Résolution de problèmes de tournées avec synchronisation : applications au cas multi-échelons et au cross-docking

Philippe Grangier

Thèse de Doctorat

Philippe GRANGIER

Mémoire présenté en vue de l'obtention du
grade de Docteur de l'École nationale supérieure des Mines de Nantes
sous le label de l'Université de Nantes Angers Le Mans

École doctorale : STIM

Discipline : Informatique et applications
Spécialité : Recherche opérationelle
Unité de recherche : IRCCyN

Soutenue le 8 décembre 2015
Thèse n° : 2015EMNA0228

Résolution de problèmes de tournées avec synchronisation : applications au cas multi-échélons et au cross-docking

JURY

Rapporteurs :	**M. Dominique FEILLET**, Professeur, Ecole des Mines de Saint Etienne **M. Daniele VIGO**, Professeur, Université Bologne
Examinateurs :	**M. Jorge MENDOZA**, Maître de conférences, Polytech Tours **M. Olivier PÉTON**, Maître assistant, École des Mines de Nantes **M^me^ Caroline PRODHON**, Maître de conférences, Université de Technologie de Troyes
Directeur de thèse :	**M. Fabien LEHUÉDÉ**, Maître assistant, École des Mines de Nantes
Co-directeurs de thèse :	**M. Michel GENDREAU**, Professeur, École Polytechnique de Montréal **M. Louis-Martin ROUSSEAU**, Professeur, École Polytechnique de Montréal

1

Introduction

Au cours des dernières années, la consolidation est apparue comme des un facteurs critiques pour l'efficacité de la chaîne logistique. Ainsi, le monde des transports s'est tourné vers des systèmes de plus en plus interconnectés qui offrent plus de flexibilité et permettent de prendre en compte des exigences environnementales plus strictes. Cette transition a notamment été rendue possible par les progrès des systèmes d'information géographique. Dans le domaine des tournées de véhicules un nouveau champ d'investigation a ainsi été récemment identifié : les tournées de véhicules avec contraintes de synchronisation. Ce type de contraintes modélise des situations dans lesquelles « plus d'un véhicule peut ou doit être utilisé pour réaliser une tâche » [31]. C'est le cas, par exemple, dans un problème de collectes et livraisons lorsque que le véhicule qui collecte la requête est différent du véhicule qui la livre. Les deux véhicules doivent alors se rencontrer en route pour le transfert de la requête. Alors, tout changement dans la tournée de l'un peut modifier l'heure du transfert et donc affecter l'autre véhicule. Ce phénomène est appelé *interdépendance* entre tournées et distingue les problèmes de tournées de véhicules avec contraintes de synchronisation des problèmes de tournées de véhicules classiques. En effet, dans ces derniers, une fois l'affectation des tâches à chaque véhicule faite, la réalisation de chaque tournée est indépendante des autres. Cette thèse, intitulée *résolution de problèmes de tournées avec synchronisation : applications au cas multi-échelons et au cross-docking*, porte sur l'étude de trois problèmes présentant des contraintes de synchronisation. Un problème de chargement étudié pendant un stage réalisé au cours de cette thèse est également présenté.
Dans ce chapitre d'introduction, nous présentons tout d'abord les enjeux et objectifs de la thèse, puis son contexte scientifique. Dans un troisième temps, chaque article constituant ce manuscrit est présenté et résumé. Enfin le chapitre se termine sur les perspectives ouvertes par ces travaux.

1.1 Enjeux et objectifs de la thèse

Drexl a publié en 2012 un article [31] faisant la synthèse des problèmes de tournées de véhicules avec contraintes de synchronisation. En perpective de cet état de l'art, est présenté

comme enjeu majeur la conception d'une métaheuristique pour résoudre des instances de taille réelle de problèmes de tournées de véhicules avec contraintes de synchronisation multiples. La thèse de Masson [62] a porté sur des problèmes de collectes et livraisons avec transferts, et a proposé des résultats et des méthodes pour le cas de précédences temporelles. En particulier, deux défis majeurs étaient identifiés en présence de contraintes de précédence : (1) la modélisation et la gestion efficace des contraintes temporelles, et (2) la gestion de l'espace des solutions (la possibilité de transférer des requêtes rend les espaces de solutions bien plus grands que dans le cas sans transfert). Cette thèse s'inscrit dans la suite de celle de Masson. Son objectif est de relever les défis évoqués précédemment en présence d'autres contraintes de synchronisation : synchronisation temporelle exacte et ressources limitées. Ces contraintes apparaissent dans l'étude de problèmes apparus récemment en logistique, notamment pour la livraison de marchandises en ville [15] : le problème de livraison à deux échelons et le problème de livraison avec cross-docking.

1.2 Contexte scientifique

Une revue de littérature est intégrée à chaque article. Nous présentons ici les principales références de la littérature sur lesquelles s'appuie cette thèse. Celles-ci sont regroupées en quatres axes : les contraintes de synchronisation, la modélisation et la gestion des contraintes temporelles, les heuristiques à voisinage large et le problème de couverture par ensembles dans les matheuristiques.

1.2.1 Contraintes de synchronisation

Dans [31], Drexl identifie les types de synchronisation suivants (1) synchronisation d'opérations, (2) synchronisation de mouvements, (3) synchronisation des quantités et (4) synchronisation de ressources. Afin de mieux les présenter, nous les illustrons ci-après par des exemples.

La synchronisation d'opérations traduit l'existence d'une relation géographique et/ou temporelle (précédence, simultanéité, délai minimum, délai maximum) entre deux opérations. Ce type de synchronisation apparait le plus souvent dans les systèmes ou des requêtes sont transférées d'un véhicule à un autre, tels que : le problème de collectes et livraisons avec transferts [18] (évoqué dans l'introduction de ce chapitre), ou les problèmes dits multi-échelons [74].

La synchronisation de mouvements traduit la présence de véhicules qui ne peuvent se déplacer de manière autonome. C'est le cas des remorques dételables dans les attelages à double remorques [30]. La composition de paires chauffeur-véhicule peut aussi être vue comme une synchronisation de mouvements.

La synchronisation de quantités traduit le cas où la quantité à livrer pour une requête peut ou doit être partagée entre plusieurs véhicules. Ce problème est connu sous le nom de *split delivery* dans la littérature [2].

La synchronisation de ressources traduit la présence dans le système d'au moins une ressource de capacité limitée. C'est le cas par exemple dans un centre de distribution lorsque le nombre de quais de chargement est restreint [48].

Dans cette thèse, les problèmes abordés présentent des contraintes de synchronisation d'opérations et de resources.

1.2.2 Modélisation et gestion des fenêtres de temps

Dans les problèmes de tournées de véhicules on parle de fenêtres de temps lorsqu'il existe un ensemble P de points, tel que tout $p \in P$ doit être visité dans une fenêtre horaire $[a_p, b_p]$. Si le véhicule arrive à p avant l'instant a_p, on considère généralement qu'il peut attendre jusqu'au début de la fenêtre de temps, à l'inverse, une arrivée postérieure à l'instant b_p est interdite. Au problème de construction des tournées de véhicules se rajoute alors un problème d'ordonnancement des tournées.

