a given request $i \in I$ it computes the saving resulting from the removal of i from its current route, and then selects the λ requests with the largest savings.

Request insertion operators

We have implemented two request insertion operators : one is based on a basic greedy insertion and the other one on a regret heuristic. The basic greedy insertion operator computes the insertion cost of each request in each route of the current solution as in Section 5.3.1 and inserts the request with the lowest cost. The second operator uses a regret criterion : for each unvisited request it identifies its best possible position and its second best position as explained in Section 5.3.1. The regret value is computed as the difference in cost between the second best and best positions, and the request with the largest regret value is inserted.

Table 5.1 – Characteristics of the PDPTWL instances

Group	Q	W	Δ
AA	22	60	45
BB	30	60	45
CC	18	120	15
DD	25	120	15

5.4 Computational results

The algorithm just described was implemented in C++ and was tested on two sets of instances. All tests were performed on a computer equipped with an Intel(R) Xeon(R) X5675 processor (3.07GHz). In this section, we report the computational results, and we study the impact of the local search operators and of the number of iterations on the quality of the solution. The user-defined parameters $(\alpha, \beta, p_{repair}, \eta, \kappa, \gamma, \psi, \mathcal{S}_{min}, \mathcal{S}_{max}, \zeta, \lambda, \pi_l, r, \sigma_1, \sigma_2)$ were set equal to $(2, 2, 0.75, 15, 10^4, 0.2, 20, 30, 50, 0.1, 0.15n, 0.35, 0.1, 15, 5)$. To determine these parameter values, they were first set to initial values that seemed reasonable. We then sequentially modified each of the parameters individually to measure its impact on the quality of the solutions, and we have kept the best setting found. During these tests, we have observed that the quality of the solution was not highly sensitive to the different values of the parameters in the considered range.

5.4.1 Instances

The first set of instances consists of modifications of the 40 instances proposed by Ropke and Cordeau (2009) for the PDPTW. The coordinates of the depot and of the pickup and delivery nodes are the same as in their instances. For each request $i \in I$, the original time windows for the pickup nodes are kept, but the time windows for the delivery nodes are delayed by Δ, a user-defined parameter called the delay, as follows : $a_{i^-} = a_{i^-} + \Delta$ and $b_{i^-} = b_{i^-} + \Delta$. The load q_i of request $i \in I$ was not modified, but the vehicle capacity Q was increased by a factor of 1.5 for the AA and BB groups, and by 1.25 for the CC and DD groups. Table 5.1 summarizes the characteristics of these instances. For each group, we report the vehicle capacity Q, the width of the time windows W, and the delay Δ applied to the time windows of the delivery nodes. For each group, we have tested 10 instances in which the number of requests ranges from 30 to 75.

The second data set contains the 236 original PDPTW instances of Li and Lim (2003). We have solved instances containing between 50 and 300 requests : 56 instances of 50 requests, 60 instances of 100 requests, 60 instances of 200 requests, and 60 instances of 300 requests. For each instance size, six groups were tested (LC1, LC2, LR1, LR2, LRC1, and LRC2).

5.4.2 Computational results for the first set

Table 5.2 presents the results for the first set of instances. The first column indicates the name of the instance corresponding to its group and to its number of requests. The next three columns report information on the best known solution (*Best known*). We report the number of vehicles (*Veh.*) and the distance (*Dist.*). The values in boldface are optimal. For all the instances that were solved to optimality by Cherkesly *et al.* (2014), we also report the time in seconds (*Sec.*) on a computer equipped with an Intel Core i7-3770 (3.4 GHz) taken by their branch-price-and-cut with hybrid-LIFO shortest paths. According to the SPEC's CPU2006 benchmark SPEC (2014), this computer is approximatively 1.25 times faster than ours. For the instances that are not solved to optimality by Cherkesly *et al.* (2014), the best known solution values were found by our heuristic when setting the different parameter values. We then present the results found with our metaheuristic with two different configurations. The first one denoted by *25-25-150* goes through 25 iterations of phase I, 25 iterations of phase II, and 150 iterations of phase III. The second one called *50-50-100* goes through 50 iterations of phase I, 50 iterations of phase II, and 100 iterations of phase III. We have executed 10 runs for each configuration and we report the best found solution out of these. For each best found solution, we present the following information : *Sec.* the total time in seconds to execute 10 runs, *V(%)* the percentage deviation of the number of vehicles on the best known number of vehicles, and *D(%)* the percentage deviation on the total distance. The deviation of the total distance is only reported if the number of vehicles attains its minimum best known value.

For the configuration 25-25-150, the algorithm does not find the minimum best known number of vehicles for only one instance out of 40 (CC65), whereas for the second configuration 50-50-100, the algorithm does not reach the minimum best known number of vehicles for three instances out of 40 (BB50, BB65, CC65). For both configurations, 19 best known solutions out of 40 are found, the average deviation of the total distance are 0.34%, and 0.17%, respectively, for configurations 25-25-150 and 50-50-100, and the number of optimal solutions found are, respectively, 14 and 13. For these instances, performing more iterations of phase III seems to help finding a solution using a lower number of vehicles. One can realize that our algorithm is faster than the branch-price-and-cut proposed for this problem Cherkesly *et al.* (2014), and that there is no significant difference in the computational time between the two configurations.

5.4.3 Computational results for the second set

Table 5.3 presents the summary of the results obtained for the second set of instances. We have tested our algorithm with the same number of iterations as in the previous section, i.e., 25-25-150 and 50-50-100. For each of these configurations, we report *# Best known* the number of instances for which our algorithm reaches the best known solution after 10 runs, *V (%)* the average deviation on the best known number of vehicles for the best found solution after executing 10 runs with our algorithm, *D (%)* the average deviation on the total traveled distance for the best found solution after 10 runs if it reaches the best known number of vehicles, and *Seconds* the total time in seconds to execute 10 runs. Note that the best known solution values were found by our heuristic when setting the different parameter values. For instances with 50 requests, more best known solutions are obtained with the 50-50-100 configuration. With this setting, 24 best known solutions are found with an average deviation of 0.43% with respect to the best known number of vehicles and of 0.67% with respect to the best known total distance. For the 100- and 300-request instances, the 25-25-150 configuration produces better average results than the second configuration. In particular, for instances with 300 requests, nine best known solutions are found with more iterations of phase III, and six best known solutions are found with fewer iterations of phase III. The average deviation with respect to the best known number of vehicles are, respectively, 2.63% and 2.93%, and with respect to the best known total distance are 1.66% and 2.84%, respectively. For 200-request instances, the 25-25-150 configuration finds more best known solutions than the second configuration, but the 50-50-100 configuration has a better average

Table 5.2 – Comparative computational results for the first set of instances. The best known values in boldface are optimal. The reported seconds corresponds to a computer equipped with (1) an Intel Core i7-3770 processor (3.4 Ghz), and (2) an Intel(R) Xeon(R) X5675 processor (3.07 GHz).

Instance	Best known			25-25-150			50-50-100		
	Sec.(1)	Veh.	Dist.	Sec.(2)	*V(%)*	*D(%)*	Sec.(2)	*V(%)*	*D(%)*
AA30	2.5	**3**	**1129.4**	47.2	0.00	0.00	48.6	0.00	0.00
AA35	16.7	**3**	**1294.1**	65.8	0.00	0.00	65.7	0.00	0.00
AA40	7.1	**4**	**1349.2**	85.3	0.00	0.00	81.8	0.00	0.00
AA45	21.6	**4**	**1521.4**	106.5	0.00	0.00	109.6	0.00	0.00
AA50	41.0	**4**	**1643.6**	132.9	0.00	0.00	126.6	0.00	0.00
AA55	59.0	**5**	**1743.2**	156.4	0.00	0.00	160.0	0.00	0.00
AA60	789.2	**5**	**1949.7**	205.7	0.00	0.19	205.2	0.00	0.23
AA65	657.5	**5**	**2077.4**	249.5	0.00	0.15	256.2	0.00	0.03
AA70	1094.5	**5**	**2219.2**	294.8	0.00	0.31	290.8	0.00	0.23
AA75	1893.4	**5**	**2330.1**	378.2	0.00	0.73	353.8	0.00	0.83
BB30	5.4	**3**	**1077.5**	46.4	0.00	0.00	46.3	0.00	0.00
BB35	13.2	**3**	**1312.4**	61.2	0.00	0.00	61.1	0.00	0.00
BB40	130.0	**4**	**1404.0**	80.1	0.00	0.00	82.0	0.00	0.00
BB45	667.7	**4**	**1537.5**	105.2	0.00	0.00	104.1	0.00	0.00
BB50	687.3	**4**	**1791.1**	135.2	0.00	3.90	135.9	25.00	-
BB55		5	1921.3	163.3	0.00	0.00	162.9	0.00	0.29
BB60	468.4	**6**	**2305.5**	201.8	0.00	0.96	202.4	0.00	0.17
BB65	1015.3	**6**	**2564.6**	247.9	0.00	1.36	236.1	16.67	-
BB70		7	2545.1	291.7	0.00	0.12	281.6	0.00	0.22
BB75		7	2683.8	354.9	0.00	0.06	360.5	0.00	0.00
CC30	17.6	**3**	**1088.6**	50.7	0.00	0.00	50.4	0.00	0.00
CC35	39.4	**3**	**1237.4**	67.7	0.00	0.69	67.5	0.00	0.37
CC40	62.2	**3**	**1340.2**	92.1	0.00	0.00	90.1	0.00	0.00
CC45		3	1538.3	112.0	0.00	0.44	113.7	0.00	0.00
CC50	1475.2	**4**	**1673.6**	148.5	0.00	0.56	148.1	0.00	0.29
CC55	2741.1	**4**	**1793.5**	187.5	0.00	0.59	182.8	0.00	0.50
CC60		4	1972.3	247.2	0.00	0.15	228.5	0.00	0.35
CC65		4	2183.1	289.0	25.00	-	286.9	25.00	-
CC70		5	2194.9	343.6	0.00	0.00	352.6	0.00	0.64
CC75		5	2338.1	426.4	0.00	0.62	438.9	0.00	0.00
DD30	3307.2	**2**	**1103.2**	53.8	0.00	0.00	52.7	0.00	0.00
DD35	767.8	**3**	**1127.8**	73.4	0.00	0.66	73.1	0.00	0.09
DD40	3092.7	**3**	**1245.3**	105.7	0.00	0.00	101.1	0.00	0.01
DD45		3	1350.3	135.9	0.00	0.23	134.6	0.00	0.01
DD50		3	1452.7	180.4	0.00	0.09	171.2	0.00	0.00
DD55		3	1655.9	218.2	0.00	0.77	212.5	0.00	0.00
DD60		4	1762.2	269.8	0.00	0.00	259.2	0.00	0.92
DD65		4	2012.6	328.8	0.00	0.00	302.7	0.00	0.45
DD70		4	2125.9	365.7	0.00	0.78	363.5	0.00	0.00
DD75		4	2278.9	455.7	0.00	0.00	439.3	0.00	0.60
Average				189.0		0.34	186.0		0.17

Table 5.3 – Comparative computational results for the second set of instances

Configuration	25-25-150				50-50-100			
n	50	100	200	300	50	100	200	300
# Best known	20	17	15	9	24	19	9	6
V (%)	0.16	0.89	1.50	2.63	0.43	1.15	1.13	2.93
D (%)	0.92	1.30	1.47	1.66	0.67	1.38	1.78	2.84
Seconds	229.1	1,574.2	11,790.5	31,296.3	230.3	1,529.3	11,562.9	30,674.9

deviation on the best known number of vehicles.

We present our results in three tables. Table 5.4 presents the best known results for instances with 50, 100, 200 and 300 requests. The best known solution values are those found by our heuristic when choosing the different parameter settings. For each best known solution, we report the number of vehicles (*Veh.*) and the total traveled distance (*Dist.*). Table 5.5 presents the percentage deviations with respect to the best known solution values when solving each instance 10 times, and setting the number of iterations to 25-25-150 for instances with 50, 100, 200 and 300 requests. Table 5.6 reports the percentage deviations with respect to the best known solution values by setting the number of iterations to 50-50-100 for the same instances. In both Tables 5.5 and 5.6, we report the total time in seconds (*Sec.*) to execute 10 runs, and for each best found solution, *V (%)*, the percentage deviation on the best known number of vehicles, and *D (%)*, the percentage deviation on the best known total traveled distance.

5.4.4 Impact of the local search operators

This section presents the results obtained when omitting one local search operator of the algorithm at a time. We report results on the number of vehicles and the total traveled distance. For the sake of conciseness, we only report these results obtained with 25 iterations of phase I, 25 iterations of phase II and 150 iterations of phase III. The conclusions also hold for the 50-50-100 configuration. Note that our algorithm was executed only once for each instance.

Table 5.7 shows the impact on the number of instances for which the number of vehicles is the best found. Each column corresponds to a set of instances : AA-DD corresponds to the first set of instances, and 50, 100, 200 and 300 correspond to the second set of instances with 50, 100, 200, and 300 requests, respectively. The inter-route relocate operator seems to have

Table 5.4 – Best known solution values *(cont'd)*

		LC1		LC2		LR1		LR2		LRC1		LRC2	
n		Veh.	Dist.	Veh.	Dist.	Veh.	Dist.	Veh.	Dist.	Veh.	Dist.	Veh.	Dist.
50	1	15	1433.3	7	1923.0	30	2271.4	6	1705.0	18	2040.1	5	2214.4
	2	14	1472.0	5	2145.2	21	1804.2	5	1749.5	16	1988.7	5	1923.6
	3	10	1396.3	4	1261.9	14	1491.7	3	1448.7	12	1473.3	4	1535.1
	4	9	1043.5	4	986.5	10	1151.5	3	1314.9	11	1322.8	3	1152.5
	5	13	1668.0	5	1300.4	17	1568.7	4	1414.3	18	1996.7	5	1875.2
	6	13	1518.9	4	1753.1	15	1608.6	4	1459.7	14	1702.2	4	1793.1
	7	12	1618.3	4	1054.7	12	1439.6	3	1387.0	13	1523.2	4	1636.3
	8	11	1165.7	4	1031.6	11	1268.5	3	1170.4	11	1356.6	4	1282.0
	9	10	892.4			13	1523.5	4	1294.4				
	10					11	1255.3	4	1393.4				
	11					13	1352.6	3	1158.1				
	12					11	1196.5						
100	1	33	4943.5	15	6587.5	34	6932.3	9	6762.1	27	4891.7	11	5500.8
	2	26	4425.2	10	4938.5	22	5450.6	8	6272.3	20	4354.1	8	4319.7
	3	20	3769.3	8	3046.0	17	4593.8	5	5392.5	15	4045.0	6	4033.7
	4	17	3071.4	7	2885.3	12	3562.7	4	4000.5	10	3206.6	4	3869.9
	5	27	4600.0	10	4037.2	24	5918.9	7	6431.8	19	4313.1	7	4555.1
	6	26	4042.9	9	4101.6	18	5056.1	6	5862.3	20	4010.8	7	4818.2
	7	24	4140.1	8	3634.3	14	4337.9	4	5535.3	17	3833.1	6	3853.6
	8	22	3381.6	8	3281.6	10	3222.4	3	3481.6	15	3643.7	5	4775.2
	9	21	3619.6	7	3246.9	18	5630.1	6	5701.5	15	3557.8	5	3602.2
	10	19	3286.9	7	2887.2	14	4162.5	5	5327.7	14	3158.1	4	3483.5

Table 5.4 – Best known solution values *(cont'd and end)*

		LC1		LC2		LR1		LR2		LRC1		LRC2	
n		Veh.	Dist.	Veh.	Dist.	Veh.	Dist.	Veh.	Dist.	Veh.	Dist.	Veh.	Dist.
200	1	63	12154.3	25	12288.6	60	15860.1	19	16033.1	49	11937.4	23	12736.5
	2	51	11494.0	20	9511.2	41	12252.6	16	14368.2	43	10350.7	16	10286.0
	3	40	9565.1	15	9408.4	32	10593.2	9	13499.9	32	9546.2	13	10234.7
	4	32	8304.3	13	6876.1	20	7917.7	6	9205.5	21	6918.9	7	7768.9
	5	54	12313.0	20	8933.2	47	14387.4	13	14359.5	41	10724.6	16	11100.3
	6	50	10252.3	16	8161.0	36	12292.0	10	12715.7	39	10051.7	15	10468.0
	7	47	11185.9	16	7564.1	27	9553.7	7	10985.9	35	9471.2	13	10827.0
	8	45	9002.2	15	7174.6	17	7484.0	5	9125.3	32	8714.5	11	9625.4
	9	41	8825.3	15	7115.3	37	12217.3	12	13967.4	32	9135.4	11	9218.0
	10	38	8886.1	14	7289.6	26	9951.8	10	12931.8	29	8266.6	9	8736.0
300	1	107	26612.2	43	22845.4	77	31454.0	31	34962.0	75	24969.4	35	25093.1
	2	80	22574.9	31	19520.7	57	26543.0	22	28869.4	59	20314.5	26	23006.7
	3	60	18116.6	23	15306.9	41	23025.9	15	24527.8	43	17078.2	17	19231.6
	4	52	15562.2	19	12692.8	28	16373.7	8	18254.2	27	12972.9	10	16252.7
	5	86	23864.6	32	17774.8	66	28658.4	22	30608.4	57	20522.2	25	22543.0
	6	77	21174.4	26	16015.0	51	24697.6	17	29783.3	60	21855.1	23	26086.7
	7	76	21607.7	25	14749.0	35	19621.3	12	23103.7	49	19250.3	20	23632.0
	8	68	19164.6	23	14068.4	23	14851.4	7	18050.1	45	17996.3	17	20896.3
	9	63	17579.0	23	15103.3	56	26358.5	19	29407.4	43	18436.6	16	22002.1
	10	59	16754.0	21	12979.6	40	22898.9	16	28221.7	38	16933.2	13	21670.3

Table 5.5 – Computational times for 10 runs and percentage deviations with respect to the best known solution values (25-25-150 iterations)

		LC1			LC2			LR1			LR2			LRC1			LRC2		
n		Sec.	*V(%)*	*D(%)*	Sec.	*V(%)*	*D(%)*	Sec.	*V(%)*	*D(%)*	Sec.	*V(%)*	*D(%)*	Sec.	*V(%)*	*D(%)*	Sec.	*V(%)*	*D(%)*
50	1	143.5	0.00	0.00	138.7	0.00	0.15	106.4	0.00	0.00	159.2	0.00	0.21	111.6	0.00	0.22	160.9	0.00	0.30
	2	196.7	0.00	0.00	222.8	0.00	8.01	156.2	0.00	0.00	240.8	0.00	1.08	161.4	0.00	0.00	220.3	0.00	0.01
	3	222.5	0.00	2.56	335.6	0.00	1.08	181.5	0.00	0.00	302.3	0.00	0.00	178.8	0.00	0.83	281.9	0.00	0.00
	4	270.0	0.00	0.99	543.7	0.00	0.37	210.2	0.00	0.48	473.2	0.00	0.00	217.8	0.00	0.16	329.1	0.00	0.30
	5	168.6	0.00	11.01	186.3	0.00	0.00	121.0	0.00	0.00	229.8	0.00	0.00	147.7	0.00	0.00	202.6	0.00	0.07
	6	152.8	0.00	0.00	209.8	0.00	0.00	149.2	0.00	0.24	284.1	0.00	1.94	140.6	0.00	0.22	209.2	0.00	0.21
	7	154.2	0.00	4.67	248.0	0.00	0.00	186.6	0.00	0.78	336.9	0.00	0.35	177.2	0.00	2.29	262.5	0.00	1.02
	8	168.9	0.00	0.55	325.6	0.00	0.05	189.9	0.00	0.88	557.3	0.00	0.00	199.7	9.09	-	304.2	0.00	4.89
	9	183.7	0.00	0.00				154.7	0.00	0.00	289.6	0.00	1.90						
	10							200.3	0.00	0.01	297.6	0.00	0.84						
	11							219.9	0.00	0.00	377.8	0.00	0.57						
	12							227.8	0.00	1.26									
100	1	848.8	0.00	0.76	805.4	0.00	3.07	708.0	0.00	0.51	900.6	0.00	5.53	661.9	0.00	0.15	842.7	9.09	-
	2	1167.6	0.00	2.94	1464.4	0.00	1.82	1035.3	0.00	2.21	1621.6	0.00	2.13	880.1	0.00	0.09	1879.0	0.00	0.00
	3	1362.0	0.00	1.58	2096.3	0.00	5.41	1313.4	0.00	1.07	2578.4	0.00	6.57	1362.9	6.67	-	2204.1	0.00	0.00
	4	1829.4	0.00	1.19	2765.2	0.00	0.13	2340.2	0.00	1.18	4085.5	0.00	1.38	2482.5	0.00	0.00	4030.0	0.00	1.96
	5	784.3	0.00	1.03	1039.6	0.00	0.00	725.6	0.00	1.45	1156.3	0.00	0.00	909.8	0.00	0.00	1227.5	0.00	0.00
	6	958.3	0.00	2.68	1281.3	0.00	2.65	1420.0	5.56	-	1837.0	0.00	1.11	798.0	0.00	0.00	1192.8	0.00	0.00
	7	997.6	0.00	0.99	1488.4	0.00	0.83	1604.5	0.00	0.00	2855.8	0.00	0.00	1026.7	0.00	0.00	1602.6	0.00	3.30
	8	1041.7	0.00	0.00	1368.5	0.00	0.86	1802.4	0.00	0.00	5697.0	0.00	0.00	1078.1	6.67	-	1789.1	20.00	-
	9	1367.5	0.00	1.37	1703.4	0.00	4.77	823.2	5.56	-	1248.2	0.00	1.08	1164.3	0.00	0.00	1799.9	0.00	0.96
	10	1652.6	0.00	0.91	1714.7	0.00	1.53	1120.3	0.00	3.69	1526.7	0.00	0.20	1438.5	0.00	0.00	1942.8	0.00	1.25
200	1	6362.5	1.59	-	6917.0	4.00	-	4611.0	3.33	-	7194.8	0.00	1.94	4992.9	0.00	0.00	6843.8	0.00	0.00
	2	7458.4	1.96	-	11652.5	0.00	1.99	7495.4	2.44	-	11360.2	6.25	-	6440.2	0.00	0.00	12712.8	0.00	0.00
	3	9536.1	2.50	-	15464.3	0.00	6.03	11045.2	0.00	2.87	21704.8	11.11	-	9124.6	3.13	-	18705.2	7.69	-
	4	16335.2	3.13	-	22552.8	0.00	1.30	17954.6	0.00	0.00	30268.1	0.00	0.00	12602.8	0.00	2.70	30071.5	0.00	6.76
	5	5826.1	5.56	-	8057.9	0.00	0.27	4942.6	4.26	-	8671.4	7.69	-	5463.3	2.44	-	9308.8	0.00	0.73
	6	6193.6	2.00	-	10270.0	0.00	0.00	8689.7	2.78	-	15235.7	0.00	8.87	5665.5	0.00	1.39	9502.1	0.00	1.14
	7	6479.8	2.13	-	12438.5	0.00	1.91	12220.5	3.70	-	24346.1	0.00	3.04	6700.5	2.86	-	11927.8	0.00	0.00
	8	6990.9	0.00	0.00	12002.8	0.00	0.60	19461.1	0.00	0.00	34717.9	0.00	0.00	8095.0	0.00	0.00	13817.5	0.00	0.00
	9	9577.4	0.00	3.93	12158.6	0.00	0.11	5924.3	5.41	-	10791.3	0.00	1.07	7555.3	0.00	1.89	14451.7	0.00	2.75
	10	10786.9	0.00	0.00	13736.3	0.00	0.91	7562.3	3.85	-	12647.2	0.00	1.26	8621.0	0.00	2.47	17186.4	0.00	0.00
300	1	14806.5	0.93	-	17205.7	2.33	-	13880.1	3.90	-	17836.4	9.68	-	13074.0	2.67	-	17141.8	2.86	-
	2	18267.7	5.00	-	27145.4	6.45	-	23470.0	8.77	-	30935.7	9.09	-	21169.5	0.00	0.63	29712.4	0.00	0.74
	3	27710.0	5.00	-	44368.1	4.35	-	37470.5	2.44	-	59519.4	0.00	0.00	33104.6	2.33	-	49810.8	0.00	1.89
	4	40606.5	0.00	0.00	76076.5	0.00	5.33	42660.0	0.00	3.33	82179.2	0.00	0.57	50125.0	0.00	4.68	82496.9	10.00	-
	5	14278.1	2.33	-	21278.5	0.00	3.40	13910.1	6.06	-	21593.3	4.55	-	15550.3	1.75	-	23798.2	4.00	-
	6	15457.5	2.60	-	26413.3	0.00	0.00	22135.4	1.96	-	36682.9	0.00	0.00	15515.0	1.67	-	24677.1	4.35	-
	7	16563.5	2.63	-	29033.2	0.00	0.00	34004.0	2.86	-	61555.1	0.00	0.46	17183.9	4.08	-	31408.8	0.00	0.00
	8	19837.4	1.47	-	30885.4	0.00	2.27	44969.6	4.35	-	86790.1	0.00	1.98	19822.6	0.00	1.96	34196.9	0.00	0.00
	9	25536.5	0.00	0.00	34813.0	0.00	6.22	15073.3	8.93	-	24345.4	10.53	-	21862.2	2.33	-	32035.4	0.00	0.00
	10	32039.4	0.00	3.58	35031.8	0.00	1.04	19368.9	5.00	-	28606.1	0.00	3.47	23146.9	2.63	-	37604.3	7.69	-

Table 5.6 – Computational times for 10 runs and percentage deviations with respect to the best known solution values (50-50-100 iterations)