La modélisation des contraintes de synchronisation peut se faire par ajout de contraintes sur les variables temporelles concernées dans le problème sans synchronisation équivalent [52, 12, 18]. Les problèmes d'ordonnancement dans les cas avec synchronisation sont donc au moins aussi difficiles à résoudre que les problèmes de planification en l'absence de contraintes de synchronisation. En pratique, la difficulté tient souvent au fait qu'en présence de contraintes de synchronisation, le problème d'ordonnancement doit être envisagé dans la solution au complet, là où il est généralement séparable par véhicules, et donc plus petit, dans les problèmes de tournées de véhicules classiques.

De nombreuses méthodes heuristiques (comme les méthodes à base de recherche locale ou les méthodes à voisinage large) procèdent en essayant de très nombreuses suppressions-insertions de requêtes dans des solutions. Il est critique pour l'efficacité de ces méthodes, que les tests associés à la réalisabilté temporelle d'une insertion soient les plus efficaces possibles. Pour le problème de tournées de véhicules avec fenêtres de temps (Vehicle Routing Problem with Time Windows - VRPTW), Savelsbergh [88] a proposé la notion de *forward time slacks*, permettant de vérifier en temps constant si une insertion est réalisable d'un point de vue temporel. Pour cela, à chaque fois qu'une insertion est réalisée, une matrice de coefficients doit être recalculée. Ceci a une compléxité quadratique, mais comme les heuristiques testent beaucoup d'insertions avant d'en réaliser une, la combinaison : test en temps constant + recalcul quadratique est généralement un meilleur compromis que de résoudre, pour chaque insertion potentielle, le problème d'ordonnancement de complexité linéaire. Cette notion a été étendue au problème de collectes et livraisons avec transferts par Masson et al. [64].

Dans cette thèse, un accent particulier est mis sur la modélisation temporelle des contraintes de synchronisation dans les problèmes considérés, et le développement de méthodes efficaces de gestion de la réalisabilité temporelle.

1.2.3 Méthode de recherche à voisinage large

Dans cette thèse, les méthodes proposées appartiennent à la famille des méthodes à voisinage large (Large Neighborhood Search - LNS), introduites par Shaw [92] pour les problèmes de tournées de véhicules. Les méthodes LNS sont des méthodes heuristiques itératives, c'est à dire que l'on répète une suite d'opérations modifiant une solution dans le but de l'améliorer. Elles sont dites *à voisinage large* car à chaque itération une grande partie de la solution peut être modifiée (entre 10% et 40% dans [82] par exemple). Ceci les distingue des méthodes à base de *recherche locale* où, à chaque itération, seule une petite fraction de la solution est modifiée. Un pseudo-code de LNS est présenté dans l'Algorithme 1.

Les principales caractéristiques de la méthode LNS sont les suivantes :

- à chaque itération on sélectionne (1) le nombre Φ de requêtes à retirer de la solution courante (l. 5), (2) l'opérateur de sélection des requêtes à retirer M^- et (3) l'opérateur

Résultat : La meilleure solution rencontrée $s^\star$
1 Créer une solution initiale s au problème
2 $s^\star := s$
3 **tant que** *la condition d'arrêt n'est pas atteinte* **faire**
4 $s' := s$
5 **Nombre de requêtes à retirer :** sélectionner un nombre Φ de requêtes à retirer de s'
6 **Sélection des opérateurs :** sélectionner un opérateur de retrait M^- et un opérateur de réinsertion M^+
7 **Destruction :** utiliser M^- pour retirer Φ requêtes de s', et les mettre dans une liste $\mathcal{L}$
8 **Réparation :** utiliser M^+ pour réinsérer les requêtes de $\mathcal{L}$ dans s'
9 **si** s' *remplit le critère d'acceptation* **alors**
10 $s := s'$
11 **fin**
12 **si** s' *est une meilleure solution que* $s^\star$ **alors**
13 $s^\star := s'$
14 **fin**
15 **fin**
16 **retourner** $s^\star$

Algorithm 1 : Méthode de recherche à voisinage large (LNS)

de réinsertion des requêtes M^+ (l. 6) ;

- les requêtes retirées de s' en utilisant M^- sont mises dans une liste de requêtes non présentes associée à s' : $\mathcal{L}$ (l.9) ;
- la solution ainsi obtenue peut devenir la solution courante s, si elle remplit un certain critère (l. 9). Pour assurer la diversité de la recherche, ce critère peut accepter s' comme solution courante alors qu'elle est de moins bonne qualité que s.

L'une des variantes les plus courantes de LNS est la méthode de Recherche Adaptative à Voisinage Large (Adaptive Large Neighborhood Search - ALNS) proposée par Ropke et Pisinger [82]. Dans celle-ci, la sélection des méthodes de retrait et de réinsertion n'est plus aléatoire mais basée sur les performances passées : plus un opérateur a contribué à trouver de bonnes ou de nouvelles solutions (au sens : non encore visitées au cours de la recherche), plus il a de chances d'être sélectionné. Les méthodes de la famille des LNS ont été utilisées pour résoudre de très nombreux problèmes en tournées de véhicules [77, 78, 19, 27, 63, 6]. L'ALNS détient toujours les meilleurs résultats pour un certain nombre d'instances pour le problème de collectes et livraisons avec fenêtres de temps (voir [99] Chapitre 6), et reste une métaheuristique générique aux très bonnes performances étant à moins de 0,8% en moyenne des meilleurs méthodes connues pour le CVRP (voir [99], Chapitre 4). De plus, l'interdépendance impose une structure temporelle forte aux solutions, on peut donc penser qu'une méthode à base de recherche locale aura beaucoup de difficultés pour diversifier les solutions obtenues, à l'inverse détruire une grande partie de la solution, comme le font les méthodes à voisinage large, devrait permettre de contourner ce problème. Un tel constat est notamment fait par Schrimpf et al. [91].

Dans cette thèse, les méthodes présentées sont des méthodes à voisinage large, avec un accent particulier mis sur la définition de nouveaux voisinages adaptés au problème

considéré.

1.2.4 Couverture par ensembles dans les matheuristiques

Le VRP peut être formulé comme un problème de couverture par ensembles (Set Partitioning Problem - SPP), dont l'objectif est de couvrir les clients avec des routes réalisables à coût minimum. C'est notamment ce type de formulation qui est utilisée dans les méthodes de génération de colonnes ([99] Chapitre 2). Stocker certaines routes découvertes au fil de la recherche d'une (méta)heuristique, puis résoudre un SPP avec les routes stockées apparait comme une idée assez naturelle. D'après [3], sa première implémentation remonte à 1976 et est due à Foster et Ryan [38], elle a notamment été utilisée par Rochat et Taillard dans [81] en tant que post-traitement d'une recherche tabou. Dans la synthèse sur les matheuristiques pour les problèmes de tournées de véhicules d'Archetti et Speranza [3], on peut observer un regain d'intérêt massif pour cette technique à partir de 2008. Il est intéressant de le mettre en parallèle avec les progrès effectués par les solveurs MIP : d'après [8], CPLEX a progressé d'un facteur 29000 sur les problèmes MIP entre 1991 et 2009 (hors progrès dus à l'amélioration de la performance des ordinateurs). En particulier, on assiste à l'émergence de nouvelles méthodes, qui font appel périodiquement (et non plus uniquement en post-traitement) à la résolution d'un SPP [44, 68, 95, 72].
Dans cette thèse, nous proposons à deux reprises et avec deux formulations différentes des méthodes s'appuyant sur la résolution périodique d'un problème basé sur un problème de couverture par ensembles. Ceci contribue à améliorer de manière drastique les performances des LNS.