		LC1			LC2			LR1			LR2			LRC1			LRC2		
n		Sec.	*V(%)*	*D(%)*	Sec.	*V(%)*	*D(%)*	Sec.	*V(%)*	*D(%)*	Sec.	*V(%)*	*D(%)*	Sec.	*V(%)*	*D(%)*	Sec.	*V(%)*	*D(%)*
50	1	174.3	0.00	0.00	139.2	0.00	0.00	104.0	0.00	0.00	156.9	0.00	0.67	113.7	0.00	1.28	163.7	0.00	0.00
	2	213.1	0.00	0.93	223.9	0.00	1.10	159.1	0.00	0.00	232.9	0.00	0.02	168.9	6.25	-	219.7	0.00	1.24
	3	225.3	0.00	1.01	327.6	0.00	1.36	189.9	0.00	0.00	298.0	0.00	0.66	185.4	0.00	0.00	280.2	0.00	0.35
	4	274.1	0.00	0.14	497.4	0.00	0.96	207.8	10.00	-	462.7	0.00	0.13	220.2	0.00	0.00	338.7	0.00	1.56
	5	185.6	7.69	-	189.4	0.00	0.00	121.7	0.00	0.00	231.3	0.00	2.43	148.4	0.00	0.00	200.3	0.00	0.22
	6	152.9	0.00	2.22	213.4	0.00	9.79	153.3	0.00	0.00	290.0	0.00	0.56	140.2	0.00	0.54	215.6	0.00	0.00
	7	160.0	0.00	0.00	251.5	0.00	0.00	203.5	0.00	0.55	352.8	0.00	0.09	180.1	0.00	0.00	260.7	0.00	0.97
	8	178.6	0.00	0.00	335.4	0.00	0.00	195.4	0.00	0.70	506.4	0.00	0.41	199.4	0.00	0.00	313.1	0.00	1.86
	9	184.3	0.00	0.00				149.3	0.00	0.00	286.4	0.00	0.91						
	10							202.5	0.00	2.46	298.0	0.00	0.00						
	11							217.5	0.00	0.00	374.8	0.00	0.41						
	12							231.2	0.00	0.00									
100	1	1121.3	0.00	0.63	763.5	0.00	0.00	693.8	0.00	0.66	859.8	11.11	-	698.4	0.00	0.31	862.1	9.09	-
	2	1117.9	0.00	4.43	1468.1	0.00	4.15	890.0	0.00	0.00	1487.3	0.00	2.93	874.9	0.00	0.00	1712.4	0.00	1.13
	3	1241.5	0.00	0.00	1994.7	0.00	4.39	1243.7	0.00	0.23	2272.3	0.00	0.00	1367.0	6.67	-	2289.2	0.00	2.29
	4	1799.3	0.00	0.00	2581.1	0.00	0.00	2172.8	0.00	0.00	4098.8	0.00	2.21	2395.5	10.00	-	3603.9	0.00	4.66
	5	829.9	0.00	0.00	951.4	0.00	2.96	710.5	0.00	0.00	1078.5	0.00	0.49	913.4	0.00	0.40	1243.4	0.00	0.37
	6	867.0	0.00	0.00	1286.3	0.00	4.23	1289.0	0.00	0.00	1904.5	0.00	0.00	761.6	0.00	0.49	1150.3	0.00	1.06
	7	934.0	0.00	1.41	1343.3	0.00	0.72	1571.0	0.00	5.98	2624.4	0.00	4.67	954.5	0.00	1.81	1500.8	0.00	0.00
	8	969.0	0.00	0.11	1485.9	0.00	0.00	1825.1	0.00	1.38	6108.5	0.00	2.65	946.6	0.00	0.00	1716.1	20.00	-
	9	1300.6	0.00	0.00	1654.3	0.00	7.53	744.6	5.56	-	1342.1	0.00	0.19	1227.5	6.67	-	1657.8	0.00	0.50
	10	1590.3	0.00	0.78	1828.4	0.00	1.26	1056.9	0.00	5.48	1428.8	0.00	0.00	1467.0	0.00	0.85	1888.2	0.00	0.00
200	1	6940.1	0.00	0.11	6194.3	4.00	-	4764.1	3.33	-	6647.6	0.00	1.43	4625.0	2.04	-	6559.2	4.35	-
	2	7516.7	1.96	-	11210.6	0.00	6.59	6961.0	2.44	-	11226.6	6.25	-	6588.8	2.33	-	11642.4	0.00	1.76
	3	9380.3	2.50	-	15364.2	0.00	0.00	10340.4	0.00	2.43	21357.5	0.00	0.00	9091.1	3.13	-	18582.6	7.69	-
	4	15466.9	0.00	1.13	23998.8	0.00	3.63	16346.3	0.00	1.84	32087.1	0.00	0.29	12147.4	0.00	1.26	30699.6	0.00	11.66
	5	6265.3	5.56	-	8195.4	0.00	0.86	4436.3	2.13	-	8902.3	0.00	0.00	4909.6	2.44	-	9449.3	0.00	1.65
	6	6044.3	2.00	-	10112.7	0.00	8.43	7808.2	2.78	-	14902.8	0.00	0.00	5637.1	0.00	0.51	9570.8	0.00	0.00
	7	6571.0	0.00	0.00	11961.9	0.00	1.98	11579.8	3.70	-	24182.8	0.00	0.00	6466.7	2.86	-	11608.0	0.00	0.20
	8	6754.6	0.00	0.50	11340.7	0.00	0.00	18418.9	0.00	0.17	33687.3	0.00	3.83	7109.4	0.00	1.76	13526.4	0.00	2.47
	9	10021.0	0.00	3.57	13216.4	0.00	0.72	5555.9	0.00	0.14	10220.3	0.00	0.02	7744.0	0.00	1.70	14037.3	0.00	3.09
	10	11209.3	2.63	-	12525.3	0.00	3.99	7280.8	3.85	-	11508.1	0.00	0.00	8565.0	0.00	0.71	16712.5	0.00	2.66
300	1	16901.0	0.93	-	17322.9	2.33	-	13917.7	5.19	-	17638.2	9.68	-	13141.5	1.33	-	17645.1	0.00	0.00
	2	18323.6	5.00	-	24839.0	6.45	-	21352.3	10.53	-	30952.4	9.09	-	20314.5	3.39	-	29268.1	0.00	2.02
	3	27351.3	6.67	-	42559.3	4.35	-	37119.4	4.88	-	55816.9	0.00	5.44	31631.4	0.00	1.25	47772.8	0.00	1.56
	4	37855.7	0.00	0.00	72657.4	0.00	9.16	42773.9	0.00	0.00	79270.3	0.00	2.96	49663.2	3.70	-	78732.0	10.00	-
	5	15401.6	3.49	-	20135.0	0.00	9.74	14083.6	4.55	-	21388.0	4.55	-	16027.3	1.75	-	21898.2	4.00	-
	6	15580.2	2.60	-	25646.8	0.00	11.56	20407.1	3.92	-	36152.9	5.88	-	14964.2	1.67	-	24107.2	4.35	-
	7	17024.7	1.32	-	29393.6	0.00	5.48	34024.9	5.71	-	59058.3	0.00	0.00	17855.8	6.12	-	30291.4	5.00	-
	8	20752.5	1.47	-	30002.5	0.00	1.93	46688.3	0.00	1.11	86181.7	0.00	0.00	19130.5	0.00	2.62	33754.6	0.00	0.74
	9	26067.3	0.00	1.26	33567.6	4.35	-	14333.2	8.93	-	23823.7	10.53	-	20700.0	0.00	0.00	34022.2	0.00	4.54
	10	29514.5	1.69	-	35871.2	0.00	5.24	18102.8	2.50	-	28332.3	0.00	0.96	22411.9	0.00	0.64	38978.6	7.69	-

Table 5.7 – Number of instances for which the minimum best known number of vehicles is reached

		Instance set				
		AA-DD	50	100	200	300
All operators		39	55	54	38	25
All except	Inter-route exchange	37	46	39	24	11
	Inter-route relocate	36	46	37	12	3
	Inter-route multiple exchange	37	47	42	21	10
	Inter-route multiple relocate	36	48	41	27	12
	Intra-route exchange	37	49	40	29	9
	Intra-route relocate	36	47	39	19	11
	Intra-route multiple exchange	37	49	39	22	8
	Intra-route multiple relocate	36	48	40	24	11
	Perturbation	36	46	39	36	40

the largest impact on the best known number of vehicles : only three instances with 300 requests have the minimum best known number of vehicles. For these larger instances, the pertubation operator seems to have a negative effect on the best known solutions : eliminating the perturbation operator allows us to find more solutions with the best known number of vehicles.

Table 5.8 presents the impact in percentage on the total traveled distance for the instances where the number of vehicles is the best found. All operators seem to have a positive effect on the total traveled distance. In particular, the inter-route relocate and the inter-route multiple exchange operators seem to be the most important. The average deviation increases up to 23.64% and to 8.95% for instances with 300 requests respectively for the inter-route relocate operator and the inter-route multiple exchange operator.

5.4.5 Impact of the number of iterations

This section presents the impact of the number of iterations on the deviation with respect to the best known solution. Again, for the sake of conciseness, we only present in Figure 5.13 results for the instances with 200 requests and with the number of iterations set to 25-25-150. One can readily observe that in the first iterations, the average deviation is quite high (above 9% in this case), but quite rapidly, after 125 iterations, the objective values fall within 2% of the best known values. Similar results are obtained when looking at the convergence of the algorithm for the other instances and for the different settings for the number of iterations. The algorithm always identifies good solutions within a very limited number of iterations.

Table 5.8 – Average deviations with respect to the best known solution values

		Instance set				
		AA-DD	50	100	200	300
All operators		0.34	0.92	1.30	1.47	1.66
All except	Inter-route exchange	0.82	3.32	5.07	5.67	6.17
	Inter-route relocate	3.16	8.12	15.45	22.86	23.64
	Inter-route multiple exchange	2.20	2.45	6.63	8.42	8.95
	Inter-route multiple relocate	0.93	2.48	5.09	5.33	6.28
	Intra-route exchange	1.07	2.62	4.30	4.07	4.53
	Intra-route relocate	1.03	2.47	5.13	5.68	4.86
	Intra-route multiple exchange	0.90	1.64	4.39	4.49	6.74
	Intra-route multiple relocate	0.90	3.17	4.87	5.58	4.21
	Perturbation	1.08	3.05	3.81	2.58	1.59

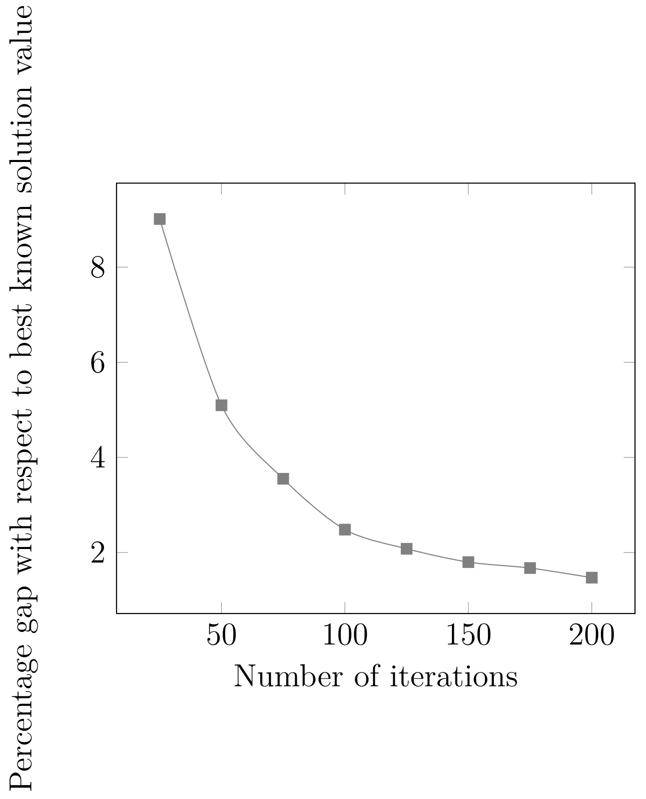


Figure 5.13 – Impact of the number of iterations on the deviation with respect to the best solution value

5.5 Conclusions

We have developed a metaheuristic for the pickup and delivery problem with time windows and LIFO loading combining a genetic algorithm with a local search algorithm. Using a single run of the proposed algorithm, all instances with up to 200 requests were solved within one hour of computation time and all instances with 300 requests were solved within three hours of computation time. Out of the 60 instances with 300 requests, 45 were solved within one hour of computation time, while 54 were solved within two hours. For the instances with known optimal values, the algorithm reaches the mininimum number of vehicles for all instances with configuration 25-25-150. For the instances without known optimal values, we provide for the first time good-quality solutions. For all the instances, our algorithm provides solutions with an average deviation with respect to the best known solution values ranging from 0.17% to 2.84%. The combination of local search and genetic algorithms seems to be very powerful and produces high-quality solutions within modest computing times.

CHAPITRE 6 ARTICLE 3 : BRANCH-PRICE-AND-CUT ALGORITHMS FOR THE PICKUP AND DELIVERY PROBLEM WITH TIME WINDOWS AND MULTIPLE STACKS

M. Cherkesly, G. Desaulniers, S. Irnich et G. Laporte, (2015), Branch-Price-and-Cut Algorithms for the Pickup and Delivery Problem with Time Windows and Multiple Stacks. *European Journal of Operational Research*, soumis le 31 mars 2015.

6.1 Introduction

This paper proposes two branch-price-and-cut algorithms for the pickup and delivery problem with time windows and multiple stacks (PDPTWMS) and analyzes their performance. In the pickup and delivery problem, vehicles based at a depot are used to satisfy a set of requests which consists of transporting goods (or items) from a specific pickup location, where the item is loaded, to a specific delivery location, where the item is unloaded. We consider an unlimited fleet of identical vehicles with multiple homogeneous compartments of limited capacity. Each compartment is rear-loaded and is operated as a last-in-first-out (LIFO) stack, meaning that when an item is picked up, it is positioned on top of a stack. An item can only be delivered if it is on top of its stack and shifting items between stacks is not allowed. To illustrate, let 0 denote the depot, and let i^+ and i^- be the pickup and the delivery nodes associated with request i. Figure 6.1 depicts a route and the load of a two-stack vehicle with respect to the multi-stack policy. Each pickup and delivery location has a specified time window during which the service must start. A vehicle route is feasible if (i) the service at each location starts within the given time windows, (ii) the load in each compartment of the vehicle does not exceed its capacity, (iii) each completed requested is first picked up at its pickup location and then delivered at its corresponding delivery location, and (iv) the loading and unloading of the items respect the LIFO policy for each stack. Two types of costs are considered : a fixed cost for each vehicle used in the solution and a distance-related variable cost. The PDPTWMS consists of determining a set of least-cost feasible routes in which the number of vehicles is first minimized.

The PDPTWMS arises in the transportation of heavy or dangerous material for which unnecessary handling should be avoided. In particular, this problem is encountered in the transportation of cars between car dealers with multi-level vehicles, where each level is operated in a LIFO fashion. This problem also arises in the transportation of livestock from farms to slaughterhouses with multi-compartment vehicles, where each compartment is operated in a

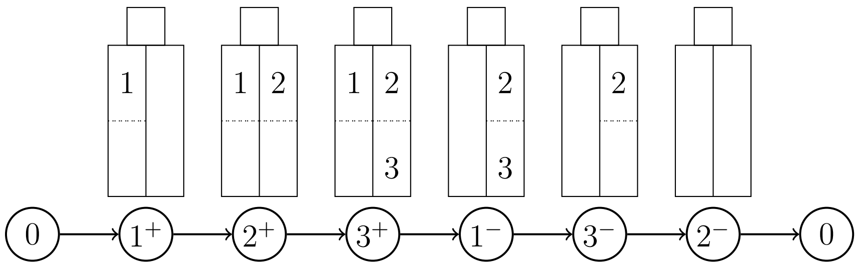


Figure 6.1 – Route satisfying the capacity constraints and the multi-stack policy for a vehicle containing two stacks, each of capacity 2. All three items have a unit demand.

LIFO fashion. To the best of our knowledge, this problem has not been previously studied, but several of its variants have been investigated, namely the pickup and delivery problem (see Berbeglia *et al.* (2007); Parragh *et al.* (2008a,b) for surveys), the pickup and delivery problem with time windows (PDPTW) (see Ropke *et al.* (2007); Battara *et al.* (2014) for exact algorithms), the traveling salesman problem with pickup and delivery and LIFO loading (TSPPDL) (see Carrabs *et al.* (2007a); Cordeau *et al.* (2010) for exact algorithms), the pickup and delivery problem with time windows and LIFO loading (PDPTWL) (see Cherkesly *et al.* (2014) for three exact branch-price-and-cut algorithms), the double traveling salesman problem with multiple stacks (see Alba Martínez *et al.* (2013); Lusby *et al.* (2010); Petersen *et al.* (2010) for exact algorithms and Petersen and Madsen (2009) for a heuristic), the double vehicle routing problem with multiple stacks (see Iori and Riera-Ledesma (2015) for exact algorithms), and the pickup and delivery traveling salesman problem with multiple stacks (PDTSPMS) (see Côté *et al.* (2012a,b) for a branch-and-cut algorithm and a heuristic).

Among the algorithms proposed for the variants of the PDPTWMS, two exact algorithms stand out and constitute the basis of this research. Côté *et al.* (2012a) have developed a branch-and-cut algorithm for the PDTSPMS, for which several families of valid inequalities were proposed and tested, and instances with up to 21 requests were solved to optimality within one hour of computation time. Cherkesly *et al.* (2014) have developed a branch-price-and-cut for the PDPTWL and have introduced three relaxations of the pricing problem, for which they developed shortest path labeling algorithms to enforce the LIFO policy. One algorithm solves the pricing problem without the LIFO policy and imposes it through additional constraints in the master problem. Another algorithm solves the pricing problem with a relaxed LIFO policy and additional constraints may be added to the master problem if needed. The last one solves the pricing problem with a complete version of the LIFO policy. Computational results revealed that for harder instances the hybrid algorithm performs best.

The main objective of this paper is to develop, for the first time, different exact branch-price-and-cut algorithms for the PDPTWMS. A branch-price-and-cut algorithm is a branch-and-price algorithm in which the linear relaxation is strengthened through the generation of valid inequalities. Column generation can be adapted to constrained vehicle routing problems and have been shown to provide some of the best known results for the PDPTW (Ropke and Cordeau (2009); Baldacci *et al.* (2011a)) and for the PDPTWL (Cherkesly *et al.* (2014)).

The first algorithm developed in this paper solves the pricing problem with the multi-stack policy, whereas the second incorporates it partially in the pricing problem and generates additional inequalities to the master problem in which infeasible multi-stack routes are used in a linear relaxation solution. The results of extensive computational experiments on instances derived from known PDTSPMS instances are reported. Instances with up to 75 requests and with up to three stacks can be solved within two hours of computation time. The results show that the first algorithm always performs better than the second.

The remainder of this paper is structured as follows. Section 6.2 proposes a set partitioning formulation for the PDPTWMS and formally introduces the pricing problem. Section 6.3 presents our first branch-price-and-cut algorithm in which the pricing problem is solved under the multi-stack policy. Section 6.4 presents our second branch-price-and-cut algorithm in which the pricing problem is partly solved under the multi-stack policy and through the introduction of additional constraints in the master problem. Computational results are reported in Section 6.5 and are followed by conclusions in Section 6.6.

6.2 A Mathematical Formulation

We now introduce a set partitioning formulation for the PDPTWMS. Beforehand, we provide the required notation.

6.2.1 Notation

Let n denote the number of requests. The PDPTWMS can be defined on a directed graph $G = (N, A)$, where $N = \{0, 1, ..., 2n, 2n + 1\}$ is the set of nodes and A is the set of arcs. Nodes 0 and $2n + 1$ represent two copies of the depot appearing at the start and at the end of a route, respectively. The subsets $P = \{1, ..., n\}$ and $D = \{n + 1, ..., 2n\}$ are the sets of pickup and delivery nodes, respectively. With each request i is associated a pickup node $i \in P$, denoted by i^+, and a delivery node $n + i \in D$, denoted by i^-. Note that $i \in P$ refers to a pickup node and to its associated request.

With each node $i \in N$ is associated a demand q_i to be picked up or delivered, and for each request $i \in P$, $q_i > 0$ and $q_{n+i} = -q_i$. For each request $i \in P$, we refer to the load picked up at node i and delivered at node $n+i$ as an item. Moreover, we assume that $q_0 = q_{2n+1} = 0$. A time window $[\underline{w}_i, \bar{w}_i]$ is associated with each node $i \in N$, where $\underline{w}_i$ and $\bar{w}_i$ represent the earliest and the latest time at which service at node i can start, respectively. An unlimited set of identical vehicles, each with S identical stacks of capacity Q, is available. A non-negative travel cost c_{ij} and a non-negative travel time t_{ij} including the service time at node i are associated with each arc $(i,j) \in A$. The cost of each arc leaving the origin node, i.e., an arc $(0,i) \in A$, $i \in P$, also includes a large vehicle fixed cost, leading to first minimizing the number of vehicles, and then the total traveled distance. The triangle inequality is assumed to be respected for travel costs and travel times.

6.2.2 Set Partitioning Formulation

Let Ω denote the set of all feasible routes with respect to the time window constraints, the capacity constraints, and the multi-stack policy. Let c_r denote the cost of route $r \in \Omega$, i.e., a fixed vehicle cost and its total traveled distance, and let a_{ir} be the number of times node $i \in P$ is visited in route r. Defining y_r as a binary variable equal to 1 if and only if route r is used in the solution, the PDPTWMS can be formulated as

$$\begin{aligned}
\text{minimize} \quad & \sum_{r\in\Omega} c_r y_r & & \quad (6.1)\\
\text{subject to} \quad & \sum_{r\in\Omega} a_{ir} y_r = 1, & \forall i \in P, & \quad (6.2)\\
& y_r \in \{0,1\}, & \forall r \in \Omega. & \quad (6.3)
\end{aligned}$$

The objective function (6.1) minimizes the total cost and constraints (6.2) ensure that each request is completed exactly once. Because the model defined by (6.1)–(6.3) generally contains a large number of variables, column generation is often used to solve its linear relaxation, also called the master problem (see Desaulniers *et al.* (2005)). At each column generation iteration, a restricted master problem (RMP) containing a subset of variables is solved through linear programming, yielding primal and dual solutions. A pricing problem is then solved to identify variables, with associated columns, of negative reduced cost. In the PDPTWMS context, the pricing problem is an elementary shortest path problem with time window constraints, capacity constraints, and multi-stack policy. When such variables are identified, they are added to the RMP and a new iteration starts. Otherwise, the process stops with an optimal solution to the master problem.

6.2.3 Pricing Problem

The column generation pricing problem aims at finding feasible routes with a negative reduced cost. In this section, we provide a formulation for this problem.

The reduced cost of arc $(i,j) \in A$ can be defined as

$$\bar{c}_{ij} = \begin{cases} c_{ij} - \alpha_i, & \forall i \in P, \\ c_{ij}, & \forall i \in N \backslash P, \end{cases} \tag{6.4}$$

where α_i, $i \in P$, are the dual variables associated with constraints (6.2).

For each node $i \in N$, let T_i be a variable representing the time at which the service begins at node i. For each arc $(i,j) \in A$, let x_{ij} be a binary variable equal to 1 if and only if arc $(i,j) \in A$ is used in the current route. For each request $i \in P$ and each stack $s \in S$, let z_i^s be a binary variable equal to 1 if and only if item i is loaded on stack s in the current route, and let Q_i^s be a variable representing the total load of stack s after leaving node i. In order to define the multi-stack policy, we introduce the following notation. Let $R = (i_1, \dots, i_\rho)$ be a path, i.e., an ordered sequence of nodes that respects the capacity constraints, and the time windows but not necessarily the multi-stack policy, such that $i_1 \neq 0$ and $i_\rho \neq 2n+1$ and let $N(R) = \{i_1, \dots, i_\rho\}$ be its corresponding ordered set of nodes. Let $\left(s_i^e\right)_{i \in N(R)}$ be an assignment (indexed by e) to the stacks in S of the items picked up or delivered in R, i.e., $s_i^e \in S$ indicates the stack to which item i is assigned. Some of these assignments might yield an infeasible path with respect to the multi-stack policy. Denote by $I(R)$ the set of infeasible item-to-stack assignments for path R. Let $\mathcal{R}$ be the set of all paths R that can have infeasible assignments, i.e., such that $I(R) \neq \emptyset$. The pricing problem for the PDPTWMS can then be modeled as

$$\text{minimize} \quad \sum_{(i,j) \in A} \bar{c}_{ij} x_{ij} \tag{6.5}$$

$$\text{subject to} \quad \sum_{j \in N | (i,j) \in A} x_{ij} - \sum_{j \in N | (n+i,j) \in A} x_{n+i,j} = 0, \quad \forall i \in P, \tag{6.6}$$

$$\sum_{j \in P} x_{0j} = 1, \tag{6.7}$$

$$\sum_{j \in N | (j,i) \in A} x_{ji} - \sum_{j \in N | (i,j) \in A} x_{ij} = 0, \quad \forall i \in P \cup D, \tag{6.8}$$

$$\sum_{i \in D} x_{i,2n+1} = 1, \tag{6.9}$$

$$\sum_{\mu=1}^{\rho-1} x_{i_\mu,i_{\mu+1}} + \sum_{i\in P\cap N(R)} z_i^{s_i^e} + \sum_{i\in D\cap N(R)} z_{i-n}^{s_{i-n}^e} \leq 2|N(R)|-2,$$

$$\forall R=(i_1,...,i_\rho)\in\mathcal{R}, e\in I(R) \quad (6.10)$$

$$\sum_{j\in N|(i,j)\in A} x_{ij} = \sum_{s\in S} z_i^s, \quad \forall i\in P, \quad (6.11)$$

$$Q_j^s \geq Q_i^s x_{ij} + q_j x_{ij} z_j^s,$$

$$\forall s\in S, (i,j)\in A \text{ such that } j\in P, \quad (6.12)$$

$$Q_j^s \geq Q_i^s x_{ij} + q_j x_{ij} z_{j-n}^s,$$

$$\forall s\in S, (i,j)\in A \text{ such that } j\in D, \quad (6.13)$$

$$\max\{0,q_i\} z_i^s \leq Q_i^s \leq \min\{Q, Q+q_i\} z_i^s, \quad \forall i\in P, s\in S, \quad (6.14)$$

$$T_j \geq (T_i + t_{ij}) x_{ij}, \quad \forall (i,j)\in A, \quad (6.15)$$

$$\underline{w}_i \leq T_i \leq \bar{w}_i, \quad \forall i\in N, \quad (6.16)$$

$$T_i + t_{i,n+i} \leq T_{n+i}, \quad \forall i\in P, \quad (6.17)$$

$$x_{ij}\in\{0,1\}, \quad \forall (i,j)\in A, \quad (6.18)$$

$$z_i^s\in\{0,1\}, \quad \forall i\in P, s\in S. \quad (6.19)$$

The objective function (6.5) minimizes the sum of reduced costs. Constraints (6.6) ensure that the pairing constraints are respected, i.e., the pickup and delivery nodes of a request are visited in the same route. Constraints (6.7)–(6.9) define a path structure for each route. More specifically, constraints (6.7) and (6.9) ensure that each route starts and ends at the depot, while (6.8) are flow conservation constraints. The multi-stack policy is imposed through constraints (6.10) which are stated through infeasible path inequalities. We note that Côté *et al.* (2012a) have proposed an alternate way of formulating these constraints. Constraints (6.11) state that each picked up item must be assigned to exactly one of the stacks. Constraints (6.12) and (6.13) compute the load variables according to the arcs used in the solution and constraints (6.14) ensure that the capacity of each stack is respected. Constraints (6.15) and (6.16) compute the time variables and ensure that the time windows are respected. Constraints (6.17) impose the precedence constraints, i.e., for each request i the pickup node must be visited before the delivery node. The model is non-linear because of constraints (6.12)–(6.15), but can be linearized (see Ropke *et al.* (2007); Côté *et al.* (2012a)).