1.3 Plan du manuscrit et résumé des contributions

Dans cette section, nous présentons un résumé des quatre contributions de la thèse : (1) une méthode de recherche adaptative pour le problème de tournées de véhicules à deux échelons et synchronisation aux satellites, (2) une matheuristique basée sur une recherche à voisinage large pour le problème de tournées de véhicules avec cross-docking, (3) une étude sur le problème de tournées de véhicules avec cross-docking et contraintes de ressources, et (4) un problème réel de chargement 3D étudié dans le cadre d'un stage doctoral au sein de l'entreprise JDA Software.

1.3.1 Une méthode de recherche adaptative pour le problème de tournées de véhicules à deux échelons et synchronisation aux satellites

De plus en plus de villes (comme Londres) interdisent ou limitent fortement l'accès des gros camions de livraison en ville pour des raisons liées à l'écologie et/ou à la qualité de vie. Pour livrer des marchandises, le recours à de plus petits véhicules (éventuellement électriques) devient obligatoire. Or, les centres de distribution urbains (City Distribution Center - CDC) sont généralement situés en grande périphérie pour être proches des grands axes de communication. Aussi, il est économiquement inefficace (allers-retours incessants) voire techniquement impossible (autonomie limitée) que des petits véhicules aillent s'approvisionner directement à un CDC. Une solution innovante consiste à utiliser deux flottes de véhicules : de gros camions pour transporter les marchandises depuis

l'entrepôt jusqu'aux boulevards de ceinture, puis de petits véhicules pour les derniers kilomètres à l'intérieur de la ville. Les requêtes sont transférées du premier échelon (gros véhicules) au second échelon (petits véhicules) dans des lieux appelés *satellites*. L'organisation géographique de ce type de système dit *à deux échelons* est présentée en Figure 1.1. Pour les satellites, deux options sont possibles. Soit utiliser des plateformes spécifiques dédiées, qui permettent notamment le stockage. Celles-ci impliquent une contrainte de précédence entre les véhicules du premier et du deuxième échelon. Soit utiliser des infrastructures existantes (arrêts de transports en commun, parkings, ...) dans lesquelles tout stockage est impossible. Il existe alors une contrainte de synchronisation dite *exacte* entre les véhicules du premier et du deuxième échelon. Le premier cas est appelé *problème de tournées de véhicules à deux échelons*, il a été étudié à plusieurs reprises dans la littérature ([26] présente une revue de synthèse), le deuxième cas est appelé *problème de tournées de véhicules à deux échelons et synchronisation aux satellites*. Nous nous plaçons dans le cadre de ce dernier et, pour davantage de réalisme, nous intégrons différentes caractéristiques : des fenêtres de temps, la possibilité pour un petit véhicule de réaliser plusieurs tournées au cours de la journée et de s'approvisionner à n'importe quel satellite. Le problème ainsi défini est le *problème de tournées de véhicules à deux échelons, multi-trip et synchronisation aux satellites* (Two Echelon Multi-Trip Vehicle Routing Problem with Satellite Synchronization - 2E-MTVRPSS), il avait été identifiée dans la littérature [25] mais, à notre connaissance, aucune méthode n'avait été proposée pour le résoudre.

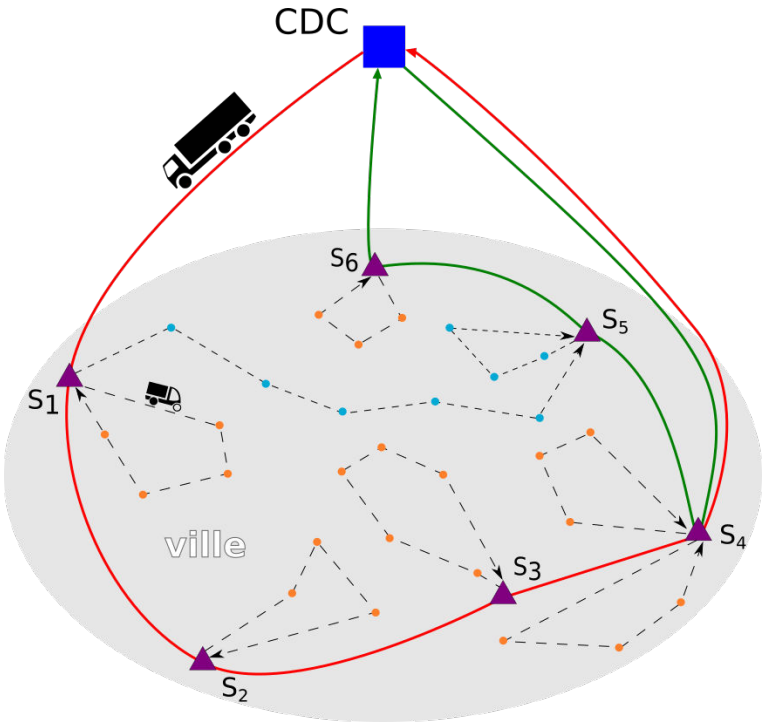


FIGURE 1.1 – Structure géographique du problème de tournées de véhicules à deux échélons dans un contexte urbain.

Nous proposons un algorithme de recherche adaptatif à voisinage large pour le 2E-MTVRPSS. L'une des originalités de ce problème tient au fait que l'on ne sait pas a priori quels satellites vont être utilisés (certains peuvent être utilisés à plusieurs reprises, par différents véhicules, et d'autres jamais), ni par quel satellite va transiter chaque requête. Une conséquence de ceci est la très grande taille de l'espace des solutions. Pour répondre à ce défi, nous avons proposé des opérateurs spécifiques de retrait et de réinsertion. Par

ailleurs, nous avons étendu au cas de synchronisation temporelle exacte la méthode de vérification de la réalisabilité temporelle en temps constant de Masson et al. [64].

Pour évaluer notre méthode, des instances adaptées du VRPTW sont proposées. Les expérimentations montrent que les opérateurs introduits contribuent à l'amélioration de la qualité des solutions et que les efforts autour de la réduction du temps de calcul (voisinages restreints, et test de réalisabilité en temps constant) portent leurs fruits (réduction du temps de calcul d'un facteur supérieur à 20). Enfin nous avons montré que la présence de fenêtres de temps jouait un rôle majeur dans le coût des solutions de ce problème.

Chronologiquement cette contribution a été la première de cette thèse.

1.3.2 Une matheuristique basée sur une méthode de recherche à voisinage large pour le problème de tournées de véhicules avec cross-docking

Le cross-docking est une stratégie de distribution dans laquelle les requêtes sont transportées des points de collecte vers une plateforme intermédiaire, le cross-dock, où elles peuvent être transférées vers d'autres véhicules avant d'être livrées. Le cross-docking permet de profiter d'opportunités de consolidation, et comme il n'y a pas de stockage, les coûts sont réduits par rapport à des centres de distribution traditionnels. Le cross-docking a ainsi été appliqué avec succès dans de nombreux secteurs (automobile, distribution, ...). Un problème de tournées de véhicules associé est le problème de tournées de véhicules avec cross-docking (Vehicle Routing Problem with Cross-Docking - VRPCD), introduit par Wen et al. [104]. Il s'agit d'un problème de collectes et de livraisons intégrant un unique cross-dock. Les véhicules partent du cross-dock, collectent les requêtes, reviennent au cross-dock où ils peuvent décharger/recharger des requêtes, et enfin livrent les requêtes. Des fenêtres de temps existent aux points de collecte et aux points de livraison.

Nous proposons une matheuristique basée sur une recherche à voisinage large pour le VRPCD. La méthodologie proposée combine une méthode de type LNS à la résolution d'un problème SPP. Au cours du LNS, les segments de collecte et de livraison de chaque route des solutions visitées sont stockés dans une mémoire. Périodiquement, un problème de couverture par ensembles et de couplage (Set Partitioning and Matching - SPM) avec ces segments est résolu. Le couplage des segments de collecte et de livraison ainsi que la vérification des contraintes de précédence inhérentes aux transferts sont réalisés au fur et à mesure de la découverte de solutions du SPP. Pour ce faire, un module de programmation par contraintes est embarqué dans un branch-and-bound pour le SPP. Cette approche est un *branch-and-check* [98].

Notre méthode a été évaluée sur les instances de la littérature. Nous avons observé que l'ajout du SPM permettait une diminution significative du coût des solutions (-7,4 %) par rapport à un LNS seul. De plus, par rapport aux autres méthodes de résolution pour le VRPCD [104, 97, 69], notre méthode améliore les meilleurs résultats connus pour 30 instances sur 35, en proposant une performance moyenne meilleure de 0,78% sur les petites instances et de 2,16% sur les grandes instances.

Chronologiquement cette contribution a été la deuxième de cette thèse.

1.3.3 Une étude sur le problème de tournées de véhicules avec cross-docking et contraintes de ressources

Cette contribution s'inscrit dans la suite de la précédente. Selon Buijs et al. [14] la grande majorité de la littérature sur le cross-docking ne tient pas compte simultanément des décisions locales au cross-dock et des décisions plus globales de routage. En particulier, les problèmes de tournées de véhicules avec cross-docking considèrent la capacité de traitement simultanée du cross-dock comme infinie [104, 75]. L'ordonnancement des opérations au cross-dock est alors un problème séparé, résolu a posteriori. Or dans la littérature dédiée, cette hypothèse de capacité infinie n'est pas systématique [100]. Nous proposons de rajouter une contrainte sur la capacité de traitement simultanée du cross-dock dans le VRPCD et définissons ainsi le problème de tournées de véhicules avec cross-docking et contraintes de ressources (Vehicle Routing Problem with Cross-Docking and Dock Resource constraints - VRPCDDR). Cette contrainte supplémentaire correspond à une contrainte de synchronisation sur les ressources, que l'on retrouve dans plusieurs problèmes de tournées de véhicules, généralement lorsqu'un équipement spécial est requis [32, 48, 90, 35].

Nous proposons une matheuristique inspirée de celle proposée pour le VRPCD. Une attention particulière a été portée au problème d'ordonnancement associé car, dans ce cas particulier, il est NP-Difficile. Deux techniques de vérification des contraintes temporelles ont été proposées : par des heuristiques de planification et par un modèle de programmation par contraintes. On observe que l'approche SPM développée pour le VRPCD implique dans ce cadre un sous-problème trop complexe. Il est préférable de mémoriser les trajets entiers des véhicules (collecte et livraison) et de résoudre uniquement un problème d'ordonnancement en sous-problème dans le branch-and-check.

Chronologiquement cette contribution a été la quatrième de cette thèse.

1.3.4 Un problème réel de chargement 3D

Cette contribution est le résultat d'un stage de cinq mois effectué au sein du JDA Labs de la société JDA Software à Montréal entre décembre 2014 et avril 2015. Chronologiquement il s'agit de la troisième contribution de cette thèse. Le problème posé était un problème de chargement, qui ne s'inscrit pas dans la continuité directe du sujet de cette thèse. Toutefois les méthodes testées et l'expérience acquise pendant ce stage ont permis d'enrichir très significativement les problèmes de tournées de véhicules avec cross-docking mentionnés précédemment : le développement des méthodes de branch-and-check en est directement issu.

Le problème posé est un problème de chargement de container pour un grand fabricant de pneus : étant donné un ensemble de pneus de matériel agricole, existe-t-il un plan de chargement respectant les règles de chargement et de sécurité ? Ce problème se distingue des autres problèmes de chargement 3D de la littérature car les formes à charger ne sont pas des boites rectangulaires mais des cylindres. Par ailleurs, il existe une très grande variété de tailles de pneus de matériel agricole, les chargements sont ainsi très différents d'une instance à l'autre.

La méthode de résolution proposée est une méthode en deux phases. Dans un premier temps, on génère l'ensemble des structures (soit le regroupement de plusieurs pneus) *intéressantes* puis un problème de couverture par ensembles visant à minimiser la surface est résolu. Les structures sélectionnées sont ensuite positionnées dans le container par un algorithme de programmation dynamique. Celui-ci exploite certaines caractéristiques du

problème et permet d'obtenir de bonnes performances dans des temps jugés acceptables par l'entreprise.

La méthode a été testée sur des instances réelles, où elle a prouvé son efficacité.

1.4 Perspectives

Le travail autour du 2E-MTVRPSS s'inscrit dans la perspective de nouveaux systèmes de distribution, principalement en ville. En pratique, l'espace urbain est soumis à de nombreux aléas (trafic, stationnement, ...). Du fait de l'interdépendance, tout retard pourrait se répercuter sur l'ensemble de la solution. Une perspective de recherche est de travailler à la robustesse des solutions proposées. Une autre direction de recherche est de travailler à une extension dynamique des méthodes proposées, avec un accent particulier sur le temps de recalcul car, malgré nos efforts, les temps de calcul sont encore importants.

Une extension possible du VRPCDDR serait d'étendre l'intégration des décisions liées aux tournées de véhicules et les décisions liées aux opérations au cross-dock dans le cas d'un réseau de cross-docks. En effet, à l'heure actuelle, la pratique la plus répandue est de séparer les décisions tactiques (le routage) des décisions opérationnelles (opérations aux cross-docks), pourtant être capable d'intégrer les deux niveaux de décisions semble être une piste d'amélioration intéressante comme évoqué dans [37, 14].