Note that constraints (6.10) can be replaced by the smaller set of constraints

$$\sum_{\mu=1}^{\rho-1} x_{i_\mu,i_{\mu+1}} \leq |N(R)|-2, \qquad \forall R=(i_1,...,i_\rho)\in\mathcal{R}^*, \quad (6.20)$$

where $\mathcal{R}^* \subseteq \mathcal{R}$ is a subset of infeasible paths such that, for each path, there exists no feasible assignment of the items to the stacks. Solving model (6.5)–(6.9), (6.11)–(6.19), (6.20) could yield an infeasible solution given the values taken by the z_i^s variables at optimality, i.e., these values would provide an infeasible item-to-stack assignment for at least one path in $\mathcal{R} \setminus \mathcal{R}^*$, but there exists an alternative feasible solution having the same cost with different z_i^s values.

6.3 A Branch-Price-and-Cut Algorithm with Multi-Stack Feasible Paths

Our first branch-price-and-cut algorithm fully enforces the multi-stack policy in the pricing problem. In this section, we first present path relaxations and labeling algorithms for the corresponding pricing problem. We then discuss valid inequalities for the PDPTWMS and branching strategies.

6.3.1 Path Relaxations and Labeling Algorithms

The pricing problem is an elementary shortest path problem with pickups and deliveries, time windows, capacity constraints, and multi-stack policy. It can be solved through a labeling algorithm. A label stores information about a partial path starting at the origin node and ending at some node η. Each element stored in a label is called a component. Starting from an initial label E_0 at the origin node 0, a labeling algorithm propagates labels toward the destination node with resource extension functions. To avoid enumerating all feasible paths, some labels are eliminated through a dominance criterion.

The pricing problem can be relaxed by allowing cycles in paths, that is, a request can be completed more than once. These relaxations usually yield weaker master problem lower bounds. Paths with cycles cannot be part of a feasible integer solution, hence branching ensures that the final solution contains only elementary paths.

Sections 6.3.1 and 6.3.1 describe labeling algorithms for the elementary and non-elementary versions of the pricing problem, respectively.

Elementary Shortest Path Problem with Pickups and Deliveries, Time Windows, Capacity Constraints and Multi-Stack Policy

The first version of the constrained shortest path problem respects the elementarity constraints. The ideas presented in this section are non-trivial extensions of those initially proposed by Cherkesly *et al.* (2014) for the PDPTWL. For a given label E, the following components are stored :

— $\eta(E)$, the end node of the partial path ;
— $t(E)$, the start of service time at node η ;
— $c(E)$, the cumulated reduced cost ;
— $U(E)$, the set of unreachable requests ;
— $l_i(E), \forall i \in P$, the load in the stack under item $i \in P$ (including its own load) ;
— $\mathcal{S}_{ij}(E), \forall i, j \in P$, a binary component representing the relative positions of items i and j in a given stack ;
— $\mathcal{C}_{ij}(E), \forall i, j \in P$, a binary component representing the concurrent presence of items i and j in different stacks.

A request $i \in P$ is said to be unreachable if i has already been visited on the partial path, or if traveling directly from η to i exceeds the upper limit of the time window at node $i \in P$. For a given label E, let $R(E) = (0, i_1, i_2, ..., i_\rho = \eta(E))$ be the partial path represented by this label. Then

$$U(E) = \{i \in P | i \in R(E)\} \cup \{i \in P | t(E) + t_{\eta(E),i} > \bar{w}_i\}. \tag{6.21}$$

The relative position of two items $i, j \in P$ in a given stack indicates that the two items are simultaneously in the same stack and item i is on top of item j, that is

$$\mathcal{S}_{ij}(E) = \begin{cases} 1 & \text{if } i = j \text{ and item } i \text{ is in the vehicle,} \\ 1 & \text{if item } i \text{ is in the same stack as item } j \text{ and on top of it} \\ 0 & \text{otherwise.} \end{cases} \tag{6.22}$$

Moreover, for any two items $i, j \in P$, we need to know whether both are simultaneously onboard and in different stacks, or not, that is

$$\mathcal{C}_{ij}(E) = \begin{cases} 1 & \text{if items } i \text{ and } j \text{ are simultaneously in the vehicle} \\ & \qquad \text{but not in the same stack,} \\ 0 & \text{otherwise.} \end{cases} \tag{6.23}$$

This new notation is as powerful as the notation proposed by Cherkesly *et al.* (2014) for the single-stack case, but is better suited for the multi-stack variant because it eliminates the symmetry between the S identical stacks.

Given a label E, its extension along an arc $(\eta(E), j) \in A$ is allowed if one of the following three conditions holds :

$$j \in P \quad \text{and} \quad j \notin U(E), \tag{6.24}$$

$$j \in D \quad \text{and} \quad \mathcal{S}_{j-n,j-n}(E) = 1 \quad \text{and} \quad \mathcal{S}_{i,j-n}(E) = 0, \forall i \in P \backslash \{j-n\}, \tag{6.25}$$

$$j = 2n+1 \quad \text{and} \quad \mathcal{S}_{ii}(E) = 0, \forall i \in P. \tag{6.26}$$

Condition (6.24) ensures that if j is a pickup node, then it must not have been previously visited, and must be reachable with respect to its time window. Condition (6.25) ensures that if j is a delivery node, then its corresponding item must be on top of one of the stacks, i.e., the item is in the vehicle and there is no item on top. Condition (6.26) ensures that if j is the destination node, then all the picked items on the path must have been delivered. Together these three conditions ensure that each request is completed at most once for any complete path from 0 to $2n+1$.

If the extension to a pickup node is allowed, i.e., condition (6.24) is fulfilled, then several new labels may result. Indeed, one new label per stack in use can be created. Thus, for a given label E, we define

$$\mathcal{H}(E) = \{i \in P | \mathcal{S}_{ii}(E) = 1 \text{ and } \mathcal{S}_{ji}(E) = 0, \forall j \in P \backslash \{i\}\} \tag{6.27}$$

as the top items in the stacks. In particular, $|\mathcal{H}(E)|$ is the number of stacks currently in use. If this number is less than the number of stacks, i.e., $|\mathcal{H}(E)| < S$, then there exists at least one empty stack. In such a case, in order to allow the addition of an item on top of an empty stack, an additional auxiliary top item $h = 0$ is created. We define

$$\mathcal{H}_0(E) = \begin{cases} \mathcal{H}(E) \cup \{0\} & \text{if } |\mathcal{H}(E)| < S, \\ \mathcal{H}(E) & \text{otherwise.} \end{cases} \tag{6.28}$$

In summary, for the extension to a pickup node $j \in P$, one new label E^h for each $h \in \mathcal{H}_0(E)$ is created. If j is a delivery node, $j \in D$, a single new label E^h for $h = j - n$ is created. If j is a pickup node and all stacks are empty, i.e., $\mathcal{H}(E) = \emptyset$, a single new label E^h is created for the auxiliary top item $h = 0$. Thus, given a label E, an arc $(\eta(E), j)$, and a top item $h \in \mathcal{H}_0(E)$ for $j \in P$, or $h = j - n$ for $j \in D$, the extension is computed as follows :

$$\eta(E^h) = j, \tag{6.29}$$

$$t(E^h) = \max\{t(E) + t_{\eta(E),j}, \underline{w}_j\}, \tag{6.30}$$

$$c(E^h) = c(E) + \bar{c}_{\eta(E),j}, \tag{6.31}$$

$$U(E^h) = \begin{cases} U(E) \cup \{j\} \cup \{i \in P | t(E^h) + t_{\eta(E^h),i} > \bar{w}_i\} & \text{if } j \in P, \\ U(E) \cup \{i \in P | t(E^h) + t_{\eta(E^h),i} > \bar{w}_i\} & \text{if } j \in D, \end{cases} \tag{6.32}$$

$$l_m(E^h) = \begin{cases} l_h(E) + q_m & \text{if } j \in P,\ j = m, \\ 0 & \text{if } j \in D,\ m = j - n, \\ l_m(E) & \text{otherwise}, \end{cases} \qquad \forall m \in P, \tag{6.33}$$

$$\mathcal{S}_{mi}(E^h) = \begin{cases} 1 & \text{if } j \in P,\ j = m,\ \mathcal{S}_{hi}(E) = 1, \\ 1 & \text{if } j \in P,\ m = i = j, \\ 0 & \text{if } j \in D,\ j - n = m, \\ \mathcal{S}_{mi}(E) & \text{otherwise}, \end{cases} \qquad \forall i, m \in P, \tag{6.34}$$

$$\mathcal{C}_{mi}(E^h) = \begin{cases} 1 & \text{if } j \in P,\ j = m,\ \mathcal{C}_{hi}(E) = 1, \\ 1 & \text{if } j \in P,\ i = j,\ \mathcal{C}_{hm}(E) = 1, \\ 0 & \text{if } j \in D,\ j - n = m \text{ or } j - n = i, \\ \mathcal{C}_{mi}(E) & \text{otherwise}, \end{cases} \qquad \forall i, m \in P. \tag{6.35}$$

Equations (6.33) state that if j is a pickup node, then the load under it must be the total load in the chosen stack, plus its own load ; if j is a delivery node, then the load under item $j - n$ is 0 because item $j - n$ is no longer in the vehicle, and otherwise the load under each item remains the same. Equations (6.34) update the positions of items that are in the same stack. If j is a pickup node, then it must be on top of all nodes below h, on top of node h, and must be in the vehicle. If j is a delivery node, then there are no more items below $j - n$. The other positions are unchanged. Equations (6.35) update the information about items that are simultaneously in the vehicle but are not in the same stack. If j is a pickup node, then it must be separated from all nodes that are not in the same stack as node h, and if j is a delivery node then there are no more items simultaneously onboard with $j - n$ and in different stacks. The other positions remain the same.

A new label E^h is kept if it respects the time windows and the capacity constraints, that is, if

$$t(E^h) \leq \bar{w}_j, \tag{6.36}$$

$$l_h(E^h) \leq Q. \tag{6.37}$$

Finally, a label E_1 dominates a label E_2 if

$$\eta(E_1) = \eta(E_2), \tag{6.38}$$

$$t(E_1) \leq t(E_2), \tag{6.39}$$

$$c(E_1) \leq c(E_2), \tag{6.40}$$

$$U(E_1) \subseteq U(E_2), \tag{6.41}$$

$$\mathcal{S}_{ij}(E_1) \leq \mathcal{S}_{ij}(E_2), \qquad \forall i, j \in P, \tag{6.42}$$

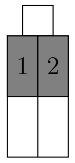
$$\mathcal{C}_{ij}(E_1) \leq \mathcal{C}_{ij}(E_2), \qquad \forall i, j \in P. \tag{6.43}$$

Conditions (6.38)–(6.42) constitute a valid dominance criterion for the single-stack case, i.e., the PDPTWL (see Cherkesly *et al.* (2014)), if the delivery triangle inequality holds. Note that the definition of $\bar{c}_{ij}$ in formula (6.4) does not necessarily ensure the delivery triangle inequality $\bar{c}_{ij} + \bar{c}_{jk} \geq \bar{c}_{ik}$ for all delivery nodes $j \in D$. In this situation, Ropke and Cordeau (2009) propose a procedure to transform an arbitrary cost matrix into a cost matrix that satisfies the delivery triangle inequality. We apply the same procedure before solving the pricing problem.

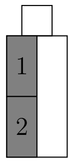
However, the single-stack dominance criterion is not valid for the multi-stack variant. In fact, without conditions (6.43), the dominance criterion would not be valid because items in the same stack for label E_2 could be in different stacks for label E_1. In such a case, the possible extensions of label E_2 could be infeasible for label E_1 with respect to capacity constraints, yielding wrongly dominated labels. Figure 6.2 depicts such a case where $Q = 2$, $q_1 = 1$, $q_2 = 1$, and $q_3 = 2$. Figure 6.2a and 6.2b illustrate the configuration of the vehicle for labels E_1 and E_2, respectively. One can see that item 3 cannot be loaded with the first configuration, but can be with the second one. In that case, conditions (6.38)–(6.42) are respected. Conditions (6.43) are then necessary to allow a proper dominance criterion.

Proposition 6.3.1. *Conditions* (6.38)*–*(6.43) *constitute a valid dominance criterion whenever* $\bar{c}_{ij}$ *satisfies the delivery triangle inequality.*

Proof. The proof is similar to that of Proposition 3 in Cherkesly *et al.* (2014). We show that for every feasible completion of E_2 there exists a feasible completion of E_1 with no greater reduced cost. Let r be a path extending $R(E_2)$ to node $2n+1$ such that $(R(E_2), r)$ is feasible with respect to time windows, elementarity constraints, pickup and delivery constraints, capacity constraints and the multi-stack policy. If no such path exists, then clearly one can remove label E_2. Let r' be the path obtained from r by removing the deliveries for each



(a) Stack configuration for label E_1 : $\mathcal{S}_{11}(E_1) = \mathcal{S}_{22}(E_1) = 1$ and $\mathcal{C}_{12}(E_1) = \mathcal{C}_{21}(E_1) = 1$



(b) Stack configuration for label E_2 : $\mathcal{S}_{11}(E_2) = \mathcal{S}_{22}(E_2) = \mathcal{S}_{21}(E_2) = 1$ and $\mathcal{C}_{12}(E_2) = \mathcal{C}_{21}(E_2) = 0$

Figure 6.2 – Example where two labels cannot be compared, $q_1 = q_2 = 1$, $q_3 = 2$ and $Q = 2$

request $i \in P$ with $\mathcal{S}_{ii}(E_1) = 0$ and $\mathcal{S}_{ii}(E_2) = 1$. Because $(R(E_2), r)$ is feasible with respect to elementarity constraints, and pickup and delivery constraints, then so is $(R(E_1), r')$. Because the triangle inequality is assumed for travel times and $(R(E_2), r)$ is feasible with respect to the time windows, then so is $(R(E_1), r')$. The capacity constraints for each stack are not violated because items in different stacks for E_1 are also in different stacks for E_2, and items in the same stack for E_1 are also in the same stack for E_2, thus the capacity constraints are respected on $(R(E_1), r')$. The multi-stack policy is not violated because the order in which the deliveries are performed on $(R(E_1), r')$ is the same as on $(R(E_2), r)$. Because $(R(E_2), r)$ is feasible, then so is $(R(E_1), r')$. Because the delivery triangle inequality holds for the reduced cost component c, the cost of r' does not exceed that of r. Thus, $c(E_1) \leq c(E_2)$ implies that the cost of $(R(E_1), r')$ is at most equal to that of $(R(E_2), r)$. Hence, label E_1 dominates label E_2. $\square$

Shortest Path Problem with Pickups and Deliveries, Time Windows, Capacity Constraints and Multi-Stack Policy

The second version of the constrained shortest path problem allows paths to contain cycles under the following two conditions :

1. a pickup cannot be performed again before its corresponding delivery has been completed ;
2. the precedence constraints for every request must be respected.

In this version of the algorithm, a label E stores the components $\eta(E), t(E), c(E), l_i(E), i \in P$, $\mathcal{S}_{ij}(E)$, and $\mathcal{C}_{ij}(E),\ i, j \in P$. The extension of a label E along arc $(\eta(E), j)$ is allowed if E and j satisfy condition (6.25), (6.26), or

$$j \in P \quad \text{and} \quad \mathcal{S}_{jj}(E) = 0. \tag{6.44}$$

A label E^h is then created for each top item $h \in \mathcal{H}_0(E)$ using the extension function (6.29)–(6.31) and (6.33)–(6.35). Condition (6.44) relaxes condition (6.24) by allowing cycles to occur while forbidding to pick up the same request twice without delivering it in the meantime. The resulting label E^h is kept if it satisfies the time windows (6.36) and the capacity constraints (6.37) at node j. If the delivery triangle inequality holds, then the following dominance criterion is valid : a label E_1 dominates a label E_2 if conditions (6.38)–(6.40), and (6.42)–(6.43) are respected.

The reader can easily adapt the arguments of Proposition 6.3.1 for this version of the shortest path problem.

6.3.2 Valid Inequalities

We now present valid inequalities commonly used to solve the PDPTW and applicable to the PDPTWMS. These are 2-path cut inequalities, rounded capacity inequalities, and subset-row inequalities. We also present a family of cuts based on the branching on the number of vehicles. These inequalities are added within the master problem. For the sake of conciseness, we omit the discussion about the impact on the reduced cost of adding such inequalities (see Desaulniers *et al.* (2011)). These inequalities are used for both branch-price-and-cut algorithms.

If the number of vehicles is fractional, two branches are created : $\sum_{r\in\Omega} y_r \leq \lfloor \sum_{r\in\Omega} \tilde{y}_r \rfloor$ and $\sum_{r\in\Omega} y_r \geq \lceil \sum_{r\in\Omega} \tilde{y}_r \rceil$, where $(\tilde{y}_1, \ldots, \tilde{y}_{|\Omega|})$ is the computed fractional solution of the master problem. Because we first minimize the number of vehicles, the number of vehicles used in the solution of the master problem is a lower bound on the number of vehicles used in the optimal integer solution. In this case, the inequality

$$\sum_{r\in\Omega} y_r \geq \left\lceil \sum_{r\in\Omega} \tilde{y}_r \right\rceil \tag{6.45}$$

is added to the master problem and replaces the branching on the number of vehicles.

Kohl *et al.* (1999) have introduced 2-path cuts in the context of the vehicle routing problem with time windows (VRPTW). These were later shown to be valid for the PDPTW (see Ropke and Cordeau (2009)). Let $N_S \subseteq P \cup D$ be a subset of nodes that cannot be served by a single vehicle and let $\delta(N_S) = \{(i,j) \in A | i \in N_S, j \in N \backslash N_S\}$ represent the set of arcs exiting set N_S. Then the inequality

$$\sum_{r\in\Omega} \sum_{(i,j)\in\delta(N_S)} b_{ij}^r y_r \geq 2 \tag{6.46}$$

is valid, where b_{ij}^r is a constant equal to the number of times arc $(i,j) \in A$ is used in route r. Identifying a subset of nodes that cannot be served by a single vehicle means determining whether the corresponding pickup and delivery traveling salesman problem with time windows is feasible. Because this is an NP-complete problem (see Savelsbergh (1985)), the separation of violated 2-path cuts is achieved by means of a greedy heuristic (see Ropke and Cordeau (2009)).

The rounded capacity inequalities are often used for the vehicle routing problem (VRP), the VRPTW, and the PDPTW (see, e.g., Naddef and Rinaldi (2002); Cordeau (2006); Ropke *et al.* (2007)) and have been adapted to the PDTSPMS (see Côté *et al.* (2012a)). Let $N_S \subseteq P \cup D$ be a subset of nodes and let $\xi(N_S)$ be a lower bound on the number of vehicles needed to visit all nodes in N_S. Then, the inequality

$$\sum_{r\in\Omega} \sum_{(i,j)\in\delta(N_S)} b_{ij}^r y_r \geq \xi(N_S) \tag{6.47}$$

is valid for $\xi(N_S) = \max\left\{1, \left\lceil \frac{q(\pi(N_S))}{SQ} \right\rceil, \left\lceil \frac{-q(\sigma(N_S))}{SQ} \right\rceil \right\}$, where SQ is the total capacity of the vehicle, $\pi(N_S) = \{i \in P | i \notin N_S, n+i \in N_S\}$ denotes the set of predecessors of N_Ss and $\sigma(N_S) = \{n+i \in D | i \in N_S, n+i \notin N_S\}$ denotes the set of successors of N_S. The lower bound on the load of the vehicles entering N_S is $q(\pi(N_S)) = \sum_{i\in\pi(N_S)} q_i$, and the lower bound on the load of the vehicles leaving N_S is $q(\sigma(N_S)) = \sum_{n+i\in\sigma(N_S)} q_i$. These inequalities are separated by means of a heuristic enumerative procedure (see Ropke *et al.* (2007)).

The subset-row inequalities were introduced by Jepsen *et al.* (2008) for the VRPTW and are a special case of the clique inequalities. These inequalities are the rank-1 Chvátal-Gomory inequalities defined as

$$\sum_{r\in\Omega} \left\lfloor \frac{1}{\chi} \sum_{i\in N_S} a_{ir} \right\rfloor y_r = \left\lfloor \frac{|N_S|}{\chi} \right\rfloor, \quad \forall N_S \subseteq P, 2 \leq \chi \leq |N_S|, \tag{6.48}$$

where N_S is a subset of pickup nodes. As in Jepsen *et al.* (2008) and Desaulniers *et al.* (2008), we focus on the inequalities defined for subsets of three customers because these can be efficiently separated. These subset-row inequalities can be rewritten as

$$\sum_{r\in\Omega_S} y_r \leq 1, \quad \forall N_S \subseteq P \text{ such that } |N_S| = 3, \tag{6.49}$$

where $\Omega_S \subseteq \Omega$ is the subset of routes completing at least two requests in N_S. Because handling the dual prices of the active subset-row inequalities in the subproblem can be highly time-consuming, we limit their usage by generating them only in the first two levels of the branching tree and adding at most 50 cuts at once.

6.3.3 Branching

In a branch-price-and-cut algorithm, branching is used to obtain integer feasible solutions and should be compatible with the column generation process, especially with the algorithm used to solve the pricing problem. With the dominance criterion (6.38)–(6.43), the removal of arcs must preserve the delivery triangle inequality (see Ropke and Cordeau (2008)). Consequently, we propose to branch on the outflow of node subsets (see Naddef and Rinaldi (2002)). This branching strategy adds constraints to the master problem, yielding additional dual prices to be incorporated into the objective function of the pricing problem (see Desaulniers *et al.* (2011)). In this branching strategy, a subset of nodes N_S is selected such that $f(N_S) = \sum_{r\in\Omega}\sum_{(i,j)\in\delta(N_S)} b_{ij}^r \tilde{y}_r$ is as far as possible from the nearest integer, where $f(N_S)$ is the total outflow for the set N_S of the computed fractional solution of the master problem. Two branches are then created by adding the following constraints to the master problem associated with each branch :

$$\sum_{r\in\Omega}\sum_{(i,j)\in\delta(N_S)} b_{ij}^r y_r \leq \lfloor f(N_S) \rfloor\,, \tag{6.50}$$

$$\sum_{r\in\Omega}\sum_{(i,j)\in\delta(N_S)} b_{ij}^r y_r \geq \lceil f(N_S) \rceil\,. \tag{6.51}$$

The exploration of the enumeration tree is achieved through a best-first strategy.

6.4 A Branch-Price-and-Cut Algorithm with Relaxed Multi-Stack Paths

This second branch-price-and-cut algorithm deals with the multi-stack policy partly in the set partitioning formulation and partly in the pricing problem. This pricing problem is easier to solve, but the extended set partitioning formulation is weaker yielding worse lower bounds. As Cherkesly *et al.* (2014) did in their hybrid branch-price-and-cut algorithm for the PDPTWL, we solve the shortest path problem under relaxed multi-stack constraints, i.e., the LIFO policy must be respected for the last κ items of each compartment. An ejection process is therefore needed : when a pickup node is visited, its corresponding item is put on top of a stack ; if the height of the stack exceeds κ, the lowest item is ejected from the stack but is kept in the corresponding compartment. Thus, a compartment can contain stacked items, for which the extension of the partial path needs to respect the LIFO policy, and ejected items, for which the extension of the partial path does not need to respect the LIFO policy. There is no given ordering for the delivery of the ejected items, but these can only be delivered if there are no more stacked items in the compartment. Corresponding infeasible path inequalities are added to the master problem when infeasible multi-stack routes are used in a linear relaxation solution. Figure 6.3 presents an example, for $\kappa = 1$, in which the path must respect the LIFO policy for the items in grey. The vehicle contains two compartments and each compartment has a corresponding stack with a maximal size of one item. Note that item 2 is ejected from the second stack in Figure 6.3c and item 3 is ejected from the second stack in Figure 6.3d because the maximal size is reached.

6.4.1 Labeling Algorithm

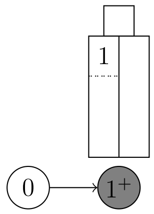
We now describe the modifications to the labeling algorithm presented in Section 6.3 that we have implemented to handle this variant. The valid inequalities and the branching decisions used are those of Sections 6.3.2 and 6.3.3.

In the elementary version of the problem, a label E stores the components $\eta(E)$, $t(E)$, $c(E)$, $U(E)$, $l_i(E)$, $i \in P$, $\mathcal{C}_{ij}(E)$, and $\mathcal{S}^{EP}_{ij}(E)$, $i, j \in P$. $\mathcal{S}^{EP}_{ij}(E)$ is a relaxation of $\mathcal{S}_{ij}(E)$ that considers an ejection process. For a given label E, $\mathcal{S}^{EP}_{ij}(E)$ is defined as

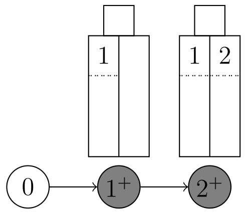
$$\mathcal{S}^{EP}_{ij}(E) = \begin{cases} 1 & \text{if } i = j \text{ and item } i \text{ is in the vehicle,} \\ 1 & \text{if items } i \text{ and } j \text{ are in the same stack and } i \text{ is on top of } j, \\ 1 & \text{if items } i \text{ and } j \text{ are in the same compartment,} \\ & \qquad i \text{ is in the stack, and } j \text{ is ejected,} \\ 0 & \text{if items } i \text{ and } j \text{ are in the same compartment} \\ & \qquad \text{and both are ejected,} \\ 0 & \text{otherwise.} \end{cases} \tag{6.52}$$

That is, if two items $i \in P$ and $j \in P$, $i \neq j$, are in the same vehicle compartment, request i is in the stack and node i^+ was visited after node j^+, then $\mathcal{S}^{EP}_{ij}(E) = 1$ and $\mathcal{S}^{EP}_{ji}(E) = 0$. If two items $i \in P$ and $j \in P$, $i \neq j$, are in the same vehicle compartment but none of them are in the stack, i.e., if both have been ejected, then $\mathcal{S}^{EP}_{ij}(E) = \mathcal{S}^{EP}_{ji}(E) = 0$.

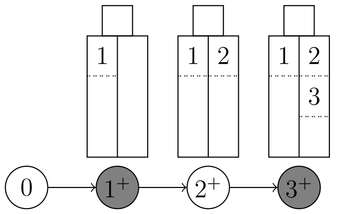
The extension of a label E along an arc $(\eta(E), j) \in A$ is allowed if it satisfies one of the three conditions, (6.24)–(6.26), where $\mathcal{S}_{ij}(E)$ is replaced with $\mathcal{S}^{EP}_{ij}(E)$, $\forall i, j \in P$.



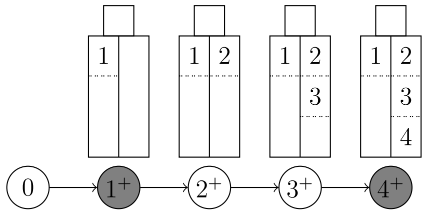
(a) Label E^a : the first stack contains item 1, thus the extension of the path must respect the LIFO policy for item 1, $\mathcal{S}^{EP}_{11}(E^a) = 1$



(b) Label E^b : the first stack contains item 1 and the second stack contains item 2, thus the extension of the path must respect the LIFO policy for items 1 and 2, $\mathcal{S}^{EP}_{11}(E^b) = \mathcal{S}^{EP}_{22}(E^b) = 1$



(c) Label E^c : the first stack contains item 1, the second stack contains item 3, and the second compartment contains items 2 and 3, thus the extension of the path does not need to respect the LIFO policy for item 2 as the maximal size of the LIFO stack has been reached for the second stack, $\mathcal{S}^{EP}_{11}(E^c) = \mathcal{S}^{EP}_{22}(E^c) = \mathcal{S}^{EP}_{33}(E^c) = \mathcal{S}^{EP}_{32}(E^c) = 1$



(d) Label E^d : the first stack contains item 1, the second stack contains item 4, and the second compartment contains items 2, 3, and 4, thus the extension of the path does not need to respect the LIFO policy for items 2 and 3, $\mathcal{S}^{EP}_{11}(E^d) = \mathcal{S}^{EP}_{22}(E^d) = \mathcal{S}^{EP}_{33}(E^d) = \mathcal{S}^{EP}_{44}(E^d) = \mathcal{S}^{EP}_{42}(E^d) = \mathcal{S}^{EP}_{43}(E^d) = 1$ but $\mathcal{S}^{EP}_{23}(E^d) = \mathcal{S}^{EP}_{32}(E^d) = 0$

Figure 6.3 – The extension of the path must respect the LIFO policy for the items in grey ($\kappa = 1$)

Defining $\mathcal{H}(E)$ as in equation (6.27), $\mathcal{H}(E)$ contains all items that can be delivered, i.e., those that have not been ejected from a stack and are on top of a stack, and those that have been ejected from a stack and are not under an item for which the multi-stack policy needs to be respected. In order to have at most one extension per compartment, the top items are defined as the items for which one of these two conditions is respected :

1. the item is in the stack and no item is on top of it ;
2. the item is not in the stack, and is in a compartment that has no item in its stack. One arbitrary item is kept to represent each non-empty compartment. In the following, we choose the item with the smallest index.

We define $\mathbb{C}(E)$ as the set of top items for the relaxed multi-stack policy which can be computed as

$$\mathbb{C}(E) = \{i \in \mathcal{H}(E) | \mathcal{C}_{ij}(E) = 1, j < i, j \in \mathcal{H}(E)\}. \tag{6.53}$$

Note that if $|\mathbb{C}(E)| < S$, an additional auxiliary top item is added in order to allow loading on top of empty stacks. Thus, we define

$$\mathbb{C}(E)_0 = \begin{cases} \mathbb{C}(E) \cup \{0\} & \text{if } |\mathbb{C}(E)| < S, \\ \mathbb{C}(E) & \text{otherwise.} \end{cases} \tag{6.54}$$

The extension of a label E along an arc $(\eta(E), j)$ will create a new label E^h, $\forall h \in \mathbb{C}(E)$. Note that in some cases only one label is created (as explained in Section 6.3.1). For each label E^h, $\mathcal{O}(E^h)$ is defined as the open requests in the vehicle that have been ejected from the stack, i.e., requests that are currently onboard but for which the extension of the label does not need to respect the multi-stack policy. $\mathcal{O}(E^h)$ is computed as

$$\begin{aligned} \mathcal{O}(E^h) = \Bigg\{ i \in P \,\Bigg|\, \mathcal{C}_{hi}(E) = 0 \text{ and } \Bigg(& \sum_{j \in P} \mathcal{S}^{EP}_{ji}(E) \geq \kappa \text{ or} \\ & \exists j \in P \backslash \{i\} \text{ such that } \mathcal{S}^{EP}_{jj}(E) = 1, \mathcal{C}_{ij}(E) = 0, \\ & \mathcal{S}^{EP}_{ij}(E) = \mathcal{S}^{EP}_{ji}(E) = 0 \Bigg) \Bigg\}. \end{aligned} \tag{6.55}$$

Equation (6.55) states that $i \in P$ is an open request with respect to top item $h \in \mathbb{C}(E)_0$ if i and h are in the same vehicle compartment, and if there are at least κ items between the positions of i and h or if i has already been ejected from the stack.

The components of label E^h are set with equations (6.29)–(6.32), (6.35) and

$$l_m(E^h) = \begin{cases} l_h(E) + q_j & \text{if } j \in P,\ j = m, \\ \sum_{i \in \mathcal{O}(E^h)} q_i & \text{if } j \in P, \\ 0 & \text{if } j \in D,\ m = j - n, \\ l_m(E) & \text{otherwise,} \end{cases} \qquad \forall m \in \mathcal{O}(E^h), \tag{6.56}$$

$$\mathcal{S}^{EP}_{mi}(E^h) = \begin{cases} 1 & \text{if } j \in P, j = m,\ \forall i \in P \text{ such that} \\ & \qquad \mathcal{S}^{EP}_{hi}(E) = 1, \\ 1 & \text{if } j \in P, j = m,\ \forall i \in \mathcal{O}(E^h), \\ 1 & \text{if } j \in P, i = m = j, \\ 0 & \text{if } j \in P, \\ 0 & \text{if } j \in D, j - n = m,\ \forall i \in P, \\ \mathcal{S}^{EP}_{mi}(E) & \text{otherwise.} \end{cases} \qquad \forall m, i \in \mathcal{O}(E^h), \tag{6.57}$$

Equation (6.56) replaces equation (6.33) ; if two requests are in the same compartment but none of them are in the stack, their total loads will be the same. Equation (6.57) updates the positions of the items in the compartment. If j is a pickup node then it is on top of the stack, i.e., on top of all other items in the compartment, either in the stack or ejected. If the stack has reached its maximal size, then no order is imposed among all ejected items. If j is a delivery node, then they are no more items linked to $j - n$. Furthermore, label E^h is kept if it respects the time window constraints (6.36) and the capacity constraints (6.37).

Finally, a label E_1 dominates a label E_2 if conditions (6.38)–(6.41), (6.43) and

$$\mathcal{S}^{EP}_{ij}(E_1) \leq \mathcal{S}^{EP}_{ij}(E_2), \qquad \forall i, j \in P, \tag{6.58}$$

hold.

Proposition 6.4.1. *Conditions* (6.38)*–*(6.41)*,* (6.43) *and* (6.58) *constitute a valid dominance criterion whenever* $\bar{c}_{ij}$ *satisfies the delivery triangle inequality.*

Proof. The proof is similar to that of Proposition 6.3.1. We show that for every feasible completion of E_2 there exists a feasible completion of E_1 with no larger reduced cost. Let r be a path extending $R(E_2)$ to node $2n + 1$ such that $(R(E_2), r)$ is feasible with respect to time windows, elementarity constraints, pickup and delivery constraints, capacity constraints and relaxed multi-stack policy. If no such path exists, then clearly one can remove label E_2. Let r' be the path obtained from r by removing the deliveries for each request $i \in P$ with $\mathcal{S}^{EP}_{ii}(E_1) = 0$ and $\mathcal{S}^{EP}_{ii}(E_2) = 1$. Because $(R(E_2), r)$ is feasible with respect to elementarity constraints, and pickup and delivery constraints, then so is $(R(E_1), r')$. Because the triangle inequality is assumed for travel times and $(R(E_2), r)$ is feasible with respect to the time windows, then so is $(R(E_1), r')$. The capacity constraints for each stack are not violated because items in different stacks for E_1 are also in different stacks for E_2, and items in the same stack for E_1 are also in the same stack for E_2, thus the capacity constraints are respected on $(R(E_1), r')$. The order in which the deliveries are performed on $(R(E_1), r')$ is the same as the order on $(R(E_2), r)$, i.e., each item that is in a stack for E_1 is also in the stack for E_2, and each item that has been ejected for E_1 can be ejected or not for E_2. No ejected item for E_2 can be in a stack for E_1. Thus, the relaxed multi-stack policy is not violated. Because $(R(E_2), r)$ is feasible, then so is $(R(E_1), r')$. Because the delivery triangle inequality holds for the reduced cost component c, the cost of r' does not exceed that of r. Thus, $c(E_1) \leq c(E_2)$ implies that the cost of $(R(E_1), r')$ is at most equal to that of $(R(E_2), r)$. Hence, label E_1 dominates label E_2. □

The reader can easily adapt this procedure to the non-elementary version of the shortest path problem.

6.4.2 Infeasible Path Cuts

When solving the shortest path problem with the labeling algorithm presented in the previous section, we might find a path for which the multi-stack policy is not respected. Figures 6.4a and 6.4b present a path and a configuration of the vehicle found with the relaxed multi-stack policy when setting $\kappa = 0$. This configuration does not respect the multi-stack policy, but the items can be rearranged as in Figure 6.4c in order to respect it. Thus, this path is feasible. Figures 6.5a and 6.5b illustrate a path and a configuration of the vehicle found with the relaxed multi-stack policy when setting $\kappa = 0$. In such a case, no rearrangement of the items is possible, and this path is infeasible.

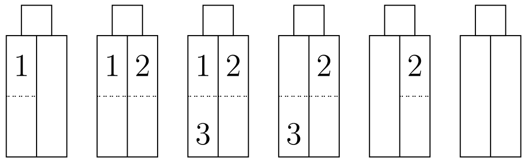
In order to find out whether a path is feasible with respect to the multi-stack policy even if its current configuration is not, Côté *et al.* (2012a) proposed solving a bin packing problem. Instead, our algorithm solves a shortest path problem with multi-stack policy. The labeling algorithm presented in Section 6.3 is applied on the reduced graph containing only the arcs used in the current path. If a solution is found, a rearrangement is possible. Otherwise, the path is infeasible and its corresponding infeasible path inequality (6.20) is added to the RMP. Thus, these inequalities can be reformulated as

$$\sum_{r \in \Omega} \left(\sum_{\mu=1}^{\rho-1} b^r_{i_\mu, i_{\mu+1}} \right) y_r \leq |N(R)| - 2, \quad \forall R = (i_0, ..., i_\rho) \in \mathcal{R}^*. \tag{6.59}$$

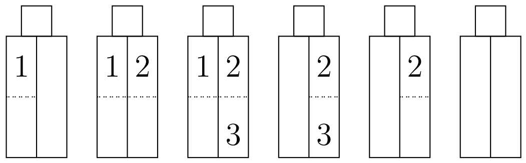
These constraints are separated through an exact enumerative procedure. For every path in a given optimal solution of the master problem, several of these constraints can be violated. The sequential search is then carried out on each active route $r \in \Omega$ with $y_r > 0$, and the first

$$\begin{tikzcd} 0 \arrow[r] & 1^+ \arrow[r] & 2^+ \arrow[r] & 3^+ \arrow[r] & 1^- \arrow[r] & 3^- \arrow[r] & 2^- \arrow[r] & 0 \end{tikzcd}$$

(a) Path



(b) Possible configuration of the vehicle not respecting the multi-stack policy



(c) Rearrangement of the items respecting the multi-stack policy

Figure 6.4 – Path for which a possible rearrangement of the items can be found such that the multi-stack policy is respected, $q_1 = q_2 = q_3 = 1$ and $Q = 2$

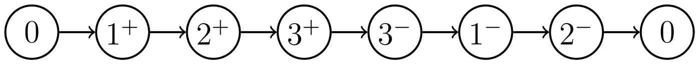
identified violated inequality is added to the master problem. Note that the dual variables of (6.59) affect the reduced cost of the arcs along the corresponding path, but we leave out the details for conciseness reasons.

6.5 Computational Results

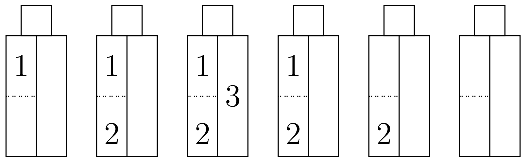
The two branch-price-and-cut algorithms just described were tested on a set of PDPTWMS instances derived from an instance of the TSPLIB. In this section, we report the computational results obtained for these PDPTWMS instances. The instances are solved by considering one, two, and three stacks. All tests were performed on a Linux computer equipped with an Intel(R) Core(TM) i7-3770 processor (3.4 GHz). The algorithms were implemented using the GENCOL library using CPLEX 12.4.0.0 to solve all restricted master problems.

6.5.1 Instances

To test our algorithms, we have generated 198 PDPTWMS instances from the *a280* instance of the TSPLIB by following the ideas of Carrabs *et al.* (2007a,b); Cordeau *et al.* (2010) for the TSPPDL, and of Côté *et al.* (2012a) for the PDTSPMS. Two classes of instances were tested. In the first class, C1, each item has a unit demand, and the total capacity of a vehicle is 6. In the second class, C2, the demand of each item is a random number between 3 and 9, and the capacity of a vehicle is 24 for the one- and two-stack variants, and 27 for the



(a) Path



(b) Possible configuration of the vehicle not respecting the multi-stack policy

Figure 6.5 – Path for which no possible rearrangement of the items can be found such that the multi-stack policy is respected, $q_1 = q_2 = 1$, $q_3 = 2$ and $Q = 2$

three-stack variant.

For each class, we have generated and tested a total of 99 instances in which the number of requests ranges from 25 to 75, i.e., the number of nodes ranges from 51 to 151. For an instance with $2n + 1$ nodes, we have kept the first $2n + 1$ nodes from the *a280* instance of the TSPLIB. For each request, a pickup and a delivery node have been randomly paired, and the time windows have been randomly generated. Three time window horizons were tested : (1) setting $\underline{w}_i \leq 500, \forall i \in P$ and $\underline{w}_i \leq 1000, \forall i \in D$, (2) setting $\underline{w}_i \leq 1000, \forall i \in P$ and $\underline{w}_i \leq 1200, \forall i \in D$, and (3) setting $\underline{w}_i \leq 1500, \forall i \in P$ and $\underline{w}_i \leq 2000, \forall i \in D$. The three time horizons are denoted by 500-1000, 1000-1200, and 1500-2000 in the following. For each time window horizon, three different time window lengths were tested, i.e., 15, 30, and 45.

In all instances, we first aim to minimize the number of vehicles. To this end, a fixed cost of 100,000 is imposed on each arc $(0, j) \in A$ with $j \in P$.

6.5.2 Detailed Computational Results

Table 6.1 presents the number of instances solved optimally for each algorithm using the elementary shortest path problem. For each instance class and number of stacks, 99 instances are tested. A time limit of 7200 seconds (two hours) was imposed for the solution of each instance. The algorithms are denoted as follows : *BPC MS* (branch-price-and-cut algorithm with multi-stack feasible paths) and *BPC Relaxed* (branch-price-and-cut algorithm with relaxed multi-stack paths). For the latter, we also specify the value of κ. We have tested different values of κ and report those with $\kappa = 0$ and $\kappa = 2$.

For any number of stacks and instance class, *BPC MS* solves the largest number of instances. In our experiments, we observed that all instances solved with *BPC Relaxed* are also solved with *BPC MS*. Therefore, we will only present detailed computational results for the *BPC MS* algorithm. We have also tested an adaptation of the ng-path relaxation proposed by Baldacci *et al.* (2010, 2011b) for the VRP and adapted by Cherkesly *et al.* (2014) for the PDPTWL and the non-elementary shortest path problem. Because the instances do not allow many cycles, neither of these two relaxations of the pricing problem has a positive impact on the quality of the lower bound or on the computational time. Thus, we do not present these results.

For the one- and two-stack variants, one can realize that the instances in class C1 are harder to solve than those in class C2. For the instances in class C1, this is probably due to the symmetry between the items, i.e., each item has unit demand. For the three-stack variant, the instances in class C2 prove to be more difficult. This is due to the increase of the maximum number of items simultaneously present in a vehicle. In fact, for each instance in class C1 solved with three stacks, the maximum number of items simultaneously in the vehicle is three (see Table 6.10). Thus, solving the problem with the multi-stack policy for these instances is equivalent to solving the PDPTW.

Tables 6.2–6.4 present summarized results for the PDPTWMS with one, two, and three stacks, respectively, with the BPC MS algorithm when solving the elementary version of the suproblem in all three cases. Detailed computational results are presented in Appendix A. In each table we present, for each instance class, summarized results on each set of 11 instances with a specified time window length and horizon. For each of these 11 instances, the number of nodes ranges from 25 to 75. In each table, the first column indicates the width of the length of the time windows (15, 30 or 45), and the time window horizon, i.e., 500-1000, 1000-1200, and 1500-2000. For example, w15-500-1000 refers to instances that have a time window length of 15 and a time window horizon 500-1000. We present the following information : *NbSolved* the number of instances solved to optimality within the prescribed time limit ; *Sec.*, the average CPU time in seconds ; *Gap (%)*, the average integrity gap in percentage computed as $(z^* - \underline{z})/(z^*)$, where z^* is the optimal solution value and $\underline{z}$ is the lower bound at the

Table 6.1 – Number of instances solved for each algorithm

	BPC MS		BPC Relaxed $\kappa = 0$		BPC Relaxed $\kappa = 2$	
Class	C1	C2	C1	C2	C1	C2
1 stack	77	96	63	85	69	93
2 stacks	27	63	5	11	26	62
3 stacks	89	52	89	21	89	52

root node before adding any cuts ; *Veh*, the average minimal number of vehicles ; and *maxItem*,

the average maximal number of items simultaneously in a vehicle. All these averages are computed over the solved instances. Furthermore, for each class of instances, we report in the row *Weighted average* the averages for the class weighted according to the number of instances solved for each set of 11 instances with a specified time window length and horizon.

We first observe that for both classes of instances and independently of the number of stacks, solving instances with larger time windows and with larger time window horizons is harder. In fact, we can, solve 157, 135 and 112 instances with time window lengths of 15, 30 and 45, respectively, and we can solve 145, 145, and 114 instances with the time window horizons of 500-1000, 1000-1200, and 1500-2000, respectively.

Second, for instances with one and two stacks, instances of class C1 are harder to solve. For the one-stack variant, 77 instances in class C1 and 96 instances in class C2 are solved, and, for the two-stack variant, 27 instances in class C1 and 63 instances in class C2 are solved. This is probably due to the symmetry between the unit demand items. For instances solved with three stacks, we obtain the opposite result which is probably due to the increase of the number of items simultaneously in the vehicle for instances in class C2.

Third, solving instances in classes C1 and C2 with one stack yields better gaps than with two and three stacks. For class C1, the gaps are on average 0.50%, 4.80%, and 3.07% for the one-, two-, and three-stack variants, respectively. For class C2, the gaps are on average 0.46%, 3.59%, and 4.04% for the one-, two-, and three-stack variants, respectively. This is probably due to a lower number of feasible paths with one stack, which also explains why solving instances with one stack yields on average more vehicles in a solution. For class C1, the average number of vehicles is 16.86, 8.74, and 12.53, while for class C2, the average number of vehicles is 17.64, 10.35, and 9.44 for the one-, two-, and three-stack variants, respectively.

Fourth, for instances in class C1, solving them with three stacks yields better gaps than with two stacks, i.e., 3.07% and 4.80% on average, respectively. We can also observe that the average number of vehicles with three stacks is greater than with two stacks, i.e., 12.53 and 8.74 vehicles on average, respectively. This result holds for instances that are solved for both two and three stacks. Interestingly, not all feasible solutions with respect to two stacks are feasible with respect to three stacks because the stack capacities differ. In the proposed instances, each vehicle has a capacity of six : with two stacks, each stack has a capacity of three ; and with three stacks, each stack has a capacity of two. Thus, from a managerial perspective, it is interesting to see that having more stacks in a vehicle does not necessarily reduce the total costs.

Table 6.2 – Summarized computational results for the variant with one stack

Instance	NbSolved	Sec.	Gap (%)	Veh	maxItem
Instances in class C1					
w15-500-1000	11	59.9	0.00	24.18	3.64
w15-1000-1200	11	60.6	0.27	19.18	3.36
w15-1500-2000	7	98.6	0.39	12.71	4.14
w30-500-1000	10	1,783.1	0.53	20.00	3.80
w30-1000-1200	10	971.8	0.31	16.70	4.10
w30-1500-2000	8	654.3	0.00	12.63	4.25
w45-500-1000	7	2,130.4	1.92	15.00	4.57
w45-1000-1200	7	772.2	0.98	13.14	4.00
w45-1500-2000	6	785.1	0.69	11.17	4.17
Weighted average	8.56	777.0	0.50	16.86	3.95
Instances in class C2					
w15-500-1000	11	3.7	0.30	24.09	3.27
w15-1000-1200	11	3.8	0.25	19.18	3.45
w15-1500-2000	11	64.8	0.00	16.45	4.00
w30-500-1000	11	49.5	0.00	21.64	4.00
w30-1000-1200	11	21.5	0.57	18.00	3.64
w30-1500-2000	11	834.9	0.46	13.82	4.27
w45-500-1000	10	310.5	0.73	17.60	4.00
w45-1000-1200	11	267.3	1.24	15.27	3.73
w45-1500-2000	9	478.3	0.67	11.56	3.89
Weighted average	10.67	219.9	0.46	17.64	3.80

Table 6.3 – Summarized computational results for the variant with two stacks

Instance	NbSolved	Sec.	Gap (%)	Veh	maxItem
Instances in class C1					
w15-500-1000	7	547.8	3.08	11.86	4.86
w15-1000-1200	5	2,036.2	1.87	9.80	4.20
w15-1500-2000	2	563.9	4.11	6.00	4.50
w30-500-1000	3	523.7	9.44	8.00	4.67
w30-1000-1200	3	2,396.4	4.80	8.33	5.00
w30-1500-2000	1	162.1	0.01	5.00	6.00
w45-500-1000	2	2,956.5	7.49	7.50	4.50
w45-1000-1200	3	3,480.1	10.95	6.33	5.33
w45-1500-2000	1	17.4	0.00	4.00	4.00
Weighted average	3.00	1,497.7	4.80	8.74	4.74
Instances in class C2					
w15-500-1000	11	541.4	2.75	14.91	4.27
w15-1000-1200	9	939.1	4.82	12.11	3.67
w15-1500-2000	6	1,391.0	5.58	7.83	4.00
w30-500-1000	6	124.0	3.16	10.83	4.50
w30-1000-1200	9	1,070.6	2.43	10.89	3.78
w30-1500-2000	6	1,410.6	2.91	7.00	4.00
w45-500-1000	4	419.7	5.89	8.25	4.25
w45-1000-1200	6	1,037.0	2.80	8.50	4.00
w45-1500-2000	6	1,234.6	3.39	7.17	4.50
Weighted average	7.00	903.2	3.59	10.35	4.08

Table 6.4 – Summarized computational results for the variant with three stacks

Instance	NbSolved	Sec.	Gap (%)	Veh	maxItem
Instances in class C1					
w15-500-1000	11	294.3	3.29	16.00	3.00
w15-1000-1200	11	492.1	3.28	12.45	3.00
w15-1500-2000	10	844.8	4.62	10.80	3.00
w30-500-1000	11	246.6	2.55	15.64	3.00
w30-1000-1200	9	559.9	2.48	10.56	3.00
w30-1500-2000	10	1,584.8	1.71	10.40	3.00
w45-500-1000	11	543.2	1.94	15.09	3.00
w45-1000-1200	9	267.6	5.55	10.78	3.00
w45-1500-2000	7	1,071.2	2.29	8.57	3.00
Weighted average	9.89	635.7	3.07	12.53	3.00
Instances in class C2					
w15-500-1000	10	1,308.4	3.04	13.50	4.60
w15-1000-1200	8	726.5	4.94	11.25	4.25
w15-1500-2000	5	1,764.7	3.47	7.20	5.00
w30-500-1000	5	838.1	4.39	9.60	4.80
w30-1000-1200	7	825.2	3.67	9.43	4.00
w30-1500-2000	4	554.0	6.78	6.00	4.75
w45-500-1000	4	3,033.5	3.14	7.50	5.00
w45-1000-1200	5	287.3	3.68	7.40	4.20
w45-1500-2000	4	1,965.9	5.43	6.25	4.50
Weighted average	5.78	1,179.5	4.04	9.44	4.52

Finally, for instances in class C2, solving them with two stacks yields better gaps than with three stacks, i.e., 3.59% and 4.04% on average, but yields more vehicles, i.e., 10.35 and 9.44 vehicles on average, respectively. In the proposed instances, a vehicle with two stacks has a total capacity of 24 and a vehicle with three stacks has a total capacity of 27. Thus, an increase of 11.1% of the total vehicle capacity decreases, on average, the number of vehicles by 8.8% and the maximum number of items by 9.7% which seems coherent.