Par les contributions sur la synchronisation temporelle exacte, sur la synchronisation de ressources, et l'apport méthodologique du SPM, cette thèse permet de se rapprocher un peu plus de l'objectif, identifié par Drexl [31], d'une métaheuristique pour résoudre des instances de tailles réelles pour les problèmes de tournées de véhicules avec contraintes de synchronisation multiples. La gestion de l'espace des solutions lié à la synchronisation de mouvements est sans doute la dernière difficulté majeure.

D'un point de vue algorithmique, la méthode proposée pour le VRPCD, en particulier le branch-and-check, pourrait être adaptée sur des problèmes comme le multi-trip VRP ou le problème de collectes et livraisons avec transferts.

2

Un algorithme de recherche adaptative pour le problème de tournées de véhicules à deux échélons, multi-trip et synchronisation aux satellites

Dans ce chapitre, nous étudions un problème de tournées de véhicules à deux échelons se présentant en particulier dans le cas de la distribution de marchandises en ville. Des requêtes, situées initialement dans un centre de distribution urbain en périphérie de la ville, doivent être livrées à des clients situés dans la ville. Deux flottes de véhicules sont utilisées : de gros camions pour transporter les marchandises depuis l'entrepôt jusqu'aux boulevards de ceinture, puis de petits véhicules pour les derniers kilomètres à l'intérieur de la ville. Les requêtes sont transférées du premier échelon (gros véhicules) au second échelon (petits véhicules) dans des lieux appelées *satellites*. Nous considérons le cas d'une contrainte de synchronisation *exacte* entre les véhicules du premier et du deuxième échelon. Pour davantage de réalisme, nous intégrons différentes caractéristiques : des fenêtres de temps, la possibilité pour un petit véhicule de réaliser plusieurs tournées au cours de la journée et de s'approvisionner à n'importe quel satellite. Le problème ainsi défini est le *problème de tournées de véhicules à deux échelons, multi-trip et synchronisation aux satellites* (Two Echelon Multi-Trip Vehicle Routing Problem with Satellite Synchronization - 2E-MTVRPSS). Nous proposons un algorithme de recherche adaptatif à voisinage large pour le 2E-MTVRPSS, avec notamment des opérateurs de retraits spécifiques et plusieurs stratégies de réduction du temps de calcul.

Un article basé sur ce chapitre a été accepté pour publication dans European Journal Of Operations Research : Grangier, P., Gendreau, M., Lehuédé, F., Rousseau, L.-M., *An adaptive large neighborhood search for the two-echelon multiple trip vehicle routing problem with satellite synchronization*.

2.1 Article I: an adaptive large neighborhood search for the two-echelon multiple trip vehicle routing problem with satellite synchronization

Abstract

The two-echelon vehicle routing problem (2E-VRP) consists in making deliveries to a set of customers using two distinct fleets of vehicles. First-level vehicles pick up requests at a distribution center and bring them to intermediate sites. At these locations, the requests are transferred to second-level vehicles, which deliver them. This paper addresses a variant of the 2E-VRP that integrates constraints arising in city logistics such as time window constraints, synchronization constraints, and multiple trips at the second level. The corresponding problem is called the two-echelon multiple-trip vehicle routing problem with satellite synchronization (2E-MTVRP-SS). We propose an adaptive large neighborhood search to solve this problem. Custom destruction and repair heuristics and an efficient feasibility check for moves have been designed and evaluated on modified benchmarks for the VRP with time windows.

2.2 Introduction

The two-echelon vehicle routing problem (2E-VRP) consists in routing freight from a central depot to customers through a set of intermediate sites. The depot is an intermodal logistics site called the distribution center (DC). It has some storage capacity, and it is where consolidation takes place. Intermediate sites, usually called satellites, have little or no storage capacity but are located closer to customers. Two fleets of vehicles are involved: first-level vehicles carry requests from the DC to the satellites, and second-level vehicles carry requests from the satellites to the customers. First-level vehicles are usually significantly larger than second-level vehicles.

Over the last few years freight transportation in urban areas has received much attention [15]. Indeed, because of increasing traffic congestion, environmental issues, and low average truckloads, new policies (e.g., London Congestion Charges, Monaco UDC) and initiatives (Amsterdam City Cargo) have emerged to ban large trucks from city centers. This movement is known as *city logistics* and represents a move from independent direct shipping strategies toward integrated logistics systems. In this context, multi-echelon distribution systems and particularly two-tiered systems are often proposed as an alternative to current distribution systems [25].

Several specific constraints arise in the urban context: time windows, multiple use of vehicles, and synchronization. Delivery hours are often restricted because of customer requirements or city regulations. Moreover, second-level vehicles are usually small in order to access every street, so even a full load does not represent an entire work-day. Finally, operating a satellite in a city is expensive, because of labor costs and high rents. More and more cities allow transporters to use dedicated or existing infrastructures (reserved parking spaces, bus depots) to unload [24]. No storage capacity is normally available at these locations, thus requiring a synchronization of the two levels.

The contribution of this paper is a solution methodology for a 2E-VRP that integrates constraints that have not yet been addressed in the literature: time windows, synchro-

nization, and multiple trips. Similar problems have been discussed in [74] under the name *two-echelon vehicle routing problem with satellite synchronization* (2E-VRP-SS) and modeled in [25] under the name *two-echelon, synchronized, scheduled, multidepot, multiple-tour, heterogeneous VRPTW* (2SS-MDMT-VRPTW). However, to the best of our knowledge, no implementation has been reported.

Related work includes models for city logistics, multi-echelon vehicle routing problems with multiple routes and transfer or synchronization constraints. A general model for city logistics systems is presented by Crainic et al. in [25], while Mancini focuses on multi-echelon systems [60]. The 2E-VRP was introduced by Perboli et al.[73], who proposed a mathematical model. Since then several algorithms have been developed: math-based heuristics [74, 73], clustering-based heuristics [20], GRASP [22, 21, 105], adaptive large neighborhood search (ALNS) [47], and a large neighbourhood search combined with a local search [13]. Exact methods include [53, 80, 83, 86]. Crainic et al. [23] study the impact of satellite location on the cost of a 2E-VRP solution compared to that of a VRP. Cuda et al. [26] recently published a survey on two-echelon routing problems. A similar problem is the two-echelon location routing problem (2E-LRP) [71]. Our problem also integrates some multiple-trip aspects [96] that have been solved with tabu search [70], ALNS [6] and iterated local search [16]. Synchronization of multiple trips supplied by a single bus line as been studied in [66] in a city logistics context. We refer to [31] for a detailed survey of synchronization in vehicle routing problems and to [15] for a recent survey of vehicle routing problems in city logistics.

The 2E-MTVRP-SS can be considered as a particular Pickup and Delivery Problem with Transfers (PDPT) which has been recently studied in [79, 63, 64]. The major differences are that in the 2E-MTVRP-SS transfers are mandatory, routes should be designed for two types of vehicles which do not share the same network, and that two vehicles must be simultaneously present at a satellite during a transfer. In this paper, we extend the previous approaches to integrate those differences and we exploit the specificities of the 2E-MTVRP-SS to propose a better exploration of the search space, as well as a more compact graph representation of temporal constraints.