6.5.3 Impact of the Number of Stacks

Table 6.5 presents computational results showing the impact of the number of stacks on the total traveled distance, the number of vehicles used, and the maximum number of items simultaneously in a vehicle. Detailed computational results are presented in Appendix B. In the table, we compare the results obtained with one stack to the results obtained with two and three stacks. We present the following information : Δ *Dist* (%), the average relative difference in the total traveled distance, computed as $(Dist_2 - Dist_1)/(Dist_1)$ and $(Dist_3 - Dist_1)/(Dist_1)$, where $Dist_j, j = \{1,2,3\}$, is the distance with j stacks ; Δ *Veh* (%) , the average relative difference in the number of vehicles, computed as $(Veh_2 - Veh_1)/(Veh_1)$ and $(Veh_3 - Veh_1)/(Veh_1)$, where $Veh_j, j = \{1,2,3\}$, is the number of vehicles with j stacks ; and Δ *maxItem* (%), the average relative difference in the maximum number of items simultaneously in a vehicle, computed as $(maxItem_2 - maxItem_1)/(maxItem_1)$ and $(maxItem_3 - maxItem_1)/(maxItem_1)$, where $maxItem_j, j = \{1,2,3\}$, is the maximum number of items simultaneously in a vehicle with j stacks.

For instances in class C1, the total capacity of the vehicle is 6 independently of the number of stacks. First, increasing the number of stacks from one to two and from one to three decreases the total traveled distance by an average of 28.6% and 22.8%, and decreases the number of vehicles used by an average of 39.0% and 29.6%, respectively. Second, increasing the number of stacks from one to two increases the maximum number of items simultaneously in a vehicle by an average of 35.2%. Comparing one with three stacks, the maximum number of requests simultaneously in a vehicle decreases by an average of 21.6%. This last result is counterintuitive, but by examining the optimal solutions with three stacks, the maximum number of items is three and is often reached, whereas with one stack, this maximum is reached less often.

For the class C2, the total capacity of the vehicle is 24 for instances with one and two stacks, and 27 for instances with three stacks. Even though the capacity is not the same for the three-stack variant, we present the impact of the number of stacks on the results to

Table 6.5 – Summarized impact of the number of stacks on the results

	1 stack VS 2 stacks			1 stack VS 3 stacks		
Instance	Δ *Dist* (%)	Δ *Veh* (%)	Δ *maxItem* (%)	Δ *Dist* (%)	Δ *Veh* (%)	Δ *maxItem* (%)
Instances in class C1						
w15-500-1000	–32.0	–41.4	45.2	–28.3	–34.0	–15.0
w15-1000-1200	–25.9	–36.3	31.7	–23.3	–34.2	–9.1
w15-1500-2000	–31.7	–45.5	29.2	–22.9	–26.4	–25.7
w30-500-1000	–30.1	–40.1	30.6	–22.6	–26.3	–20.0
w30-1000-1200	–25.1	–34.3	38.9	–20.1	–33.4	–23.9
w30-1500-2000	–30.0	–37.5	50.0	–19.2	–26.6	–27.5
w45-500-1000	–27.1	–37.4	12.5	–17.9	–18.9	–33.6
w45-1000-1200	–27.0	–34.7	33.3	–24.2	–29.8	–25.0
w45-1500-2000	–23.9	–50.0	33.3	–25.1	–32.2	–25.8
Weighted average	–28.6	–39.0	35.2	–22.8	–29.6	–21.6
Instances in class C2						
w15-500-1000	–26.2	–37.6	32.6	–29.6	–40.4	45.0
w15-1000-1200	–23.2	–32.1	11.1	–26.2	–36.9	31.3
w15-1500-2000	–22.0	–38.9	11.1	–28.0	–42.0	41.7
w30-500-1000	–29.7	–42.2	20.8	–36.6	–49.0	30.0
w30-1000-1200	–18.5	–33.2	5.2	–22.2	–34.9	14.3
w30-1500-2000	–17.9	–36.0	5.6	–26.5	–40.2	31.3
w45-500-1000	–19.6	–30.4	14.6	–21.7	–37.1	33.3
w45-1000-1200	–21.1	–33.2	16.7	–26.4	–37.2	25.0
w45-1500-2000	–14.9	–30.3	18.1	–26.5	–35.2	22.9
Weighted average	–21.8	–35.0	15.8	–27.2	–39.1	31.4

show the general trend. We first observe that increasing the number of stacks from one to two and from one to three decreases the total traveled distance by an average of 21.8% and 27.2%, respectively. It also decreases the number of vehicles used by an average of 35.0% and 39.1%, and increases the maximum number of requests simultaneously in a vehicle by an average of 15.8% and 31.4%, respectively.

Thus, for both classes C1 and C2, increasing the number of stacks from one to two has a positive impact on the total traveled distance and on the minimal number of vehicles needed. Interestingly, the additional gain of three stacks is significantly smaller than increasing the number of stacks from one to two.

6.6 Conclusions

In this paper, we have introduced the PDPTWMS and described two column generation algorithms to solve it. An ad hoc labeling algorithm for shortest path problems with multiple stacks is proposed and implemented. Moreover, we have adapted the hybrid branch-price-and-cut algorithm of Cherkesly *et al.* (2014) for the PDPTWL to the PDPTWMS. In addition, we have introduced a new notation to represent a stack in a vehicle which can be adapted to variants of the PDPTW with loading constraints such as the PDPTW with handling costs. Instances involving up to 75 requests and three stacks were solved to optimality within two hours of computational time. On the PDPTWL instances, Cherkesly *et al.* (2014) had shown that, for their instances, the *BPC Relaxed* seemed to outperform the *BPC MS*. On our new instances, we obtain the opposite result, i.e., the *BPC Relaxed* is slightly outperformed by the *BPC MS*. Our results also show that increasing the number of stacks from one to two has a positive impact on the total traveled distance and on the minimal number of vehicles used, but increasing it from two to three does not yield a significant additional gain.

Appendix A. Detailed Computational Results

This appendix presents the detailed computational results on our test instances. Tables 6.6–6.11 present the results obtained when solving each instance with the branch-price-and-cut algorithm with multi-stack feasible paths, and solving the elementary shortest path problem. In each table, the first column indicates the name of the instance corresponding to its number of nodes, its instance class (C1 or C2), the length of the time windows (15, 30 or 45), and the time window horizon. For example, instance a280-51-c1-w15-500-1000 involves 51 nodes, is in class C1, has a time window length of 15, and a time window horizon such that $\underline{w}_i \leq 500, \forall i \in P$ and $\underline{w}_i \leq 1000, \forall i \in D$. We present the following information : *Sec.*, the CPU time in seconds ; $\underline{z}$, the lower bound at the root node before adding any cuts ; z^*, the optimal solution value ; *Veh.*, the minimal number of vehicles in the optimal solution ; *maxItem*, the maximal number of items simultaneously in a vehicle ; *OC*, the total number of constraints (6.46), (6.47), and (6.49) added to the master problem ; and *B*, the number of nodes in the search tree including the root node. For all the instances tested with our algorithms, no feasible solution was found whenever the time limit was reached. Thus, we do not report a lower bound value even if one was found.

Table 6.6 – Computational results for the variant with one stack for instances in class C1 *(cont'd)*

Instance	Sec.	$\underline{z}$	z^*	Veh.	maxItem	OC	B
a280-51-c1-w15-500-1000	0.0	1,105,647.0	1,105,647.0	11	4	0	1
a280-61-c1-w15-500-1000	0.1	1,808,284.8	1,808,284.8	18	3	0	1
a280-71-c1-w15-500-1000	0.4	1,709,058.6	1,709,058.6	17	3	0	1
a280-81-c1-w15-500-1000	2.1	2,212,102.1	2,212,102.1	22	3	0	1
a280-91-c1-w15-500-1000	1.2	2,514,106.2	2,514,106.2	25	4	0	1
a280-101-c1-w15-500-1000	1.6	2,514,863.5	2,514,863.5	25	3	0	1
a280-111-c1-w15-500-1000	0.5	2,516,452.5	2,516,452.5	25	4	0	1
a280-121-c1-w15-500-1000	48.4	2,917,967.1	2,917,967.1	29	4	0	1
a280-131-c1-w15-500-1000	53.9	2,917,267.7	2,917,267.7	29	3	0	1
a280-141-c1-w15-500-1000	19.2	3,419,880.8	3,419,880.8	34	4	0	1
a280-151-c1-w15-500-1000	532.0	3,118,717.5	3,118,717.5	31	5	0	1
a280-51-c1-w15-1000-1200	0.1	1,105,937.2	1,105,937.2	11	3	0	1
a280-61-c1-w15-1000-1200	0.2	1,207,347.2	1,207,347.2	12	3	0	1
a280-71-c1-w15-1000-1200	0.5	1,308,216.2	1,308,216.2	13	3	0	1
a280-81-c1-w15-1000-1200	12.0	1,509,621.7	1,509,621.7	15	3	0	1
a280-91-c1-w15-1000-1200	8.2	2,112,198.8	2,112,198.8	21	4	0	1
a280-101-c1-w15-1000-1200	3.3	1,663,127.4	1,713,265.2	17	4	8	3
a280-111-c1-w15-1000-1200	6.4	2,113,948.0	2,113,948.0	21	3	0	1
a280-121-c1-w15-1000-1200	373.4	2,315,420.2	2,315,420.2	23	4	0	1
a280-131-c1-w15-1000-1200	119.8	2,114,163.8	2,114,163.8	21	4	0	1
a280-141-c1-w15-1000-1200	102.6	2,919,615.0	2,919,615.0	29	3	0	1
a280-151-c1-w15-1000-1200	40.0	2,817,812.9	2,817,812.9	28	3	0	1
a280-51-c1-w15-1500-2000	0.1	1,105,889.5	1,105,889.5	11	3	0	1
a280-61-c1-w15-1500-2000	1.7	1,106,952.5	1,106,952.5	11	4	0	1
a280-71-c1-w15-1500-2000	17.9	1,006,670.1	1,006,670.1	10	5	0	1
a280-81-c1-w15-1500-2000	6.2	1,008,842.1	1,008,842.1	10	5	0	1

Table 6.6 – Computational results for the variant with one stack for instances in class C1 *(cont'd)*

Instance	Sec.	$\underline{z}$	z^*	Veh.	maxItem	OC	B
a280-91-c1-w15-1500-2000	288.5	1,409,820.3	1,409,820.3	14	4	0	1
a280-101-c1-w15-1500-2000	68.6	1,510,936.0	1,510,936.0	15	4	0	1
a280-111-c1-w15-1500-2000	307.4	1,763,132.1	1,813,022.0	18	4	0	2
a280-121-c1-w15-1500-2000							
a280-131-c1-w15-1500-2000							
a280-141-c1-w15-1500-2000							
a280-151-c1-w15-1500-2000							
a280-51-c1-w30-500-1000	0.5	1,004,669.6	1,004,669.6	10	4	0	1
a280-61-c1-w30-500-1000	1.6	1,407,454.9	1,407,454.9	14	3	0	1
a280-71-c1-w30-500-1000	0.5	1,608,457.6	1,608,457.6	16	4	0	1
a280-81-c1-w30-500-1000	96.2	1,660,592.6	1,710,398.8	17	3	0	2
a280-91-c1-w30-500-1000	150.9	1,811,246.7	1,811,246.7	18	4	0	1
a280-101-c1-w30-500-1000	31.2	2,213,749.1	2,213,759.2	22	4	5	2
a280-111-c1-w30-500-1000	1,796.0	2,465,393.9	2,515,458.0	25	4	8	5
a280-121-c1-w30-500-1000	6,988.8	2,515,845.2	2,515,845.2	25	4	0	1
a280-131-c1-w30-500-1000	2,283.1	2,415,711.4	2,415,711.4	24	4	0	1
a280-141-c1-w30-500-1000							
a280-151-c1-w30-500-1000	6,481.7	2,906,085.9	2,918,603.3	29	4	2	3
a280-51-c1-w30-1000-1200	0.4	1,004,966.1	1,004,966.1	10	4	0	1
a280-61-c1-w30-1000-1200	0.5	1,206,358.7	1,206,358.7	12	3	0	1
a280-71-c1-w30-1000-1200	10.2	1,307,419.2	1,307,419.2	13	4	0	1
a280-81-c1-w30-1000-1200	20.1	1,409,119.5	1,409,119.5	14	4	0	1
a280-91-c1-w30-1000-1200	8.8	1,509,756.4	1,509,756.4	15	5	0	1
a280-101-c1-w30-1000-1200	22.1	1,562,241.1	1,612,187.1	16	4	0	6
a280-111-c1-w30-1000-1200	120.2	1,913,682.3	1,913,683.8	19	4	7	2
a280-121-c1-w30-1000-1200	1,647.7	2,215,871.3	2,215,871.3	22	4	0	1
a280-131-c1-w30-1000-1200							
a280-141-c1-w30-1000-1200	2,486.5	2,215,548.2	2,215,548.2	22	4	0	1
a280-151-c1-w30-1000-1200	5,401.1	2,415,924.4	2,415,924.4	24	5	0	1
a280-51-c1-w30-1500-2000	0.1	804,751.1	804,751.1	8	4	0	1
a280-61-c1-w30-1500-2000	6.5	805,473.4	805,473.4	8	4	0	1
a280-71-c1-w30-1500-2000	27.8	1,307,372.4	1,307,372.4	13	3	0	1
a280-81-c1-w30-1500-2000	16.6	1,208,314.2	1,208,314.2	12	4	0	1
a280-91-c1-w30-1500-2000	431.8	1,309,938.8	1,309,938.8	13	5	0	1

Table 6.6 – Computational results for the variant with one stack for instances in class C1 *(cont'd)*

Instance	Sec.	$\underline{z}$	z^*	Veh.	maxItem	OC	B
a280-101-c1-w30-1500-2000	412.2	1,411,245.3	1,411,245.3	14	4	0	1
a280-111-c1-w30-1500-2000	598.6	1,512,520.4	1,512,520.4	15	5	0	1
a280-121-c1-w30-1500-2000	3,740.8	1,813,294.2	1,813,294.2	18	5	0	1
a280-131-c1-w30-1500-2000							
a280-141-c1-w30-1500-2000							
a280-151-c1-w30-1500-2000							
a280-51-c1-w45-500-1000	1.7	1,105,693.2	1,105,693.2	11	4	0	1
a280-61-c1-w45-500-1000	9.9	1,256,820.3	1,306,849.4	13	4	0	2
a280-71-c1-w45-500-1000	269.2	1,073,686.5	1,107,077.6	11	5	14	6
a280-81-c1-w45-500-1000	1,382.7	1,459,158.4	1,509,170.9	15	4	12	3
a280-91-c1-w45-500-1000	803.1	1,609,893.8	1,609,893.8	16	5	0	1
a280-101-c1-w45-500-1000	6,504.3	1,946,106.6	2,012,805.5	20	5	0	2
a280-111-c1-w45-500-1000	5,941.8	1,911,894.5	1,911,894.5	19	5	0	1
a280-121-c1-w45-500-1000							
a280-131-c1-w45-500-1000							
a280-141-c1-w45-500-1000							
a280-151-c1-w45-500-1000							
a280-51-c1-w45-1000-1200	1.1	804,649.0	804,649.0	8	4	0	1
a280-61-c1-w45-1000-1200	1.6	905,443.6	905,443.6	9	4	0	1
a280-71-c1-w45-1000-1200	3.1	1,207,359.0	1,207,359.0	12	4	0	1
a280-81-c1-w45-1000-1200	3,041.7	1,358,979.4	1,409,060.4	14	4	0	62
a280-91-c1-w45-1000-1200	123.1	1,459,618.5	1,509,809.5	15	4	16	3
a280-101-c1-w45-1000-1200	1,751.2	1,711,792.9	1,711,812.7	17	4	13	12
a280-111-c1-w45-1000-1200	483.3	1,713,895.5	1,713,895.5	17	4	0	1
a280-121-c1-w45-1000-1200							
a280-131-c1-w45-1000-1200							
a280-141-c1-w45-1000-1200							
a280-151-c1-w45-1000-1200							
a280-51-c1-w45-1500-2000	0.1	805,142.4	805,142.4	8	3	0	1
a280-61-c1-w45-1500-2000	4.5	1,006,154.4	1,006,154.4	10	5	0	1
a280-71-c1-w45-1500-2000	36.3	1,007,200.9	1,007,200.9	10	5	0	1
a280-81-c1-w45-1500-2000	11.6	1,308,730.7	1,308,730.7	13	4	0	1
a280-91-c1-w45-1500-2000	50.7	1,159,978.6	1,209,905.5	12	4	0	2
a280-101-c1-w45-1500-2000	4,607.2	1,411,697.7	1,411,697.7	14	4	0	1

Table 6.6 – Computational results for the variant with one stack for instances in class C1 *(cont'd and end)*

Instance	Sec.	$\underline{z}$	z^*	Veh.	maxItem	OC	B
a280-111-c1-w45-1500-2000							
a280-121-c1-w45-1500-2000							
a280-131-c1-w45-1500-2000							
a280-141-c1-w45-1500-2000							
a280-151-c1-w45-1500-2000							

Table 6.7 – Computational results for the variant with one stack for instances in class C2 *(cont'd)*

Instance	Sec.	$\underline{z}$	z^*	Veh.	maxItem	OC	B
a280-51-c2-w15-500-1000	0.0	1,506,585.3	1,506,585.3	15	3	0	1
a280-61-c2-w15-500-1000	0.2	1,457,204.4	1,507,163.0	15	3	0	2
a280-71-c2-w15-500-1000	0.2	1,508,562.9	1,508,562.9	15	3	0	1
a280-81-c2-w15-500-1000	0.4	1,809,825.0	1,809,825.0	18	4	0	1
a280-91-c2-w15-500-1000	1.1	1,911,167.3	1,911,167.3	19	3	0	1
a280-101-c2-w15-500-1000	0.5	2,716,189.8	2,716,189.8	27	3	0	1
a280-111-c2-w15-500-1000	1.1	2,817,077.5	2,817,077.5	28	3	0	1
a280-121-c2-w15-500-1000	11.8	2,616,706.3	2,616,706.3	26	4	0	1
a280-131-c2-w15-500-1000	8.4	3,418,753.8	3,418,753.8	34	3	0	1
a280-141-c2-w15-500-1000	12.0	3,118,588.4	3,118,588.4	31	3	0	1
a280-151-c2-w15-500-1000	5.1	3,721,381.7	3,721,381.7	37	4	0	1
a280-51-c2-w15-1000-1200	0.0	1,206,090.1	1,206,090.1	12	3	0	1
a280-61-c2-w15-1000-1200	0.0	1,608,096.6	1,608,096.6	16	3	0	1
a280-71-c2-w15-1000-1200	0.1	1,407,622.1	1,407,622.1	14	3	0	1
a280-81-c2-w15-1000-1200	0.4	1,811,171.3	1,811,171.3	18	3	0	1
a280-91-c2-w15-1000-1200	0.5	1,811,570.2	1,811,570.2	18	4	0	1
a280-101-c2-w15-1000-1200	1.7	1,712,400.2	1,712,400.2	17	4	0	1
a280-111-c2-w15-1000-1200	2.4	1,913,406.5	1,913,406.5	19	3	0	1
a280-121-c2-w15-1000-1200	6.6	1,762,821.5	1,812,664.0	18	4	0	2
a280-131-c2-w15-1000-1200	6.9	2,215,077.3	2,215,077.3	22	4	0	1
a280-141-c2-w15-1000-1200	8.4	3,017,913.1	3,017,913.1	30	3	0	1
a280-151-c2-w15-1000-1200	15.3	2,718,035.8	2,718,035.8	27	4	0	1
a280-51-c2-w15-1500-2000	0.3	1,105,556.8	1,105,556.8	11	3	0	1
a280-61-c2-w15-1500-2000	0.1	1,006,239.9	1,006,239.9	10	4	0	1
a280-71-c2-w15-1500-2000	1.2	1,208,386.1	1,208,386.1	12	4	0	1
a280-81-c2-w15-1500-2000	2.9	1,609,497.7	1,609,497.7	16	3	0	1
a280-91-c2-w15-1500-2000	3.3	1,309,653.6	1,309,653.6	13	4	0	1
a280-101-c2-w15-1500-2000	7.3	1,511,317.0	1,511,317.0	15	4	0	1
a280-111-c2-w15-1500-2000	58.2	1,913,589.1	1,913,589.1	19	5	0	1
a280-121-c2-w15-1500-2000	57.2	1,613,194.9	1,613,194.9	16	4	0	1
a280-131-c2-w15-1500-2000	47.6	2,214,746.1	2,214,746.1	22	4	0	1
a280-141-c2-w15-1500-2000	75.7	2,415,844.1	2,415,844.1	24	5	0	1
a280-151-c2-w15-1500-2000	459.1	2,317,319.0	2,317,319.0	23	4	0	1

Table 6.7 – Computational results for the variant with one stack for instances in class C2 *(cont'd)*

Instance	Sec.	$\underline{z}$	z^*	Veh.	maxItem	OC	B
a280-51-c2-w30-500-1000	0.0	1,306,410.9	1,306,410.9	13	3	0	1
a280-61-c2-w30-500-1000	0.5	1,607,490.2	1,607,490.2	16	3	0	1
a280-71-c2-w30-500-1000	0.6	1,809,178.3	1,809,178.3	18	4	0	1
a280-81-c2-w30-500-1000	6.3	1,910,466.8	1,910,466.8	19	4	0	1
a280-91-c2-w30-500-1000	3.7	2,212,887.0	2,212,923.8	22	4	8	4
a280-101-c2-w30-500-1000	3.4	2,414,303.6	2,414,303.6	24	5	0	1
a280-111-c2-w30-500-1000	28.0	2,314,435.2	2,314,435.2	23	4	0	1
a280-121-c2-w30-500-1000	91.9	2,415,380.8	2,415,380.8	24	4	0	1
a280-131-c2-w30-500-1000	86.2	2,616,226.7	2,616,226.7	26	4	0	1
a280-141-c2-w30-500-1000	134.4	2,716,713.0	2,716,713.0	27	4	0	1
a280-151-c2-w30-500-1000	189.8	2,616,748.4	2,616,774.5	26	5	19	2
a280-51-c2-w30-1000-1200	0.0	904,649.6	904,649.6	9	3	0	1
a280-61-c2-w30-1000-1200	0.7	1,005,432.3	1,005,437.1	10	3	0	3
a280-71-c2-w30-1000-1200	1.0	1,407,500.7	1,407,507.1	14	5	4	2
a280-81-c2-w30-1000-1200	0.9	1,409,499.0	1,409,499.0	14	4	0	1
a280-91-c2-w30-1000-1200	1.0	1,811,139.4	1,811,139.4	18	3	0	1
a280-101-c2-w30-1000-1200	4.9	2,013,638.1	2,013,638.1	20	3	0	1
a280-111-c2-w30-1000-1200	51.1	1,863,134.4	1,913,243.5	19	4	14	28
a280-121-c2-w30-1000-1200	54.9	2,190,325.7	2,215,280.7	22	4	0	2
a280-131-c2-w30-1000-1200	16.3	1,963,883.7	2,013,900.9	20	4	0	2
a280-141-c2-w30-1000-1200	35.3	2,616,709.1	2,616,709.1	26	4	0	1
a280-151-c2-w30-1000-1200	70.6	2,616,560.0	2,616,560.0	26	3	0	1
a280-51-c2-w30-1500-2000	0.2	904,923.5	904,923.5	9	3	0	1
a280-61-c2-w30-1500-2000	0.5	706,464.3	706,464.3	7	4	0	1
a280-71-c2-w30-1500-2000	1.2	1,307,458.4	1,307,458.4	13	4	0	1
a280-81-c2-w30-1500-2000	2.1	1,208,839.3	1,208,839.3	12	4	0	1
a280-91-c2-w30-1500-2000	4.8	1,109,237.8	1,109,237.8	11	4	0	1
a280-101-c2-w30-1500-2000	58.2	1,245,157.2	1,311,314.2	13	5	0	2
a280-111-c2-w30-1500-2000	15.1	1,411,640.6	1,411,640.6	14	4	0	1
a280-121-c2-w30-1500-2000	118.0	1,512,718.8	1,512,718.8	15	5	0	1
a280-131-c2-w30-1500-2000	326.5	1,713,721.4	1,713,721.4	17	5	0	1
a280-141-c2-w30-1500-2000	4,938.5	1,914,710.9	1,914,755.5	19	5	2	2
a280-151-c2-w30-1500-2000	3,719.1	2,216,229.3	2,216,235.2	22	4	9	4

Table 6.7 – Computational results for the variant with one stack for instances in class C2 *(cont'd and end)*

Instance	Sec.	$\underline{z}$	z^*	Veh.	maxItem	OC	B
a280-51-c2-w45-500-1000	0.5	1,005,077.4	1,005,077.4	10	3	0	1
a280-61-c2-w45-500-1000	20.2	1,005,400.6	1,005,434.9	10	4	20	5
a280-71-c2-w45-500-1000	16.3	1,307,698.0	1,307,744.3	13	4	14	6
a280-81-c2-w45-500-1000	3.6	1,509,460.8	1,509,460.8	15	4	0	1
a280-91-c2-w45-500-1000	8.3	1,409,459.2	1,409,459.2	14	4	0	1
a280-101-c2-w45-500-1000	163.9	1,807,855.2	1,912,082.1	19	4	43	4
a280-111-c2-w45-500-1000	105.6	1,913,493.3	1,913,493.3	19	4	0	1
a280-121-c2-w45-500-1000							
a280-131-c2-w45-500-1000	226.9	2,214,734.3	2,214,734.3	22	4	0	1
a280-141-c2-w45-500-1000	780.2	2,667,246.8	2,716,908.3	27	4	0	2
a280-151-c2-w45-500-1000	1,779.5	2,716,695.1	2,716,695.1	27	5	0	1
a280-51-c2-w45-1000-1200	0.1	1,005,255.3	1,005,255.3	10	3	0	1
a280-61-c2-w45-1000-1200	0.4	1,106,345.6	1,106,345.6	11	3	0	1
a280-71-c2-w45-1000-1200	1.6	1,206,790.8	1,206,858.9	12	4	6	2
a280-81-c2-w45-1000-1200	3.6	1,157,910.0	1,207,777.5	12	3	0	2
a280-91-c2-w45-1000-1200	2.9	1,410,414.7	1,410,414.7	14	4	0	1
a280-101-c2-w45-1000-1200	11.5	1,662,326.9	1,712,007.2	17	4	13	3
a280-111-c2-w45-1000-1200	16.7	1,612,507.0	1,612,507.0	16	4	0	1
a280-121-c2-w45-1000-1200	152.1	1,813,012.4	1,813,012.4	18	4	0	1
a280-131-c2-w45-1000-1200	745.3	1,813,883.3	1,913,259.6	19	4	25	17
a280-141-c2-w45-1000-1200	1,452.3	1,789,304.4	1,814,371.8	18	4	49	32
a280-151-c2-w45-1000-1200	553.3	2,116,420.0	2,116,463.9	21	4	12	5
a280-51-c2-w45-1500-2000	0.2	705,047.1	705,047.1	7	4	0	1
a280-61-c2-w45-1500-2000	0.5	1,106,182.1	1,106,182.1	11	3	0	1
a280-71-c2-w45-1500-2000	12.9	1,039,596.5	1,106,349.9	11	4	9	3
a280-81-c2-w45-1500-2000	20.1	908,183.5	908,466.8	9	4	9	9
a280-91-c2-w45-1500-2000	3.5	1,208,948.2	1,208,948.2	12	4	0	1
a280-101-c2-w45-1500-2000	23.9	1,211,215.2	1,211,215.2	12	4	0	1
a280-111-c2-w45-1500-2000	17.9	1,210,969.4	1,210,969.4	12	4	0	1
a280-121-c2-w45-1500-2000	368.5	1,413,167.6	1,413,207.2	14	4	4	5
a280-131-c2-w45-1500-2000	3,857.3	1,613,115.5	1,613,128.6	16	4	10	8
a280-141-c2-w45-1500-2000							
a280-151-c2-w45-1500-2000							

Table 6.8 – Computational results for the variant with two stacks for instances in class C1 *(cont'd)*

Instance	Sec.	$\underline{z}$	z^*	Veh.	maxItem	OC	B
a280-51-c1-w15-500-1000	1.8	704,140.1	704,140.1	7	5	0	1
a280-61-c1-w15-500-1000	51.5	955,662.7	1,005,105.4	10	5	0	2
a280-71-c1-w15-500-1000	74.3	1,014,253.7	1,106,783.1	11	4	13	4
a280-81-c1-w15-500-1000	867.4	1,207,975.7	1,207,975.7	12	6	0	1
a280-91-c1-w15-500-1000	1,677.1	1,309,472.0	1,309,504.1	13	4	3	8
a280-101-c1-w15-500-1000	693.5	1,359,661.3	1,409,613.4	14	5	0	2
a280-111-c1-w15-500-1000	469.2	1,535,073.1	1,611,220.5	16	5	17	3
a280-121-c1-w15-500-1000							
a280-131-c1-w15-500-1000							
a280-141-c1-w15-500-1000							
a280-151-c1-w15-500-1000							
a280-51-c1-w15-1000-1200	3.6	671,088.0	704,241.7	7	3	0	2
a280-61-c1-w15-1000-1200	27.1	805,276.5	805,276.5	8	5	0	1
a280-71-c1-w15-1000-1200	682.9	881,291.8	906,162.9	9	4	17	5
a280-81-c1-w15-1000-1200							
a280-91-c1-w15-1000-1200	4,935.1	1,209,115.3	1,209,115.3	12	5	0	1
a280-101-c1-w15-1000-1200							
a280-111-c1-w15-1000-1200	4,532.5	1,285,981.9	1,310,818.1	13	4	8	3
a280-121-c1-w15-1000-1200							
a280-131-c1-w15-1000-1200							
a280-141-c1-w15-1000-1200							
a280-151-c1-w15-1000-1200							
a280-51-c1-w15-1500-2000	340.4	554,372.4	604,017.4	6	4	0	2
a280-61-c1-w15-1500-2000	787.4	604,751.2	604,751.2	6	5	0	1
a280-71-c1-w15-1500-2000							
a280-81-c1-w15-1500-2000							
a280-91-c1-w15-1500-2000							
a280-101-c1-w15-1500-2000							
a280-111-c1-w15-1500-2000							
a280-121-c1-w15-1500-2000							
a280-131-c1-w15-1500-2000							
a280-141-c1-w15-1500-2000							
a280-151-c1-w15-1500-2000							

Table 6.8 – Computational results for the variant with two stacks for instances in class C1 *(cont'd)*

Instance	Sec.	$\underline{z}$	z^*	Veh.	maxItem	OC	B
a280-51-c1-w30-500-1000	711.5	553,767.5	603,547.1	6	5	7	3
a280-61-c1-w30-500-1000	494.6	714,272.0	804,773.3	8	5	21	3
a280-71-c1-w30-500-1000	365.0	917,222.5	1,005,906.0	10	4	25	7
a280-81-c1-w30-500-1000							
a280-91-c1-w30-500-1000							
a280-101-c1-w30-500-1000							
a280-111-c1-w30-500-1000							
a280-121-c1-w30-500-1000							
a280-131-c1-w30-500-1000							
a280-141-c1-w30-500-1000							
a280-151-c1-w30-500-1000							
a280-51-c1-w30-1000-1200	232.1	653,619.3	703,716.1	7	5	10	3
a280-61-c1-w30-1000-1200	165.6	704,727.8	704,727.8	7	5	0	1
a280-71-c1-w30-1000-1200							
a280-81-c1-w30-1000-1200							
a280-91-c1-w30-1000-1200							
a280-101-c1-w30-1000-1200	6,791.4	1,028,448.3	1,109,220.7	11	5	34	5
a280-111-c1-w30-1000-1200							
a280-121-c1-w30-1000-1200							
a280-131-c1-w30-1000-1200							
a280-141-c1-w30-1000-1200							
a280-151-c1-w30-1000-1200							
a280-51-c1-w30-1500-2000	162.1	503,285.3	503,325.0	5	6	39	16
a280-61-c1-w30-1500-2000							
a280-71-c1-w30-1500-2000							
a280-81-c1-w30-1500-2000							
a280-91-c1-w30-1500-2000							
a280-101-c1-w30-1500-2000							
a280-111-c1-w30-1500-2000							
a280-121-c1-w30-1500-2000							
a280-131-c1-w30-1500-2000							
a280-141-c1-w30-1500-2000							
a280-151-c1-w30-1500-2000							

Table 6.8 – Computational results for the variant with two stacks for instances in class C1 *(cont'd and end)*

Instance	Sec.	$\underline{z}$	z^*	Veh.	maxItem	OC	B
a280-51-c1-w45-500-1000	2,845.0	644,466.8	704,103.9	7	4	27	21
a280-61-c1-w45-500-1000	3,068.0	752,580.7	805,048.0	8	5	2	3
a280-71-c1-w45-500-1000							
a280-81-c1-w45-500-1000							
a280-91-c1-w45-500-1000							
a280-101-c1-w45-500-1000							
a280-111-c1-w45-500-1000							
a280-121-c1-w45-500-1000							
a280-131-c1-w45-500-1000							
a280-141-c1-w45-500-1000							
a280-151-c1-w45-500-1000							
a280-51-c1-w45-1000-1200	1,198.3	472,853.8	503,405.3	5	5	5	3
a280-61-c1-w45-1000-1200	3,559.5	516,846.8	603,928.4	6	6	29	5
a280-71-c1-w45-1000-1200	5,682.5	705,751.0	805,415.2	8	5	55	17
a280-81-c1-w45-1000-1200							
a280-91-c1-w45-1000-1200							
a280-101-c1-w45-1000-1200							
a280-111-c1-w45-1000-1200							
a280-121-c1-w45-1000-1200							
a280-131-c1-w45-1000-1200							
a280-141-c1-w45-1000-1200							
a280-151-c1-w45-1000-1200							
a280-51-c1-w45-1500-2000	17.4	403,914.1	403,914.1	4	4	0	1
a280-61-c1-w45-1500-2000							
a280-71-c1-w45-1500-2000							
a280-81-c1-w45-1500-2000							
a280-91-c1-w45-1500-2000							
a280-101-c1-w45-1500-2000							
a280-111-c1-w45-1500-2000							
a280-121-c1-w45-1500-2000							
a280-131-c1-w45-1500-2000							
a280-141-c1-w45-1500-2000							
a280-151-c1-w45-1500-2000							

Table 6.9 – Computational results for the variant with two stacks for instances in class C2 *(cont'd)*

Instance	Sec.	$\underline{z}$	z^*	Veh.	maxItem	OC	B
a280-51-c2-w15-500-1000	6.1	804,375.0	804,506.1	8	4	48	16
a280-61-c2-w15-500-1000	1.9	867,904.7	905,317.9	9	4	0	2
a280-71-c2-w15-500-1000	6.0	956,421.7	1,006,320.0	10	4	0	2
a280-81-c2-w15-500-1000	8.7	1,250,523.1	1,307,681.1	13	5	15	3
a280-91-c2-w15-500-1000	110.3	1,258,642.3	1,308,469.4	13	5	26	11
a280-101-c2-w15-500-1000	54.0	1,611,601.6	1,611,817.0	16	4	25	13
a280-111-c2-w15-500-1000	69.9	1,787,934.1	1,812,796.4	18	4	12	4
a280-121-c2-w15-500-1000	2,738.1	1,626,114.1	1,712,384.1	17	4	53	39
a280-131-c2-w15-500-1000	2,177.2	1,797,801.6	1,813,249.3	18	5	36	37
a280-141-c2-w15-500-1000	551.6	1,964,742.4	2,014,119.1	20	4	35	10
a280-151-c2-w15-500-1000	231.2	2,142,837.7	2,215,602.5	22	4	20	4
a280-51-c2-w15-1000-1200	1.0	854,650.1	904,616.7	9	3	7	3
a280-61-c2-w15-1000-1200	1.4	822,315.4	905,305.4	9	3	0	2
a280-71-c2-w15-1000-1200	3.8	955,943.9	1,005,845.1	10	3	11	3
a280-81-c2-w15-1000-1200	41.1	1,174,609.8	1,208,065.7	12	4	23	11
a280-91-c2-w15-1000-1200	82.1	1,174,241.1	1,209,279.3	12	4	19	22
a280-101-c2-w15-1000-1200	563.8	1,222,364.3	1,310,248.8	13	4	64	39
a280-111-c2-w15-1000-1200	1,012.1	1,256,198.9	1,311,126.3	13	4	69	104
a280-121-c2-w15-1000-1200	669.2	1,138,498.8	1,209,965.6	12	4	54	9
a280-131-c2-w15-1000-1200							
a280-141-c2-w15-1000-1200	6,077.2	1,888,867.7	1,913,702.4	19	4	31	64
a280-151-c2-w15-1000-1200							
a280-51-c2-w15-1500-2000	8.1	554,299.1	603,995.5	6	4	31	4
a280-61-c2-w15-1500-2000	8.3	605,268.3	605,570.0	6	4	26	4
a280-71-c2-w15-1500-2000	153.5	726,483.4	805,775.4	8	4	27	9
a280-81-c2-w15-1500-2000	324.0	824,160.5	906,454.0	9	4	0	2
a280-91-c2-w15-1500-2000	749.4	865,179.4	908,308.7	9	4	61	18
a280-101-c2-w15-1500-2000	7,102.7	895,442.9	909,507.8	9	4	119	160
a280-111-c2-w15-1500-2000							
a280-121-c2-w15-1500-2000							
a280-131-c2-w15-1500-2000							
a280-141-c2-w15-1500-2000							
a280-151-c2-w15-1500-2000							

Table 6.9 – Computational results for the variant with two stacks for instances in class C2 *(cont'd)*

Instance	Sec.	$\underline{z}$	z^*	Veh.	maxItem	OC	B
a280-51-c2-w30-500-1000	2.1	679,392.0	704,319.5	7	4	0	2
a280-61-c2-w30-500-1000	33.2	855,367.6	905,162.2	9	5	47	4
a280-71-c2-w30-500-1000	11.0	1,056,080.2	1,106,008.6	11	4	0	2
a280-81-c2-w30-500-1000	171.7	1,107,899.8	1,107,899.8	11	5	0	1
a280-91-c2-w30-500-1000	249.7	1,284,756.7	1,309,579.4	13	4	38	26
a280-101-c2-w30-500-1000	276.5	1,360,302.8	1,410,052.2	14	5	16	5
a280-111-c2-w30-500-1000							
a280-121-c2-w30-500-1000							
a280-131-c2-w30-500-1000							
a280-141-c2-w30-500-1000							
a280-151-c2-w30-500-1000							
a280-51-c2-w30-1000-1200	0.5	704,314.8	704,324.1	7	3	3	2
a280-61-c2-w30-1000-1200	4.7	655,125.4	704,687.0	7	3	18	5
a280-71-c2-w30-1000-1200	12.5	906,049.3	906,094.7	9	4	9	2
a280-81-c2-w30-1000-1200	22.7	1,004,991.1	1,008,098.4	10	4	10	4
a280-91-c2-w30-1000-1200	129.7	1,168,697.1	1,208,357.5	12	4	19	15
a280-101-c2-w30-1000-1200	103.0	1,210,282.4	1,210,282.4	12	4	0	1
a280-111-c2-w30-1000-1200	4,133.4	1,276,467.0	1,311,048.2	13	3	107	221
a280-121-c2-w30-1000-1200							
a280-131-c2-w30-1000-1200	3,861.4	1,221,912.7	1,310,717.5	13	5	64	27
a280-141-c2-w30-1000-1200	1,367.8	1,485,128.7	1,512,917.1	15	4	5	3
a280-151-c2-w30-1000-1200							
a280-51-c2-w30-1500-2000	14.5	503,907.4	503,914.7	5	4	2	2
a280-61-c2-w30-1500-2000	11.9	505,504.5	505,504.5	5	4	0	1
a280-71-c2-w30-1500-2000	78.1	756,981.7	805,654.8	8	4	52	4
a280-81-c2-w30-1500-2000	84.9	700,065.3	707,568.9	7	4	30	4
a280-91-c2-w30-1500-2000	1,943.6	723,751.3	807,307.3	8	4	68	50
a280-101-c2-w30-1500-2000							
a280-111-c2-w30-1500-2000	6,330.7	909,872.6	910,183.0	9	4	75	43
a280-121-c2-w30-1500-2000							
a280-131-c2-w30-1500-2000							
a280-141-c2-w30-1500-2000							
a280-151-c2-w30-1500-2000							

Table 6.9 – Computational results for the variant with two stacks for instances in class C2 *(cont'd and end)*

Instance	Sec.	$\underline{z}$	z^*	Veh.	maxItem	OC	B
a280-51-c2-w45-500-1000	29.2	644,309.0	703,927.4	7	4	11	5
a280-61-c2-w45-500-1000	495.7	721,863.2	804,639.2	8	4	20	4
a280-71-c2-w45-500-1000	83.8	806,421.5	806,421.5	8	5	0	1
a280-81-c2-w45-500-1000	1,070.1	958,832.5	1,007,186.1	10	4	62	43
a280-91-c2-w45-500-1000							
a280-101-c2-w45-500-1000							
a280-111-c2-w45-500-1000							
a280-121-c2-w45-500-1000							
a280-131-c2-w45-500-1000							
a280-141-c2-w45-500-1000							
a280-151-c2-w45-500-1000							
a280-51-c2-w45-1000-1200	0.9	703,894.7	703,894.7	7	4	0	1
a280-61-c2-w45-1000-1200	44.5	654,455.1	704,350.5	7	4	52	7
a280-71-c2-w45-1000-1200	27.7	705,805.6	705,809.2	7	4	1	2
a280-81-c2-w45-1000-1200	114.7	796,563.7	806,929.8	8	4	20	3
a280-91-c2-w45-1000-1200	427.1	983,156.6	1,007,973.5	10	4	54	15
a280-101-c2-w45-1000-1200	5,607.2	1,137,055.0	1,209,618.5	12	4	98	82
a280-111-c2-w45-1000-1200							
a280-121-c2-w45-1000-1200							
a280-131-c2-w45-1000-1200							
a280-141-c2-w45-1000-1200							
a280-151-c2-w45-1000-1200							
a280-51-c2-w45-1500-2000	53.0	479,238.6	504,155.2	5	5	61	22
a280-61-c2-w45-1500-2000	89.2	704,556.5	704,677.8	7	4	50	20
a280-71-c2-w45-1500-2000	1,644.7	655,713.5	705,136.8	7	5	62	10
a280-81-c2-w45-1500-2000	2,734.5	647,827.7	706,828.1	7	5	112	103
a280-91-c2-w45-1500-2000	293.3	809,319.0	809,325.4	8	4	3	2
a280-101-c2-w45-1500-2000							
a280-111-c2-w45-1500-2000	2,592.8	909,438.6	909,514.5	9	4	52	10
a280-121-c2-w45-1500-2000							
a280-131-c2-w45-1500-2000							
a280-141-c2-w45-1500-2000							
a280-151-c2-w45-1500-2000							

Table 6.10 – Computational results for the variant with three stacks for instances in class C1 *(cont'd)*

Instance	Sec.	$\underline{z}$	z^*	Veh.	maxItem	OC	B
a280-51-c1-w15-500-1000	0.1	771,396.7	804,243.4	8	3	0	2
a280-61-c1-w15-500-1000	0.5	1,038,700.9	1,105,192.8	11	3	3	3
a280-71-c1-w15-500-1000	8.5	1,073,706.8	1,107,074.8	11	3	17	14
a280-81-c1-w15-500-1000	1.1	1,274,987.6	1,308,179.7	13	3	1	3
a280-91-c1-w15-500-1000	37.3	1,342,754.9	1,409,233.6	14	3	28	30
a280-101-c1-w15-500-1000	11.1	1,644,142.4	1,710,847.6	17	3	20	8
a280-111-c1-w15-500-1000	172.8	1,695,055.5	1,711,766.3	17	3	37	78
a280-121-c1-w15-500-1000	18.3	1,946,387.9	2,012,695.1	20	3	24	5
a280-131-c1-w15-500-1000	2,576.7	1,946,173.5	2,012,928.2	20	3	166	366
a280-141-c1-w15-500-1000	257.9	2,247,860.3	2,314,592.9	23	3	41	55
a280-151-c1-w15-500-1000	152.8	2,181,145.6	2,214,373.5	22	3	26	19
a280-51-c1-w15-1000-1200	4.2	637,638.0	704,178.9	7	3	20	28
a280-61-c1-w15-1000-1200	0.2	805,245.0	805,245.0	8	3	0	1
a280-71-c1-w15-1000-1200	0.6	892,260.1	906,384.6	9	3	0	2
a280-81-c1-w15-1000-1200	9.1	1,051,503.3	1,108,123.3	11	3	23	7
a280-91-c1-w15-1000-1200	19.1	1,242,724.8	1,309,120.9	13	3	27	10
a280-101-c1-w15-1000-1200	103.2	1,148,591.6	1,209,326.8	12	3	50	35
a280-111-c1-w15-1000-1200	238.6	1,344,351.1	1,411,124.3	14	3	60	46
a280-121-c1-w15-1000-1200	46.0	1,478,511.8	1,511,805.5	15	3	28	7
a280-131-c1-w15-1000-1200	1,719.5	1,396,817.3	1,412,334.7	14	3	80	158
a280-141-c1-w15-1000-1200	1,292.1	1,594,751.3	1,613,862.8	16	3	74	113
a280-151-c1-w15-1000-1200	1,980.0	1,801,830.6	1,814,289.0	18	3	42	202
a280-51-c1-w15-1500-2000	3.1	637,223.1	703,938.1	7	3	10	11
a280-61-c1-w15-1500-2000	21.8	738,422.2	805,122.9	8	3	24	36
a280-71-c1-w15-1500-2000	14.3	739,172.5	805,594.2	8	3	48	9
a280-81-c1-w15-1500-2000	41.5	807,266.7	807,350.6	8	3	54	15
a280-91-c1-w15-1500-2000	107.4	1,107,525.7	1,107,577.8	11	3	76	26
a280-101-c1-w15-1500-2000	424.0	1,042,218.4	1,108,513.7	11	3	91	104
a280-111-c1-w15-1500-2000	384.8	1,176,880.3	1,210,071.7	12	3	68	55
a280-121-c1-w15-1500-2000	1,917.6	1,343,986.4	1,410,583.7	14	3	81	122
a280-131-c1-w15-1500-2000	1,025.5	1,377,192.8	1,410,572.2	14	3	61	41
a280-141-c1-w15-1500-2000	4,507.6	1,444,926.8	1,511,435.5	15	3	101	231
a280-151-c1-w15-1500-2000							

Table 6.10 – Computational results for the variant with three stacks for instances in class C1 *(cont'd)*

Instance	Sec.	$\underline{z}$	z^*	Veh.	maxItem	OC	B
a280-51-c1-w30-500-1000	7.1	703,832.1	703,905.1	7	3	17	7
a280-61-c1-w30-500-1000	8.3	871,468.7	904,887.6	9	3	47	10
a280-71-c1-w30-500-1000	0.7	1,013,737.7	1,106,203.5	11	3	0	2
a280-81-c1-w30-500-1000	1.1	1,274,842.3	1,308,059.0	13	3	0	2
a280-91-c1-w30-500-1000	69.4	1,342,490.8	1,409,180.7	14	3	40	24
a280-101-c1-w30-500-1000	14.8	1,577,926.0	1,611,012.6	16	3	23	5
a280-111-c1-w30-500-1000	66.3	1,677,678.4	1,710,981.0	17	3	41	14
a280-121-c1-w30-500-1000	1,701.3	1,945,921.7	2,012,740.7	20	3	172	236
a280-131-c1-w30-500-1000	27.2	2,012,655.2	2,012,673.4	20	3	6	3
a280-141-c1-w30-500-1000	245.0	2,280,698.1	2,313,981.9	23	3	23	16
a280-151-c1-w30-500-1000	571.0	2,214,890.5	2,214,964.9	22	3	65	38
a280-51-c1-w30-1000-1200	2.6	670,549.8	703,829.2	7	3	5	7
a280-61-c1-w30-1000-1200	1.6	705,245.5	705,282.5	7	3	10	2
a280-71-c1-w30-1000-1200	53.5	872,543.4	905,768.0	9	3	44	27
a280-81-c1-w30-1000-1200	13.0	907,210.3	907,261.1	9	3	10	5
a280-91-c1-w30-1000-1200	18.8	1,075,040.8	1,108,303.0	11	3	26	5
a280-101-c1-w30-1000-1200	75.7	1,065,388.4	1,109,516.5	11	3	54	9
a280-111-c1-w30-1000-1200	507.4	1,250,937.6	1,310,816.8	13	3	53	71
a280-121-c1-w30-1000-1200	1,542.5	1,378,738.1	1,411,791.0	14	3	69	106
a280-131-c1-w30-1000-1200							
a280-141-c1-w30-1000-1200	2,823.6	1,412,967.4	1,413,207.4	14	3	78	132
a280-151-c1-w30-1000-1200							
a280-51-c1-w30-1500-2000	0.3	504,074.5	504,074.5	5	3	0	1
a280-61-c1-w30-1500-2000	17.4	704,435.2	704,468.8	7	3	50	12
a280-71-c1-w30-1500-2000	169.2	772,317.7	805,558.0	8	3	78	51
a280-81-c1-w30-1500-2000	2.6	806,556.9	806,556.9	8	3	0	1
a280-91-c1-w30-1500-2000	87.6	1,041,509.9	1,108,169.4	11	3	31	15
a280-101-c1-w30-1500-2000	370.9	1,109,456.0	1,109,540.4	11	3	92	52
a280-111-c1-w30-1500-2000	1,543.6	1,109,733.2	1,109,871.9	11	3	96	156
a280-121-c1-w30-1500-2000	5,866.3	1,277,203.7	1,310,533.8	13	3	133	326
a280-131-c1-w30-1500-2000	866.1	1,377,474.0	1,410,762.4	14	3	51	24
a280-141-c1-w30-1500-2000							
a280-151-c1-w30-1500-2000	6,923.6	1,579,566.0	1,612,873.2	16	3	76	205

Table 6.10 – Computational results for the variant with three stacks for instances in class C1 *(cont'd and end)*