The remainder of this paper is organized as follows. Section 2.3 presents a formulation of the problem, and Sections 2.4 and 2.5 are devoted to the solution method with a special focus on efficiently solving the timing subproblem. Computational results are presented in Section 3.6.

2.3 Problem formulation

In this section we define the problem and discuss the synchronization model at satellites.

2.3.1 Problem statement

We introduce the two-echelon multiple-trip vehicle routing problem with satellite synchronization (2E-MTVRP-SS). We consider a city distribution center (CDC), a set of satellites V_s, a set of requests R, and two homogeneous fleets of vehicles K_1 and K_2 of capacity Q_1 and Q_2, based at o_1 and o_2. Each request r is located at the CDC at the beginning of the time horizon and must be delivered within the time window $[e_r, l_r]$ to a customer denoted by d_r (the set of customers is denoted by V_c). The quantity associated with r is q_r. No direct shipping from the CDC is allowed. Second-level vehicles can perform multiple trips, which may start at different satellites. As second-level vehicles are small we assume that

they are empty every time they arrive at a satellite, a similar assumption is made in [25]. Satellites have no storage capacity, thus requiring an exact synchronization between the vehicles of the two levels.

The 2E-MTVRP-SS is defined on a directed graph $G = (V, A)$, which reflects the two-level system. The first level is defined by $G_1 = (V_1, A_1)$ with $V_1 = \{o_1\} \cup \{CDC\} \cup V_s$ and $A_1 = \{(o_1, CDC)\} \cup \{(CDC, i) | i \in V_s\} \cup \{(i, j) | i, j \in V_s\} \cup \{(i, o_1) | i \in V_s\}$. The second level is defined by $G_2 = (V_2, A_2)$ with $V_2 = \{o_2\} \cup V_c \cup V_s$ and $A_2 = \{(o_2, i) | i \in V_s\} \cup \{(i, j) | i \in V_s, j \in V_c\} \cup \{(i, j) | i, j \in V_c\} \cup \{(i, j) | i \in V_c, j \in V_s\} \cup \{(i, o_2) | i \in V_c\}$. With each arc $(i, j) \in A = A_1 \cup A_2$ is associated a travel time $t_{i,j}$ and a travel cost $c_{i,j}$. Each node i has a known service duration s_i. Solving the 2E-MTVRP-SS involves finding $|K_1|$ first-level routes and $|K_2|$ second-level routes, and a schedule for them, such that the capacity and time-related constraints are satisfied.

2.3.2 Transfer and synchronization at satellites

We define a transfer as the operation during which a first-level vehicle transfers one or more requests to a second-level vehicle at a satellite. Given the lack of storage capacity at the satellites, the two vehicles must be at the satellite at the same time. Thus, the first and second levels must be synchronized. In details, for a transfer to happen, the two vehicles:

- may need time to get ready for the transfer (for example if the first-level vehicle has a lift gate to open),
- should spend some time transferring the items,
- should get ready to leave the satellite (for example the second-level vehicle may have to sort items).

Figure 2.1 illustrates the temporal aspects of a transfer. In this example, if $t_2 > t_1 + s_R$, the first-level vehicle must wait. Conversely, if $t_2 < t_1 + s_R$ the second-level vehicle must wait for the first-level vehicle. If the first-level vehicle transfers requests to several second-level vehicles, it cannot leave before $\max_{i \in K_2} t_i + s_T$. In this paper, we assume that s_R, s_T, and s_L can reasonably be considered independent of the transferred quantity, without inducing a significant imprecision. This simplifying hypothesis allow to integrate those times into the travel times from and to the satellites. Thus, we later consider that all the transfer-related periods (s_R, s_T, s_L) are equal to zero. If a second-level vehicle returns several times to pick up requests from the same first-level vehicle at the same satellite, each visit corresponds to a different transfer.

2.3.3 Mathematical formulation

We present a mixed integer linear programming formulation for the 2E-MTVRP-SS. In the model, to represent the transfer of one request at one satellite, for each request r and satellite s, we create a node $v_{s,r}$ whose associated demand is $-q_r$, and we denote by $\tilde{V}_s = \{v_{i,r} | v_i \in V_s, r \in R\}$ all the satellites. For each vehicle k, we create a start node o_k and an end node o'_k ($\tilde{O} = \cup_k o_k$, $\tilde{O}' = \cup_k o'_k$).

The mathematical formulation is defined on a graph $G^{\text{math}} = (V^{\text{math}}, A^{\text{math}})$. The first level is $G_1^{\text{math}} = (V_1^{\text{math}}, A_1^{\text{math}})$ with $V_1^{\text{math}} = \tilde{O}_1 \cup \tilde{O}'_1 \cup \tilde{V}_s$ and $A_1^{\text{math}} = \{(o, i) | o \in \tilde{O}_1, i \in \tilde{V}_s\} \cup \{(i, j) | i, j \in \tilde{V}_s\} \cup \{(i, o') | i \in \tilde{V}_s, o' \in \tilde{O}'_1\} \cup O_1 \times \tilde{O}_1$. The second level is

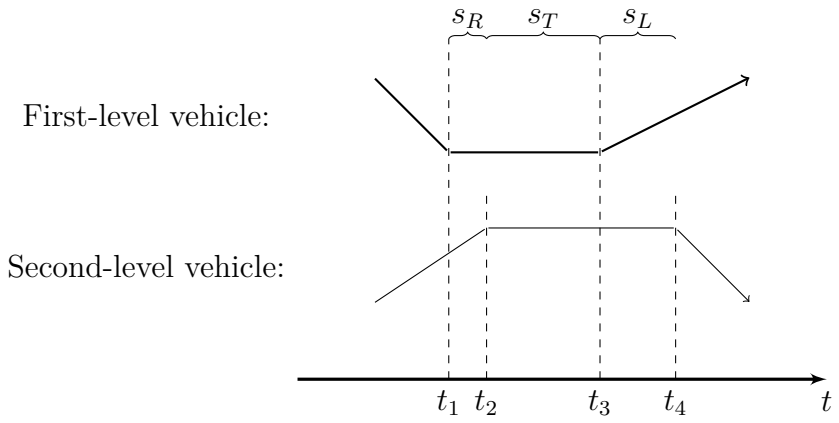


Figure 2.1 – Time chart for a transfer

s_R $[t_1, t_2]$: Preparation time for the first-level vehicle.

s_T $[t_2, t_3]$: Time in common for the vehicles of both levels. For a transfer to occur, the two vehicles should spend at least this time together at the satellite.

s_L $[t_3, t_4]$: Sorting time for the second-level vehicle.

$G_2^{\text{math}} = (V_2^{\text{math}}, A_2^{\text{math}})$ with $V_2^{\text{math}} = \tilde{O}_2 \cup \tilde{O}_2' \cup V_c \cup \tilde{V}_s$ and $A_2^{\text{math}} = \{(o,i)|o \in \tilde{O}_2, i \in \tilde{V}_s\} \cup \{(i,j)|i \in \tilde{V}_s, j \in V_c\} \cup \{(i,j)|i,j \in V_c\} \cup \{(i,j)|i \in V_c, j \in \tilde{V}_s\} \cup \{(i,o')|i \in V_c, o' \in \tilde{O}_2'\} \cup O_2 \times \tilde{O}_2$. We introduce the following variables:

$$x_{i,j}^k = \begin{cases} 1 & \text{if vehicle } k \text{ travels from node } i \text{ to node } j \\ 0 & \text{otherwise} \end{cases}$$

$h_i =$ service time at node i (point in time when service at node i starts)

$u_i =$ load of the second-level vehicle after serving i.