Instance	Sec.	$\underline{z}$	z^*	Veh.	maxItem	OC	B
a280-51-c1-w45-500-1000	3.9	670,998.4	704,064.9	7	3	7	5
a280-61-c1-w45-500-1000	0.5	905,467.5	905,467.5	9	3	0	1
a280-71-c1-w45-500-1000	7.4	972,894.5	1,006,087.9	10	3	10	4
a280-81-c1-w45-500-1000	793.0	1,274,267.2	1,307,675.7	13	3	211	201
a280-91-c1-w45-500-1000	175.0	1,408,377.3	1,408,428.5	14	3	63	29
a280-101-c1-w45-500-1000	257.2	1,576,915.6	1,610,308.2	16	3	44	54
a280-111-c1-w45-500-1000	439.2	1,677,230.6	1,710,508.8	17	3	41	32
a280-121-c1-w45-500-1000	360.3	1,744,599.4	1,811,360.8	18	3	47	18
a280-131-c1-w45-500-1000	55.6	1,912,004.9	1,912,031.6	19	3	25	4
a280-141-c1-w45-500-1000	3,685.7	2,080,515.1	2,113,707.4	21	3	150	195
a280-151-c1-w45-500-1000	197.0	2,181,064.3	2,214,374.5	22	3	32	9
a280-51-c1-w45-1000-1200	1.0	537,064.6	603,479.8	6	3	0	2
a280-61-c1-w45-1000-1200	11.9	616,449.2	704,077.5	7	3	42	5
a280-71-c1-w45-1000-1200	134.6	780,661.8	805,654.4	8	3	81	33
a280-81-c1-w45-1000-1200	332.8	906,496.2	906,578.9	9	3	61	44
a280-91-c1-w45-1000-1200	423.3	923,007.9	1,007,429.4	10	3	79	54
a280-101-c1-w45-1000-1200	566.1	1,142,730.9	1,209,398.5	12	3	50	60
a280-111-c1-w45-1000-1200							
a280-121-c1-w45-1000-1200	275.3	1,243,203.9	1,309,712.6	13	3	54	8
a280-131-c1-w45-1000-1200							
a280-141-c1-w45-1000-1200	110.3	1,444,918.3	1,511,335.9	15	3	5	3
a280-151-c1-w45-1000-1200	552.9	1,713,169.7	1,713,257.0	17	3	53	14
a280-51-c1-w45-1500-2000	0.7	403,943.5	403,943.5	4	3	0	1
a280-61-c1-w45-1500-2000	30.6	670,852.7	704,214.6	7	3	47	12
a280-71-c1-w45-1500-2000	33.1	805,772.5	805,819.7	8	3	43	8
a280-81-c1-w45-1500-2000	2,118.7	739,374.8	806,155.9	8	3	179	439
a280-91-c1-w45-1500-2000	74.9	807,624.2	807,681.5	8	3	57	7
a280-101-c1-w45-1500-2000	550.0	1,075,475.0	1,108,801.1	11	3	54	38
a280-111-c1-w45-1500-2000							
a280-121-c1-w45-1500-2000							
a280-131-c1-w45-1500-2000							
a280-141-c1-w45-1500-2000	4,690.5	1,411,266.7	1,411,337.1	14	3	64	103
a280-151-c1-w45-1500-2000							

Table 6.11 – Computational results for the variant with three stacks for instances in class C2 *(cont'd)*

Instance	Sec.	$\underline{z}$	z^*	Veh.	maxItem	OC	B
a280-51-c2-w15-500-1000	1.9	704,424.7	704,424.7	7	4	0	1
a280-61-c2-w15-500-1000	5.7	865,165.9	905,043.8	9	4	0	2
a280-71-c2-w15-500-1000	94.8	905,863.5	905,917.6	9	4	30	15
a280-81-c2-w15-500-1000	114.9	1,245,390.9	1,307,786.2	13	5	38	20
a280-91-c2-w15-500-1000	1,148.8	1,258,444.9	1,308,336.8	13	5	34	40
a280-101-c2-w15-500-1000	387.4	1,399,688.1	1,510,667.3	15	5	34	27
a280-111-c2-w15-500-1000	118.9	1,712,128.8	1,712,128.8	17	4	0	1
a280-121-c2-w15-500-1000	4,713.0	1,523,939.3	1,611,521.1	16	5	28	8
a280-131-c2-w15-500-1000	4,687.2	1,680,142.8	1,712,233.6	17	5	47	10
a280-141-c2-w15-500-1000	1,810.9	1,860,423.0	1,913,410.7	19	5	56	7
a280-151-c2-w15-500-1000							
a280-51-c2-w15-1000-1200	3.1	754,574.8	804,354.3	8	3	3	4
a280-61-c2-w15-1000-1200	3.9	771,728.7	804,876.3	8	4	0	2
a280-71-c2-w15-1000-1200	8.5	906,789.5	906,910.3	9	4	5	3
a280-81-c2-w15-1000-1200	33.4	1,132,702.8	1,207,431.8	12	4	10	4
a280-91-c2-w15-1000-1200	47.0	1,115,772.0	1,208,755.2	12	5	10	5
a280-101-c2-w15-1000-1200	994.2	1,134,718.4	1,209,308.8	12	5	28	18
a280-111-c2-w15-1000-1200	1,567.8	1,159,692.0	1,210,475.7	12	5	82	53
a280-121-c2-w15-1000-1200							
a280-131-c2-w15-1000-1200							
a280-141-c2-w15-1000-1200	3,154.4	1,708,632.4	1,713,062.9	17	4	12	3
a280-151-c2-w15-1000-1200							
a280-51-c2-w15-1500-2000	9.2	503,814.4	503,814.4	5	5	0	1
a280-61-c2-w15-1500-2000	8.9	585,037.5	604,889.9	6	5	0	2
a280-71-c2-w15-1500-2000	761.2	738,758.9	805,232.7	8	5	43	12
a280-81-c2-w15-1500-2000	6,018.1	873,171.3	906,400.1	9	5	19	7
a280-91-c2-w15-1500-2000	2,026.0	790,680.0	808,044.5	8	5	58	8
a280-101-c2-w15-1500-2000							
a280-111-c2-w15-1500-2000							
a280-121-c2-w15-1500-2000							
a280-131-c2-w15-1500-2000							
a280-141-c2-w15-1500-2000							
a280-151-c2-w15-1500-2000							

Table 6.11 – Computational results for the variant with three stacks for instances in class C2 *(cont'd)*

Instance	Sec.	$\underline{z}$	z^*	Veh.	maxItem	OC	B
a280-51-c2-w30-500-1000	4.2	604,267.7	604,267.7	6	4	0	1
a280-61-c2-w30-500-1000	148.3	755,006.6	804,538.2	8	5	9	3
a280-71-c2-w30-500-1000	142.6	872,630.9	905,705.0	9	5	34	15
a280-81-c2-w30-500-1000							
a280-91-c2-w30-500-1000	299.1	1,129,789.1	1,208,022.4	12	5	29	10
a280-101-c2-w30-500-1000	3,596.1	1,234,939.8	1,309,393.0	13	5	64	16
a280-111-c2-w30-500-1000							
a280-121-c2-w30-500-1000							
a280-131-c2-w30-500-1000							
a280-141-c2-w30-500-1000							
a280-151-c2-w30-500-1000							
a280-51-c2-w30-1000-1200	0.7	704,314.8	704,324.1	7	3	3	2
a280-61-c2-w30-1000-1200	2.7	618,378.3	704,338.5	7	4	0	2
a280-71-c2-w30-1000-1200	262.3	791,007.5	806,080.0	8	5	51	28
a280-81-c2-w30-1000-1200	61.3	969,668.5	1,006,878.1	10	4	14	4
a280-91-c2-w30-1000-1200	557.3	1,051,651.3	1,107,908.5	11	4	43	20
a280-101-c2-w30-1000-1200	1,165.2	1,077,967.4	1,109,227.0	11	4	33	7
a280-111-c2-w30-1000-1200	3,727.0	1,210,422.9	1,210,549.7	12	4	43	41
a280-121-c2-w30-1000-1200							
a280-131-c2-w30-1000-1200							
a280-141-c2-w30-1000-1200							
a280-151-c2-w30-1000-1200							
a280-51-c2-w30-1500-2000	82.3	437,036.0	503,573.7	5	6	14	4
a280-61-c2-w30-1500-2000	105.5	504,662.3	504,708.6	5	4	40	3
a280-71-c2-w30-1500-2000	600.0	672,606.3	705,468.4	7	4	64	13
a280-81-c2-w30-1500-2000	1,428.1	641,417.2	706,655.6	7	5	52	20
a280-91-c2-w30-1500-2000							
a280-101-c2-w30-1500-2000							
a280-111-c2-w30-1500-2000							
a280-121-c2-w30-1500-2000							
a280-131-c2-w30-1500-2000							
a280-141-c2-w30-1500-2000							
a280-151-c2-w30-1500-2000							

Table 6.11 – Computational results for the variant with three stacks for instances in class C2 *(cont'd and end)*

Instance	Sec.	$\underline{z}$	z^*	Veh.	maxItem	OC	B
a280-51-c2-w45-500-1000	39.4	604,419.3	604,419.3	6	4	0	1
a280-61-c2-w45-500-1000	2,702.7	677,875.0	704,438.8	7	5	40	4
a280-71-c2-w45-500-1000	3,554.5	752,402.1	805,445.0	8	6	44	17
a280-81-c2-w45-500-1000	5,837.3	887,015.6	907,002.7	9	5	62	81
a280-91-c2-w45-500-1000							
a280-101-c2-w45-500-1000							
a280-111-c2-w45-500-1000							
a280-121-c2-w45-500-1000							
a280-131-c2-w45-500-1000							
a280-141-c2-w45-500-1000							
a280-151-c2-w45-500-1000							
a280-51-c2-w45-1000-1200	5.4	703,866.0	703,880.1	7	4	10	2
a280-61-c2-w45-1000-1200	34.5	554,048.8	603,773.1	6	4	0	2
a280-71-c2-w45-1000-1200	86.3	671,978.4	705,150.7	7	5	21	4
a280-81-c2-w45-1000-1200	837.1	777,024.2	806,567.4	8	4	48	11
a280-91-c2-w45-1000-1200	473.1	891,572.5	907,833.8	9	4	51	6
a280-101-c2-w45-1000-1200							
a280-111-c2-w45-1000-1200							
a280-121-c2-w45-1000-1200							
a280-131-c2-w45-1000-1200							
a280-141-c2-w45-1000-1200							
a280-151-c2-w45-1000-1200							
a280-51-c2-w45-1500-2000	20.3	470,300.9	503,496.4	5	4	0	2
a280-61-c2-w45-1500-2000	48.6	537,642.8	604,088.8	6	5	32	4
a280-71-c2-w45-1500-2000							
a280-81-c2-w45-1500-2000	3,483.0	606,788.8	607,006.4	6	5	50	20
a280-91-c2-w45-1500-2000	4,311.8	773,673.9	806,792.8	8	4	61	8
a280-101-c2-w45-1500-2000							
a280-111-c2-w45-1500-2000							
a280-121-c2-w45-1500-2000							
a280-131-c2-w45-1500-2000							
a280-141-c2-w45-1500-2000							
a280-151-c2-w45-1500-2000							

Appendix B. Detailed Results on the Impact of the Number of Stacks

This appendix presents detailed computational results to assess the impact of the number of stacks. Tables 6.12 and 6.13 present the impact of the number of stacks on the computational results for instances in class C1 and C2, respectively. For each table, we compare the results obtained with one stack to the results obtained with two and three stacks. We present the following information : Δ *Dist (%)*, the impact in percentage on the total traveled distance, computed as $(Dist_2 - Dist_1)/(Dist_1)$ and $(Dist_3 - Dist_1)/(Dist_1)$, where $Dist_j, j = \{1, 2, 3\}$, is the distance with j stacks, respectively ; Δ *Veh (%)*, the impact in percentage on the number of vehicles, computed as $(Veh_2 - Veh_1)/(Veh_1)$ and $(Veh_3 - Veh_1)/(Veh_1)$, where $Veh_j, j = \{1, 2, 3\}$, is the number of vehicles with j stacks, respectively ; and Δ *maxItem (%)*, the impact in percentage on the maximum number of items simultaneously in a vehicle, computed as $(maxItem_2 - maxItem_1)/(maxItem_1)$ and $(maxItem_3 - maxItem_1)/(maxItem_1)$, where $maxItem_j, j = \{1, 2, 3\}$, is the maximum number of items simultaneously in a vehicle with j stacks.

Table 6.12 – Impact of the number of stacks on the results for instances in class C1 *(cont'd)*

	1 stack VS 2 stacks			1 stack VS 3 stacks		
Instance	Δ Dist (%)	Δ Veh (%)	Δ maxItem (%)	Δ Dist (%)	Δ Veh (%)	Δ maxItem (%)
a280-51-c1-w15-500-1000	–26.7	–36.4	25.0	–24.9	–27.3	–25.0
a280-61-c1-w15-500-1000	–38.4	–44.4	66.7	–37.3	–38.9	0.0
a280-71-c1-w15-500-1000	–25.1	–35.3	33.3	–21.9	–35.3	0.0
a280-81-c1-w15-500-1000	–34.1	–45.5	100.0	–32.4	–40.9	0.0
a280-91-c1-w15-500-1000	–32.6	–48.0	0.0	–34.5	–44.0	–25.0
a280-101-c1-w15-500-1000	–35.3	–44.0	66.7	–27.0	–32.0	0.0
a280-111-c1-w15-500-1000	–31.8	–36.0	25.0	–28.5	–32.0	–25.0
a280-121-c1-w15-500-1000				–29.3	–31.0	–25.0
a280-131-c1-w15-500-1000				–25.1	–31.0	0.0
a280-141-c1-w15-500-1000				–26.6	–32.4	–25.0
a280-151-c1-w15-500-1000				–23.2	–29.0	–40.0
a280-51-c1-w15-1000-1200	–28.6	–36.4	0.0	–29.6	–36.4	0.0
a280-61-c1-w15-1000-1200	–28.2	–33.3	66.7	–28.6	–33.3	0.0
a280-71-c1-w15-1000-1200	–25.0	–30.8	33.3	–22.3	–30.8	0.0
a280-81-c1-w15-1000-1200				–15.6	–26.7	0.0
a280-91-c1-w15-1000-1200	–25.3	–42.9	25.0	–25.2	–38.1	–25.0
a280-101-c1-w15-1000-1200				–29.7	–29.4	–25.0
a280-111-c1-w15-1000-1200	–22.4	–38.1	33.3	–20.2	–33.3	0.0
a280-121-c1-w15-1000-1200				–23.4	–34.8	–25.0
a280-131-c1-w15-1000-1200				–12.9	–33.3	–25.0
a280-141-c1-w15-1000-1200				–29.3	–44.8	0.0
a280-151-c1-w15-1000-1200				–19.8	–35.7	0.0

Table 6.12 – Impact of the number of stacks on the results for instances in class C1 *(cont'd)*

	1 stack VS 2 stacks			1 stack VS 3 stacks		
Instance	Δ Dist (%)	Δ Veh (%)	Δ maxItem (%)	Δ Dist (%)	Δ Veh (%)	Δ maxItem (%)
a280-51-c1-w15-1500-2000	–31.8	–45.5	33.3	–33.1	–36.4	0.0
a280-61-c1-w15-1500-2000	–31.7	–45.5	25.0	–26.3	–27.3	–25.0
a280-71-c1-w15-1500-2000				–16.1	–20.0	–40.0
a280-81-c1-w15-1500-2000				–16.9	–20.0	–40.0
a280-91-c1-w15-1500-2000				–22.8	–21.4	–25.0
a280-101-c1-w15-1500-2000				–22.1	–26.7	–25.0
a280-111-c1-w15-1500-2000				–22.7	–33.3	–25.0
a280-121-c1-w15-1500-2000						
a280-131-c1-w15-1500-2000						
a280-141-c1-w15-1500-2000						
a280-151-c1-w15-1500-2000						
a280-51-c1-w30-500-1000	–24.0	–40.0	25.0	–16.4	–30.0	–25.0
a280-61-c1-w30-500-1000	–36.0	–42.9	66.7	–34.4	–35.7	0.0
a280-71-c1-w30-500-1000	–30.2	–37.5	0.0	–26.7	–31.3	–25.0
a280-81-c1-w30-500-1000				–22.5	–23.5	0.0
a280-91-c1-w30-500-1000				–18.4	–22.2	–25.0
a280-101-c1-w30-500-1000				–20.0	–27.3	–25.0
a280-111-c1-w30-500-1000				–29.0	–32.0	–25.0
a280-121-c1-w30-500-1000				–19.6	–20.0	–25.0
a280-131-c1-w30-500-1000				–19.3	–16.7	–25.0
a280-141-c1-w30-500-1000						
a280-151-c1-w30-500-1000				–19.6	–24.1	–25.0

Table 6.12 – Impact of the number of stacks on the results for instances in class C1 *(cont'd)*

	1 stack VS 2 stacks			1 stack VS 3 stacks		
Instance	Δ Dist (%)	Δ Veh (%)	Δ maxItem (%)	Δ Dist (%)	Δ Veh (%)	Δ maxItem (%)
a280-51-c1-w30-1000-1200	–25.2	–30.0	25.0	–22.9	–30.0	–25.0
a280-61-c1-w30-1000-1200	–25.6	–41.7	66.7	–16.9	–41.7	0.0
a280-71-c1-w30-1000-1200				–22.3	–30.8	–25.0
a280-81-c1-w30-1000-1200				–20.4	–35.7	–25.0
a280-91-c1-w30-1000-1200				–14.9	–26.7	–40.0
a280-101-c1-w30-1000-1200	–24.3	–31.3	25.0	–21.9	–31.3	-25.0
a280-111-c1-w30-1000-1200				–21.0	–31.6	–25.0
a280-121-c1-w30-1000-1200				–25.7	–36.4	–25.0
a280-131-c1-w30-1000-1200						
a280-141-c1-w30-1000-1200				–15.1	–36.4	–25.0
a280-151-c1-w30-1000-1200						
a280-51-c1-w30-1500-2000	–30.0	–37.5	50.0	–14.2	–37.5	–25.0
a280-61-c1-w30-1500-2000				–18.4	–12.5	–25.0
a280-71-c1-w30-1500-2000				–24.6	–38.5	0.0
a280-81-c1-w30-1500-2000				–21.1	–33.3	–25.0
a280-91-c1-w30-1500-2000				–17.8	–15.4	–40.0
a280-101-c1-w30-1500-2000				–15.2	–21.4	–25.0
a280-111-c1-w30-1500-2000				–21.2	–26.7	–40.0
a280-121-c1-w30-1500-2000				–20.8	–27.8	–40.0
a280-131-c1-w30-1500-2000						
a280-141-c1-w30-1500-2000						
a280-151-c1-w30-1500-2000						

Table 6.12 – Impact of the number of stacks on the results for instances in class C1 *(cont'd)*

	1 stack VS 2 stacks			1 stack VS 3 stacks		
Instance	Δ Dist (%)	Δ Veh (%)	Δ maxItem (%)	Δ Dist (%)	Δ Veh (%)	Δ maxItem (%)
a280-51-c1-w45-500-1000	–27.9	–36.4	0.0	–28.6	–36.4	–25.0
a280-61-c1-w45-500-1000	–26.3	–38.5	25.0	–20.2	–30.8	–25.0
a280-71-c1-w45-500-1000				–14.0	–9.1	–40.0
a280-81-c1-w45-500-1000				–16.3	–13.3	–25.0
a280-91-c1-w45-500-1000				–14.8	–12.5	–40.0
a280-101-c1-w45-500-1000				–19.5	–20.0	–40.0
a280-111-c1-w45-500-1000				–11.6	–10.5	–40.0
a280-121-c1-w45-500-1000						
a280-131-c1-w45-500-1000						
a280-141-c1-w45-500-1000						
a280-151-c1-w45-500-1000						
a280-51-c1-w45-1000-1200	–26.8	–37.5	25.0	–25.2	–25.0	–25.0
a280-61-c1-w45-1000-1200	–27.8	–33.3	50.0	–25.1	–22.2	–25.0
a280-71-c1-w45-1000-1200	–26.4	–33.3	25.0	–23.2	–33.3	–25.0
a280-81-c1-w45-1000-1200				–27.4	–35.7	–25.0
a280-91-c1-w45-1000-1200				–24.3	–33.3	–25.0
a280-101-c1-w45-1000-1200				–20.4	–29.4	–25.0
a280-111-c1-w45-1000-1200						
a280-121-c1-w45-1000-1200						
a280-131-c1-w45-1000-1200						
a280-141-c1-w45-1000-1200						
a280-151-c1-w45-1000-1200						

Table 6.12 – Impact of the number of stacks on the results for instances in class C1 *(cont'd and end)*

	1 stack VS 2 stacks			1 stack VS 3 stacks		
Instance	Δ Dist (%)	Δ Veh (%)	Δ maxItem (%)	Δ Dist (%)	Δ Veh (%)	Δ maxItem (%)
a280-51-c1-w45-1500-2000	–23.9	–50.0	33.3	–23.3	–50.0	0.0
a280-61-c1-w45-1500-2000				–31.5	–30.0	–40.0
a280-71-c1-w45-1500-2000				–19.2	–20.0	–40.0
a280-81-c1-w45-1500-2000				–29.5	–38.5	–25.0
a280-91-c1-w45-1500-2000				–22.5	–33.3	–25.0
a280-101-c1-w45-1500-2000				–24.8	–21.4	–25.0
a280-111-c1-w45-1500-2000						
a280-121-c1-w45-1500-2000						
a280-131-c1-w45-1500-2000						
a280-141-c1-w45-1500-2000						
a280-151-c1-w45-1500-2000						
Average	–28.6	–39.0	35.2	–22.8	–29.6	–21.6

Table 6.13 – Impact of the number of stacks on the results for instances in class C2 *(cont'd)*

	1 stack VS 2 stacks			1 stack VS 3 stacks		
Instance	Δ Dist (%)	Δ Veh (%)	Δ maxItem (%)	Δ Dist (%)	Δ Veh (%)	Δ maxItem (%)
a280-51-c2-w15-500-1000	–31.6	–46.7	33.3	–32.8	–53.3	33.3
a280-61-c2-w15-500-1000	–25.8	–40.0	33.3	–29.6	–40.0	33.3
a280-71-c2-w15-500-1000	–26.2	–33.3	33.3	–30.9	–40.0	33.3
a280-81-c2-w15-500-1000	–21.8	–27.8	25.0	–20.8	–27.8	25.0
a280-91-c2-w15-500-1000	–24.2	–31.6	66.7	–25.3	–31.6	66.7
a280-101-c2-w15-500-1000	–27.0	–40.7	33.3	–34.1	–44.4	66.7
a280-111-c2-w15-500-1000	–25.1	–35.7	33.3	–29.0	–39.3	33.3
a280-121-c2-w15-500-1000	–25.9	–34.6	0.0	–31.0	–38.5	25.0
a280-131-c2-w15-500-1000	–29.4	–47.1	66.7	–34.8	–50.0	66.7
a280-141-c2-w15-500-1000	–24.0	–35.5	33.3	–27.9	–38.7	66.7
a280-151-c2-w15-500-1000	–27.0	–40.5	0.0			
a280-51-c2-w15-1000-1200	–24.2	–25.0	0.0	–28.5	–33.3	0.0
a280-61-c2-w15-1000-1200	–34.5	–43.8	0.0	–39.8	–50.0	33.3
a280-71-c2-w15-1000-1200	–23.3	–28.6	0.0	–9.3	–35.7	33.3
a280-81-c2-w15-1000-1200	–27.8	–33.3	33.3	–33.5	–33.3	33.3
a280-91-c2-w15-1000-1200	–19.8	–33.3	0.0	–24.3	–33.3	25.0
a280-101-c2-w15-1000-1200	–17.4	–23.5	0.0	–24.9	–29.4	25.0
a280-111-c2-w15-1000-1200	–17.0	–31.6	33.3	–21.9	–36.8	66.7
a280-121-c2-w15-1000-1200	–21.3	–33.3	0.0			
a280-131-c2-w15-1000-1200						
a280-141-c2-w15-1000-1200	–23.5	–36.7	33.3	–27.1	–43.3	33.3
a280-151-c2-w15-1000-1200						

Table 6.13 – Impact of the number of stacks on the results for instances in class C2 *(cont'd)*

	1 stack VS 2 stacks			1 stack VS 3 stacks		
Instance	Δ Dist (%)	Δ Veh (%)	Δ maxItem (%)	Δ Dist (%)	Δ Veh (%)	Δ maxItem (%)
a280-51-c2-w15-1500-2000	–28.1	–45.5	33.3	–31.4	–54.5	66.7
a280-61-c2-w15-1500-2000	–10.7	–40.0	0.0	–21.6	–40.0	25.0
a280-71-c2-w15-1500-2000	–31.1	–33.3	0.0	–37.6	–33.3	25.0
a280-81-c2-w15-1500-2000	–32.0	–43.8	33.3	–32.6	–43.8	66.7
a280-91-c2-w15-1500-2000	–13.9	–30.8	0.0	–16.7	–38.5	25.0
a280-101-c2-w15-1500-2000	–16.0	–40.0	0.0			
a280-111-c2-w15-1500-2000						
a280-121-c2-w15-1500-2000						
a280-131-c2-w15-1500-2000						
a280-141-c2-w15-1500-2000						
a280-151-c2-w15-1500-2000						
a280-51-c2-w30-500-1000	–32.6	–46.2	33.3	–33.4	–53.8	33.3
a280-61-c2-w30-500-1000	–31.1	–43.8	66.7	–39.4	–50.0	66.7
a280-71-c2-w30-500-1000	–34.5	–38.9	0.0	–37.8	–50.0	25.0
a280-81-c2-w30-500-1000	–24.5	–42.1	25.0			
a280-91-c2-w30-500-1000	–25.9	–40.9	0.0	–37.9	–45.5	25.0
a280-101-c2-w30-500-1000	–29.7	–41.7	0.0	–34.3	–45.8	0.0
a280-111-c2-w30-500-1000						
a280-121-c2-w30-500-1000						
a280-131-c2-w30-500-1000						
a280-141-c2-w30-500-1000						
a280-151-c2-w30-500-1000						

Table 6.13 – Impact of the number of stacks on the results for instances in class C2 *(cont'd)*