We introduce two constants: M_h corresponds to the end of the planning horizon, and M_u is the sum of all the ordered quantities.

We use a classical approach in vehicle routing problems with time windows, which consists in lexicographically minimizing the fleet-size and the routing cost. The first objective is the number of first-level vehicles, the second objective is the number of second-level vehicles and the third objective is the sum of arc costs.

$$lex - min(\sum_{k \in K_1} \sum_{j \in \tilde{V}_s} x_{o_k,j}, \sum_{k \in K_2} \sum_{j \in \tilde{V}_s} x_{o_k,j}, \sum_{k \in K_1} \sum_{(i,j) \in A_1^{\text{math}}} c_{i,j} \times x_{i,j}^k + \sum_{k \in K_2} \sum_{(i,j) \in A_2^{\text{math}}} c_{i,j} \times x_{i,j}^k) \tag{2.1}$$

s.t.

$$\sum_{(o_k,j)\in A_e^{\text{math}}} x^k_{o_k,j} = \sum_{(j,o'_k)\in A_e^{\text{math}}} x^k_{j,o'_k} = 1 \qquad \forall e \in \{1,2\},\ \forall k \in K_e \tag{2.2}$$

$$\sum_{(i,j)\in A_e^{\text{math}}} x^k_{i,j} = \sum_{(j,i)\in A_e^{\text{math}}} x^k_{j,i} \qquad \forall e \in \{1,2\}, \forall k \in K_e, \forall i \in V_e^{\text{math}} \setminus (\tilde{O}_e \cup \tilde{O}'_e) \tag{2.3}$$

$$\sum_{k\in K_2} \sum_{(j,d_r)\in A_2^{\text{math}}} x^k_{j,d_r} = 1 \qquad \forall r \in R \tag{2.4}$$

$$\sum_{k\in K_e} \sum_{j\in V_{s,r}} \sum_{(i,j)\in A_e^{\text{math}}} x^k_{i,j} = 1 \qquad \forall e \in \{1,2\}, \forall r \in R \tag{2.5}$$

$$\sum_{k\in K_1} \sum_{i\in V_1^{\text{math}}} x^k_{i,v_{s,r}} = \sum_{k\in K_2} \sum_{i\in V_2^{\text{math}}} x^k_{i,v_{s,r}} \qquad \forall v \in V_s, \forall r \in R \tag{2.6}$$

$$\sum_{v\in V_{s,r}} \sum_{(i,v)\in A_2^{\text{math}}} x^k_{i,v} = \sum_{(j,d_r)\in A_2^{\text{math}}} x^k_{j,d_r} \qquad \forall r \in R, \forall k \in K_2 \tag{2.7}$$

$$h_j \geq h_i + s_i + t_{i,j} - M_h \times (1 - \sum_{k\in K_e} x^k_{i,j}) \qquad \forall e \in \{1,2\}, \forall (i,j) \in A_e^{\text{math}} \tag{2.8}$$

$$h_{o_k} \geq 0 \qquad \forall k \in K \tag{2.9}$$

$$e_i \leq h_i \leq l_i \qquad \forall i \in V_c \tag{2.10}$$

$$h_{o'_k} \leq M_h \qquad \forall k \in K \tag{2.11}$$

$$\sum_{r\in R} q_r \times \sum_{j\in V_{s,r}} \sum_{(i,j)\in A_1^{\text{math}}} x^k_{i,j} \leq Q_1 \qquad \forall k \in K_1 \tag{2.12}$$

$$u_j \geq u_i - q_i - M_u \times (1 - \sum_{k\in K_2} x^k_{i,j}) \qquad \forall (i,j) \in A_2^{math} \tag{2.13}$$

$$u_j \leq M_u \times (1 - \sum_{k\in K_2} \sum_{i\in V_2^{\text{math}}\setminus \tilde{V}_s} x^k_{i,j}) \qquad \forall j \in \tilde{V}_s \tag{2.14}$$

$$0 \leq u_i \leq Q_2 \qquad \forall i \in V_2^{\text{math}} \tag{2.15}$$

$$x^k_{i,j} \in \{0,1\} \qquad \forall k \in K, (i,j) \in A^{\text{math}} \tag{2.16}$$

The objective function (2.1) lexicographically minimizes the fleet-size and the travel costs. Constraints (2.2) state that each vehicle must start and end its route at its base. Constraints (2.3) are flow conservation constraints. Constraints (2.4) ensure that each request is delivered. Constraints (2.5) ensure for each request that only one transfer node is used, and (2.6) ensure that it is visited by both a first and a second-level vehicle. Constraints (2.7) ensure for each request that the second-level vehicle that visits the transfer node is the one that delivers the request. Constraints (2.8) compute the travel time between two nodes if they are visited consecutively by the same vehicle, and constraints (2.9) handle the special case of bases for which there is no predecessor. Constraints (2.10) ensure that each request is delivered within its time window. Constraints (2.11) ensure that each vehicle has completed its route within the time horizon. Constraints (2.12) (resp. (2.15)) ensure for each first-level (second-level) vehicle that the load does not exceed the vehicle capacity. Constraints (2.13) ensure for each second-level vehicle that the load after visiting a node is equal to the load before plus (or minus) the quantity that has been loaded (unloaded). Constraints (2.14) ensure that each second-level vehicle is empty when arriving at a satellite.

2.4 An ALNS for the 2E-MTVRP-SS

In this section we describe the destruction and repair methods used in our ALNS for the 2E-MTVRP-SS. ALNS was proposed by Ropke and Pisinger [82] as an extension of the large neighborhood search introduced by Shaw [92]. The general principle of ALNS is described in Algorithm 2: it iteratively destroys and repairs the current solution using heuristics, which are selected based on their past successes. Destroying here means removing a number p of requests from the current solution (p is bounded by some parameters), while repairing means inserting unplanned requests in the solution. Note that a solution s may be *incomplete*, if a set of requests $\mathcal{L}_s$ remained unplanned. The size of this set is then penalized in the objective function. The solution obtained after the destroy and repair operations is accepted if it satisfies an acceptance criterion. As in [82], we use a roulette-wheel mechanism as adaptive layer for the selection of destruction and repair methods, and a simulated annealing as acceptance criterion.
Since its introduction, ALNS has been used to solve the 2E-VRP [47] as well as many complex vehicle routing problems [77, 54, 63, 6, 65, 9].