	1 stack VS 2 stacks			1 stack VS 3 stacks		
Instance	Δ Dist (%)	Δ Veh (%)	Δ maxItem (%)	Δ Dist (%)	Δ Veh (%)	Δ maxItem (%)
a280-51-c2-w30-1000-1200	–7.0	–22.2	0.0	–7.0	–22.2	0.0
a280-61-c2-w30-1000-1200	–13.8	–30.0	0.0	–20.2	–30.0	33.3
a280-71-c2-w30-1000-1200	–18.8	–35.7	–20.0	–19.0	–42.9	0.0
a280-81-c2-w30-1000-1200	–14.7	–28.6	0.0	–27.6	–28.6	0.0
a280-91-c2-w30-1000-1200	–25.0	–33.3	33.3	–29.0	–38.9	33.3
a280-101-c2-w30-1000-1200	–24.6	–40.0	33.3	–32.3	–45.0	33.3
a280-111-c2-w30-1000-1200	–16.6	–31.6	–25.0	–20.3	–36.8	0.0
a280-121-c2-w30-1000-1200						
a280-131-c2-w30-1000-1200	–22.9	–35.0	25.0			
a280-141-c2-w30-1000-1200	–22.7	–42.3	0.0			
a280-151-c2-w30-1000-1200						
a280-51-c2-w30-1500-2000	–20.5	–44.4	33.3	–27.4	–44.4	100.0
a280-61-c2-w30-1500-2000	–14.8	–28.6	0.0	–27.2	–28.6	0.0
a280-71-c2-w30-1500-2000	–24.2	–38.5	0.0	–26.7	–46.2	0.0
a280-81-c2-w30-1500-2000	–14.4	–41.7	0.0	–24.7	–41.7	25.0
a280-91-c2-w30-1500-2000	–20.9	–27.3	0.0			
a280-101-c2-w30-1500-2000						
a280-111-c2-w30-1500-2000	–12.5	–35.7	0.0			
a280-121-c2-w30-1500-2000						
a280-131-c2-w30-1500-2000						
a280-141-c2-w30-1500-2000						
a280-151-c2-w30-1500-2000						

Table 6.13 – Impact of the number of stacks on the results for instances in class C2 *(cont'd)*

	1 stack VS 2 stacks			1 stack VS 3 stacks		
Instance	Δ Dist (%)	Δ Veh (%)	Δ maxItem (%)	Δ Dist (%)	Δ Veh (%)	Δ maxItem (%)
a280-51-c2-w45-500-1000	–22.6	–30.0	33.3	–13.0	–40.0	33.3
a280-61-c2-w45-500-1000	–14.6	–20.0	0.0	–18.3	–30.0	25.0
a280-71-c2-w45-500-1000	–17.1	–38.5	25.0	–29.7	–38.5	50.0
a280-81-c2-w45-500-1000	–24.0	–33.3	0.0	–26.0	–40.0	25.0
a280-91-c2-w45-500-1000						
a280-101-c2-w45-500-1000						
a280-111-c2-w45-500-1000						
a280-121-c2-w45-500-1000						
a280-131-c2-w45-500-1000						
a280-141-c2-w45-500-1000						
a280-151-c2-w45-500-1000						
a280-51-c2-w45-1000-1200	–25.9	–30.0	33.3	–26.2	–30.0	33.3
a280-61-c2-w45-1000-1200	–31.4	–36.4	33.3	–40.5	–45.5	33.3
a280-71-c2-w45-1000-1200	–15.3	–41.7	0.0	–24.9	–41.7	25.0
a280-81-c2-w45-1000-1200	–10.9	–33.3	33.3	–15.6	–33.3	33.3
a280-91-c2-w45-1000-1200	–23.4	–28.6	0.0	–24.8	–35.7	0.0
a280-101-c2-w45-1000-1200	–19.9	–29.4	0.0			
a280-111-c2-w45-1000-1200						
a280-121-c2-w45-1000-1200						
a280-131-c2-w45-1000-1200						
a280-141-c2-w45-1000-1200						
a280-151-c2-w45-1000-1200						

Table 6.13 – Impact of the number of stacks on the results for instances in class C2 *(cont'd and end)*

	1 stack VS 2 stacks			1 stack VS 3 stacks		
Instance	Δ Dist (%)	Δ Veh (%)	Δ maxItem (%)	Δ Dist (%)	Δ Veh (%)	Δ maxItem (%)
a280-51-c2-w45-1500-2000	–17.7	–28.6	25.0	–30.7	–28.6	0.0
a280-61-c2-w45-1500-2000	–24.3	–36.4	33.3	–33.9	–45.5	66.7
a280-71-c2-w45-1500-2000	–19.1	–36.4	25.0			
a280-81-c2-w45-1500-2000	–19.4	–22.2	25.0	–17.2	–33.3	25.0
a280-91-c2-w45-1500-2000	4.2	–33.3	0.0	–24.1	–33.3	0.0
a280-101-c2-w45-1500-2000						
a280-111-c2-w45-1500-2000	–13.3	–25.0	0.0			
a280-121-c2-w45-1500-2000						
a280-131-c2-w45-1500-2000						
a280-141-c2-w45-1500-2000						
a280-151-c2-w45-1500-2000						
Average	–21.8	–35.0	15.8	–27.2	–39.1	31.4

CHAPITRE 7 DISCUSSION GÉNÉRALE

Dans cette thèse, deux variantes du problème de tournées de véhicules avec cueillettes, livraisons, fenêtres de temps et contrainte de manutention ont été étudiées, le PDPTWL et le PDPTWMS. Bien que cette thèse se concentre sur les problèmes de tournées de véhicules avec cueillettes, livraisons, fenêtres de temps et où la séquence de livraison de la marchandise doit respectée la politique LIFO, plusieurs idées présentées pourraient être adaptées à d'autres politiques de manutention.

7.1 Synthèse des travaux

L'objectif de cette thèse était de développer des algorithmes exacts et heuristiques afin de résoudre pour la première fois le problème de tournées de véhicules avec cueillettes, livraisons, fenêtres de temps et contrainte de manutention.

Dans le chapitre 4, nous avons présenté une formulation mathématique pour le PDPTWL. Nous avons également développé trois algorithmes exacts de génération de colonnes où la politique LIFO est introduite à la fois dans le problème maître et dans le sous-problème. L'algorithme d'étiquetage et le critère de dominance permettant de s'assurer du respect de la politique LIFO sont novateurs.

Dans le chapitre 5, nous avons présenté un algorithme génétique hybride pour le PDPTWL. Au cours de la dernière décennie, les algorithmes heuristiques les plus performants pour résoudre les problèmes de tournées de véhicules sont des algorithmes génétiques et de recherche à grand voisinage. Peu d'algorithmes mélangent ces deux approches. L'algorithme génétique hybride est en lui-même une innovation. De plus, grâce à cet algorithme génétique, il est maintenant possible de résoudre des instances allant jusqu'à 300 requêtes en moins de trois heures. Les résultats obtenus par l'heuristique sont de haute qualité.

Finalement, dans le chapitre 6, nous avons proposé une formulation pour le PDPTWMS. Nous avons également développé deux algorithmes exacts de génération de colonnes. La nouvelle représentation de pile proposée dans ce chapitre permet d'éliminer la symétrie entre les compartiments et est donc appropriée pour le cas à plusieurs compartiments. De plus, la méthode hybride présente plusieurs aspects intéressants. En effet, il est possible de générer des chemins qui sont irréalisables avec une certaine configuration des items dans le véhicule, mais pour lesquels il existe une autre configuration réalisable. Dans ce cas, il suffit de résoudre un problème de plus court chemin contenant un réseau réduit pour déterminer s'il existe une configuration réalisable ou pas. Par le passé, il était nécessaire de résoudre un problème de sac à dos pour affecter chaque item à une pile. Avec les algorithmes proposés, il n'est plus nécessaire de résoudre un problème additionnel car la structure est comprise dans la résolution du problème de plus court chemin.

Cette thèse démontre également que l'ajout de la politique LIFO peut faire augmenter le nombre de véhicules requis pour compléter l'ensemble des requêtes et peut faire augmenter les coûts reliés à la distance parcourue. Toutefois, sur les instances testées, le nombre de véhicules augmente de un pour seulement quatre des 14 instances et, pour les autres instances, les coûts reliés à la distance augmentent d'au plus 20%. De plus, contrairement à ce qu'on pourrait penser, augmenter le nombre de compartiments dans un véhicule n'a pas nécessairement un impact positif sur le nombre de véhicules utilisés et sur la distance totale parcourue. Pour les instances testées, augmenter le nombre de piles de un à deux fait diminuer le nombre de véhicules utilisés ainsi que la distance totale parcourue, mais l'augmenter de deux à trois n'est pas nécessairement plus avantageux.

7.2 Limitations de la solution proposée et améliorations futures

Les algorithmes exacts et heuristiques développés dans cette thèse pour résoudre le PDPTWL et le PDPTWMS sont innovateurs. Malgré tout, ils possèdent certains défauts.

Tout d'abord, tous les algorithmes proposés se concentrent sur la politique de manutention LIFO. Celle-ci doit être respectée à tout prix. Dans la pratique, il est fort probable que les camionneurs désirent respecter cette politique, mais que, de temps en temps, ils permettent la réorganisation de la marchandise à l'intérieur du véhicule. Il serait donc intéressant de développer des algorithmes permettant la réorganisation de la marchandise à un coût. Nous avons d'ailleurs débuté des travaux qui vont dans cette direction, mais qui ne sont pas contenus dans cette thèse. La considération de diverses politiques de manutention ajoute de la complexité au problème. En effet, chaque politique de manutention doit être gérée séparément.

De plus, dans tous les algorithmes de génération de colonnes proposés, nous avons mis l'emphase sur le développement des algorithmes d'étiquetage et des critères de dominance spécialisés. Afin de rendre les algorithmes de génération de colonnes plus performants, nous pensons qu'il serait intéressant de développer plus d'heuristiques pour générer les routes, de développer des méthodes de branchement plus appropriées et de proposer des inégalités valides permettant de réduire le saut d'intégrité.

Au sein d'un noeud de branchement, il faudrait d'abord permettre de générer des routes avec des heuristiques plutôt qu'avec l'algorithme d'étiquetage et le critère de dominance exact. Ces heuristiques pourraient se baser sur l'algorithme génétique développé au chapitre 5. Ce faisant, plusieurs instances pourraient probablement être résolues plus rapidement et certaines instances non résolues pourraient l'être.

Nous avons également remarqué que certaines instances ne peuvent pas être résolues à cause d'un arbre de branchement trop grand. Il serait donc intéressant de développer une méthode de branchement plus appropriée. En ce moment, la méthode de branchement se fait sur le flot sortant d'un ensemble de noeuds. On pourrait tenter de mieux choisir les sous-ensembles de noeuds sur lesquels brancher et aussi définir de nouvelles règles de branchement.

De plus, nous n'avons pas développé de coupes spécialisées pour le PDPTWL et le PDPTWMS afin de renforcer la borne inférieure. Nous avons implanté des coupes connues pour le PDPTW et les avons adaptées au PDPTWL et au PDPTWMS. Il est probable que le développement de coupes plus spécialisées pour le PDPTWL et le PDPTWMS permettrait d'accélérer la vitesse de résolution en réduisant la taille de l'arbre de branchement.

L'heuristique développée au chapitre 5 pourrait être améliorée sur plusieurs aspects. Tout d'abord, nous avons utilisé une structure informatique en liste pour représenter la contrainte de manutention LIFO. Certains auteurs (voir Li *et al.* (2011); Gao *et al.* (2011); Cheang *et al.* (2012)) ont démontré qu'une structure informatique plus appropriée pour la contrainte de manutention LIFO pouvait réduire les temps de calcul. Nous avons remarqué que lors de la résolution, ce sont les vérifications du respect des contraintes de capacité, de fenêtres de temps et de la politique LIFO qui prennent le plus de temps. Dans notre cas, il serait donc intéressant de développer une structure qui pourrait à la fois conserver les données sur la capacité du véhicule, sur les fenêtres de temps et sur la politique LIFO. De plus, les opérateurs de recherche locale pourraient être améliorés. Par exemple, l'opérateur *inter-route multiple request exchange* permet d'échanger des listes d'arcs, mais se restreint à deux cas particuliers. Il serait intéressant de permettre tous les échanges possibles. Cela pourrait probablement permettre d'obtenir de meilleures solutions. Il serait également intéressant de perfectionner la gestion de la population. Nous nous sommes inspirés des idées proposées par Vidal *et al.* (2012), mais ne sommes pas aussi raffinés que ces derniers. Finalement, la gestion des solutions intermédiaires non réalisables pourrait être plus sophistiquée. En effet, ces solutions ne sont pas permises en tout temps. Puis, nous ne permettons pas de solutions intermédaires non réalisables par rapport à la politique LIFO. Permettre plus souvent les solutions irréalisables et permettre des solutions irréalisables par rapport à la politique LIFO nous permettrait une exploration plus large du voisinage. De plus, la réparation des solutions non réalisables est assez simple et pourrait être perfectionnée.

CHAPITRE 8 CONCLUSION

En conclusion, nous avons proposé une formulation mathématique pour le PDPTWL et le PDPTWMS. Nous avons développé des algorithmes exacts pour résoudre ces deux problèmes, ainsi qu'un algorithme génétique hybride pour résoudre le PDPTWL. Plusieurs idées présentées dans cette thèse peuvent être adaptées à d'autres variantes des problèmes de tournées de véhicules. De plus, les algorithmes d'étiquetage et les critères de dominance proposés pour résoudre les sous-problèmes en génération de colonnes présentent des idées novatrices pour la représentation d'une pile. Ces algorithmes sont des premiers vers la résolution pratique de problèmes complexes de tournées de véhicules avec cueillettes et livraisons et contraintes de manutention. Nous espérons qu'ils serviront de base à de travaux futurs qui mèneront à des algorithmes pratiques utilisés par de nombreuses compagnies de transport.

Quatre ans de travail, quatre ans de bonheur et quatre ans de moments plus difficiles ont mené à cette thèse. La qualité des travaux réalisés démontre que tout étudiant motivé peut réussir le passage d'une école de gestion, HEC Montréal, vers une école de génie, l'École Polytechnique de Montréal, bien que l'inverse soit aussi difficile.

En guise de mot de la fin, je vous laisse sur cette pensée de Winston Churchill.

"Now this is not the end.
It is not even the beginning of the end.
But it is, perhaps, the end of the beginning."
– Winston Churchill

RÉFÉRENCES

M. Alba Martínez, J.-F. Cordeau, M. Dell'Amico et M. Iori, (2013), A branch-and-cut algorithm for the double traveling salesman problem with multiple stacks. *INFORMS Journal on Computing*, *25*(1), 41–55.

M. Ambrosini, T. Caruso, S. Foresti et G. Righini, (2004). A GRASP for the pickup and delivery problem with rear loading. Rapport technique, Université de Milan, Milan, Italie.

R. Baldacci, E. Bartolini et A. Mingozzi, (2011a), An exact algorithm for the pickup and delivery problem with time windows. *Operations Research*, *59*(2), 414–426.

R. Baldacci, E. Bartolini, A. Mingozzi et R. Roberti, (2010), An exact solution framework for a broad class of vehicle routing problems. *Computational Management Science*, *7*(3), 229–268.

R. Baldacci, A. Mingozzi et R. Roberti, (2011b), New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research*, *59*(5), 1269–1283.

M. Battara, J.-F. Cordeau et M. Iori, (2014). Pickup-and-delivery problems for goods transportation. P. Toth et D. Vigo, éditeurs, *Vehicle Routing : Problems, Methods, and Applications - Second Edition*, MOS-SIAM, Philadelphie, chapitre 6. 161–191.

M. Battarra, G. Erdoğan, G. Laporte et D. Vigo, (2010), The traveling salesman problem with pickups, deliveries, and handling costs. *Transportation Science*, *44*(3), 383–399.

E. Benavent, M. Landete, E. Mota et G. Tirado, (2015), The multiple vehicle pickup and delivery problem with LIFO constraints. *European Journal of Operational Research*, *243*(3), 752–762.

R. Bent et P. Van Hentenryck, (2006), A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. *Computers & Operations Research*, *33*(4), 875–893.

G. Berbeglia, J.-F. Cordeau, I. Gribkovskaia et G. Laporte, (2007), Static pickup and delivery problems : A classification scheme and survey. *TOP*, *15*(1), 1–31.

O. Bräysy et M. Gendreau, (2005a), Vehicle routing problem with time windows, Part I : Route construction and local search algorithms. *Transportation Science*, *39*(1), 104–118.

O. Bräysy et M. Gendreau, (2005b), Vehicle routing problem with time windows, Part II : Metaheuristics. *Transportation Science*, *39*(1), 119–139.

F. Carrabs, R. Cerulli et J.-F. Cordeau, (2007a), An additive branch-and-bound algorithm for the pickup and delivery traveling salesman problem with LIFO or FIFO loading. *INFOR*, *45*(4), 223–238.

F. Carrabs, J.-F. Cordeau et G. Laporte, (2007b), Variable neighborhood search for the pickup and delivery traveling salesman problem with LIFO loading. *INFORMS Journal on Computing*, *19*(4), 618–632.

L. Cassani, (2004). Algoritmi euristici per il TSP with rear-loading. Tesi di laurea, Université de Milan, Milan, Italie.

B. Cheang, X. Gao, A. Lim, H. Qin et W. Zhu, (2012), Multiple pickup and delivery traveling salesman problem with last-in-first-out loading and distance constraints. *European Journal of Operational Research*, *223*(1), 60–75.

M. Cherkesly, G. Desaulniers et G. Laporte, (2014), Branch-price-and-cut algorithms for the pickup and delivery problem with time windows and last-in-first-out loading. *Transportation Science*.

J.-F. Cordeau, (2006), A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, *54*(3), 573–586.

J.-F. Cordeau, M. Iori, G. Laporte et J. Salazar-González, (2010), A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with LIFO loading. *Networks*, *55*(1), 46–59.

J.-F. Cordeau et G. Laporte, (2003), A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B*, *37*(6), 579–594.

J.-F. Côté, C. Archetti, M. Speranza, M. Gendreau et J.-Y. Potvin, (2012a), A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with multiple stacks. *Networks*, *60*(4), 212–226.

J.-F. Côté, M. Gendreau et J.-Y. Potvin, (2012b), Large neighborhood search for the pickup and delivery traveling salesman problem with multiple stacks. *Networks*, *60*(1), 19–30.

G. Desaulniers, J. Desrosiers, A. Erdmann, M. M. Solomon et F. Soumis, (2002). VRP with pickup and delivery. P. Toth et D. Vigo, éditeurs, *The Vehicle Routing Problem. SIAM Monographs on Discrete Mathematics and Applications*, SIAM, Philadelphie, chapitre 9. 225–242.

G. Desaulniers, J. Desrosiers et M. M. Solomon, eds., (2005). *Column Generation*. Springer, New York.

G. Desaulniers, J. Desrosiers et S. Spoorendonk, (2011), Cutting planes for branch-and-price algorithms. *Networks*, *58*(4), 301–310.

G. Desaulniers, F. Lessard et A. Hadjar, (2008), Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows. *Transportation Science*, *42*(3), 387–404.

G. Desaulniers, O. B. G. Madsen et S. Ropke, (2014). The vehicle routing problem with time windows. P. Toth et D. Vigo, éditeurs, *Vehicle Routing : Problems, Methods, and Applications - Second Edition*, MOS-SIAM, Philadelphie, chapitre 5. 119–159.

M. Desrochers, J. Desrosiers et M. M. Solomon, (1992), A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, *40*(2), 342–354.

K. F. Doerner et J. J. Salazar-González, (2014). Pickup-and-delivery problems for people transportation. P. Toth et D. Vigo, éditeurs, *Vehicle Routing : Problems, Methods, and Applications - Second Edition*, MOS-SIAM, Philadelphie, chapitre 7. 193–212.

G. Erdoğan, M. Battarra, G. Laporte et D. Vigo, (2012), Metaheuristics for the traveling salesman problem with pickups, deliveries and handling costs. *Computers & Operations Research*, *39*(5), 1074–1086.

T. A. Feo et M. G. C. Resende, (1989), A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, *8*(2), 67–71.

T. A. Feo et M. G. C. Resende, (1995), Greedy randomized adaptive search procedures. *Journal of Global Optimization*, *6*(2), 109–133.

X. Gao, A. Lim, H. Qin et W. Zhu, (2011). Multiple pickup and delivery TSP with LIFO and distance constraints : A VNS approach. K. Mehrotra, C. Mohan, J. Oh, P. Varshney et M. Ali, éditeurs, *Modern Approaches in Applied Intelligence*, Springer Berlin / Heidelberg, vol. 6704 de *Lecture Notes in Computer Science*. 193–202.

M. Iori et S. Martello, (2010), Routing problems with loading constraints. *TOP*, *18*(1), 4–27.

M. Iori et J. Riera-Ledesma, (2015). Exact algorithms for the double vehicle routing problem with multiple stacks. Rapport technique, DIIS-Universidad de La Laguna, San Cristóbal de La Laguna, Espagne.

S. Irnich et G. Desaulniers, (2005). Shortest path problems with resource constraints. G. Desaulniers, J. Desrosiers et M. M. Solomon, éditeurs, *Column Generation*, Springer, New York, chapitre 2. 33–65.

S. Irnich, P. Toth et D. Vigo, (2014). The family of vehicle routing problems. P. Toth et D. Vigo, éditeurs, *Vehicle Routing : Problems, Methods, and Applications - Second Edition*, MOS-SIAM, Philadelphie, chapitre 1. 1–33.

M. Jepsen, B. Petersen, S. Spoorendonk et D. Pisinger, (2008), Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, *56*(2), 497–511.

B. Kallehauge, J. Larsen, O. B. G. Madsen et M. M. Solomon, (2005). Vehicle routing problem with time windows. G. Desaulniers, J. Desrosiers et M. M. Solomon, éditeurs, *Column Generation*, Springer, New York, chapitre 3. 67–98.

N. Kohl, J. Desrosiers, O. B. G. Madsen, M. M. Solomon et F. Soumis, (1999), 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, *33*(1), 101–116.

G. Laporte, (2007), What you should know about the vehicle routing problem. *Naval Research Logistics*, *54*(8), 811–819.

G. Laporte, (2009), Fifty years of vehicle routing. *Transportation Science*, *43*(4), 408–416.

H. Li et A. Lim, (2003), A metaheuristic for the pickup and delivery problem with time windows. *International Journal on Artificial Intelligence Tools*, *12*(2), 173–186.

Y. Li, A. Lim, W.-C. Oon, H. Qin et D. Tu, (2011), The tree representation for the pickup and delivery traveling salesman problem with LIFO loading. *European Journal of Operational Research*, *212*(3), 482–496.

R. M. Lusby, J. Larsen, M. Ehrgott et D. M. Ryan, (2010), An exact method for the double TSP with multiple stacks. *International Transactions in Operational Research*, *17*(5), 637–652.

D. Naddef et G. Rinaldi, (2002). Branch-and-cut algorithms for the capacitated VRP. P. Toth et D. Vigo, éditeurs, *The Vehicle Routing Problem. SIAM Monographs on Discrete Mathematics and Applications*, SIAM, Philadelphie, chapitre 3. 53–84.

Y. Nagata, O. Bräysy et W. Dullaert, (2010), A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows. *Computers & Operations Research*, *37*(4), 724–737.

S. Parragh, K. F. Doerner et R. F. Hartl, (2008a), A survey on pickup and delivery problems, Part I : Transportation between customers and depot. *Journal für Betriebswirtschaft*, *58*(1), 21–51.

S. Parragh, K. F. Doerner et R. F. Hartl, (2008b), A survey on pickup and delivery problems, Part II : Transportation between pickup and delivery locations. *Journal für Betriebswirtschaft*, *58*(2), 81–117.

H. Petersen, C. Archetti et M. Speranza, (2010), Exact solutions to the double travelling salesman problem with multiple stacks. *Networks*, *56*(4), 229–243.

H. Petersen et O. B. G. Madsen, (2009), The double travelling salesman problem with multiple stacks–formulation and heuristic solution approaches. *European Journal of Operational Research*, *198*(1), 139–147.

C. Prins, (2004), A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, *31*(12), 1985–2002.

M. G. C. Resende et C. C. Ribeiro, (2010). Greedy randomized adaptive search procedures : Advances, hybridizations, and applications. M. Gendreau et J.-Y. Potvin, éditeurs, *Handbook of Metaheuristics*, Springer, New York. 283–319.

Y. Rochat et E. D. Taillard, (1995), Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, *1*(1), 147–167.

S. Ropke et J.-F. Cordeau, (2008). Branch-and-cut-and-price for the pickup and delivery problem with time windows. Rapport technique, CIRRELT-2008-33, HEC Montréal, Montréal.

S. Ropke et J.-F. Cordeau, (2009), Branch-and-cut-and-price for the pickup and delivery problem with time windows. *Transportation Science*, *43*(3), 267–286.

S. Ropke, J.-F. Cordeau et G. Laporte, (2007), Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks*, *49*(4), 258–272.

S. Ropke et D. Pisinger, (2006), An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, *40*(4), 455–472.

M. W. P. Savelsbergh, (1985), Local search in routing problems with time windows. *Annals of Operations research*, *4*(1), 285–305.

P. Shaw, (1997). A new local search algorithm providing high quality solutions to vehicle routing problems. Rapport technique, Départment d'informatique, Université de Strathclyde, Glasgow, Royaume-Uni.

M. Sol, (1994). *Column generation techniques for pickup and delivery problems.* Thèse de doctorat, Université technique de Eidhoven, Eidhoven, Pays-Bas.

SPEC, (2014). CPU 2006 results. `http://www.spec.org/cpu2006/results/cpu2006.html`.

Statistique Canada, (2012). Annuaire du Canada 2012, no.11-402-X, Chapitre 29-Transport. `http://www.statcan.gc.ca/pub/11-402-x/2012000/pdf/transport-fra.pdf`.

P. Toth et D. Vigo, (2002). *The Vehicle Routing Problem.* SIAM, Philadelphie.

P. Toth et D. Vigo, (2014). *Vehicle Routing : Problems, Methods, and Applications*, vol. 18. MOS-SIAM. Philadelphie.

T. Vidal, T. G. Crainic, M. Gendreau, N. Lahrichi et W. Rei, (2012), A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, *60*(3), 611–624.

T. Vidal, T. G. Crainic, M. Gendreau et C. Prins, (2013a), Heuristics for multi-attribute vehicle routing problems : a survey and synthesis. *European Journal of Operational Research*, *231*(1), 1–21.

T. Vidal, T. G. Crainic, M. Gendreau et C. Prins, (2013b), A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers & Operations Research*, *40*(1), 475–489.