Data : Candidate solution s, destroy operators $\mathcal{N}^-$, repair operators $\mathcal{N}^+$
Result : The best found solution $s^\star$

$s^\star \leftarrow s$
while *stop-criterion not met* **do**
  $s' \leftarrow s$
  Destroy quantity: select a number p of requests to remove from the candidate solution
  Operator selection: select a destruction method $or \in \mathcal{N}^-$ and a repair method $oi \in \mathcal{N}^+$
  Removal: $\mathcal{L} \leftarrow \mathcal{L}_{s'} \cup or(s', \mathrm{p})$
  Insertion: $s' \leftarrow oi(s', \mathcal{L})$
  if $f(s') \leq f(s^\star)$ **then**
    $s^\star \leftarrow s'$
  end
  Acceptance criterion: set the candidate solution s to s'
end
return $s^\star$

Algorithm 2 : Adaptive Large Neighborhood Search

We refer to [82] for a more detailed explanation of ALNS. In the following, we focus on the specific components of our method, namely the construction of the initial solution, and the destroy and repair methods.

2.4.1 Initial solution

We design a two-phase constructive algorithm to obtain the initial solution. First, we design the second level routes using a best insertion algorithm for a multiple-trip multiple-depot problem. In this heuristic the possible insertion of a customer are all the insertions in existing trips and all insertions by creation of a new trip. Then we create the first-level routes to supply the satellites according to the second-level routing plan previously created.

2.4.2 Destroy methods

When partially destroying a solution we select a method and a number p of requests to remove. Unless stated otherwise, this method is reused until p is reached. Following Azi et al. [6], we use three levels of destruction methods: workday, route, and customer.

Workday level

The following operators are used for first and second-level vehicles.
Random Vehicle Removal: we randomly remove a vehicle.
Least Used Vehicle Removal: we remove the vehicle with the smallest load. For the second level, the total load of a vehicle is defined as the sum of the load of each trip.

Route level

Random Trip Removal: we randomly remove a trip from the solution.
First-level Stop Removal: we randomly remove a first-level stop from the solution. The trips that get their requests from this stop are removed.
Trip Related Removal: this method is similar to that of Azi et al. [6]. Trips are removed based on a proximity measure: we start by randomly selecting a trip and removing it. We then find the trip that contains the nearest customer to any customer in the trip just removed, and we remove that trip.
Synchronization-Based Trip Removal: intuitively, a *good* synchronization occurs when the vehicles involved arrive at approximately the same time. If a second-level vehicle arrives a long time before (or after) the first-level vehicle there will be a long waiting time; this should be avoided. This method removes the trip for which the time between the arrival of the second-level vehicle and the arrival of the first-level vehicle is maximum.

Customer level

Random Customer Removal: we randomly remove a customer.
Worst Removal: this operator is the same as in [82]. For each request we compute the difference in cost of the solution with and without this request. We then sort the requests from the largest to the smallest difference. And we remove the *worst* (largest difference) request. Some randomization is introduced to avoid repeatedly removing the same requests.
Related Removal Heuristics: these methods aim to remove related requests. Let the relatedness of requests i and j be $R(i, j)$. We use two distinct relatedness measures: distance and time. The distance measure is the distance between the delivery points of i and j. The time measure is the sum of the absolute gap between their start of service and the absolute gap between their latest delivery times. Each measure is normalized by dividing it by the longest distance (resp. travel time) between two customers. In both cases a lower $R(i, j)$ value indicates a great degree of relatedness.

We ran preliminary tests to compare these two measures with that of Shaw [92], which groups time and distance into a single measure. The methods gives similar results, but we chose time and distance because they do not require parameter tuning.
History-based Removal: This is inspired by [63] and removes requests that seem poorly placed in the current solution with regard to the best-known solutions. For requests r and r', let $\xi_{r,r'}$ be the number of solutions among the 50 best-known in which r' is a direct successor of r. For each request, let $\delta^-(r)$ (resp. $\delta^+(r)$ be its direct predecessor (resp.

successor). For request r and satellite s, let $\chi_{s,r}$ be the number of solutions in which r is delivered via a transfer at s. For each request r, delivered in the current solution via a transfer at s, we define a score ϕ as follows:

$$\phi_r = \xi_{\delta^-(r),r} + \xi_{r,\delta^+(r)} + \chi_{s,r}.$$

Then we remove the p requests with the lowest scores.

2.4.3 Repair methods

In this section we describe the methods used to repair a solution. We first describe the three different ways to insert a given customer into a given second-level route that we use, and then we describe the repair methods.

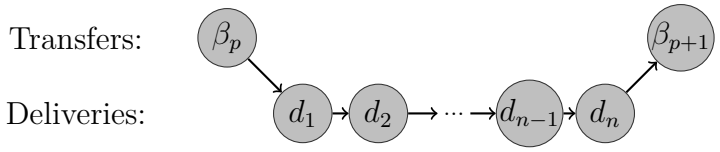
Three insertion operators

In the VRP, the insertion of a customer c into a partially built solution is fully described by giving the route and the position for the insertion. Thus, all possible insertions can be described by the unique set $\{(k,i) : \ k \in \text{Vehicles}, \ 0 \leq i \leq |route(k)| + 1\}$. Given the multiple-trip and two-echelon characteristics of the 2E-MTVRP-SS, we consider three distinct greedy ways of inserting a customer into a solution. We call them *insertion operators* and describe them below.

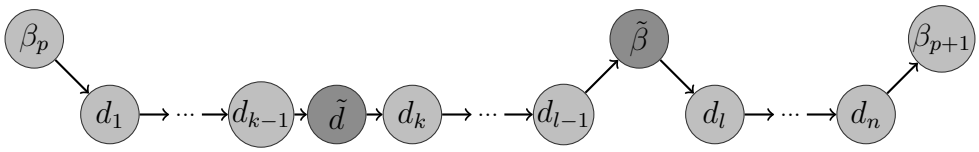
Insertion into an existing trip: The customer is inserted into an existing trip.

Insertion by creation of a new trip: A new trip is created for the customer. This new trip can be connected either to an existing stop or to a newly created stop of a first-level vehicle.

Insertion by trip split: Before inserting c into trip t, we split t into two trips, t_1 and t_2. Trip t_1 is still connected to the same first-level stop as t, but t_2 is connected either to an existing stop or to a newly created stop of a first-level vehicle. This vehicle can be different from that involved in t_1. Then c is inserted into one of the two resulting trips. Figure 2.2 illustrates the use of a trip split operator.



(a) Example of a second-level trip; β are transfer points and d_i are delivery points.



(b) Insertion into the first trip

Figure 2.2 – Example of the use of a trip split operator with insertion of the request into the first resulting trip.

Reducing the size of the neighborhoods

The insertion of a request corresponds to two decisions: one at the first level and one at the second level. Thus, the neighborhoods generated by the insertion operators are huge. For example, to test every possible insertion of a request r with the *trip split* operator, we have to test for each pair (*insertion position*, *split position*) in each trip of second-level vehicles, each existing stop of first-level vehicles, and each creation of a stop (i.e., each satellite at each position). To ensure a reasonable runtime, we have created restricted neighborhoods.

For the *trip split* we have introduced two variants: *existing stops* and *customer first*. In the *existing stops* variant, we only try to connect t_2 with an existing stop of a first-level vehicle. In the *customer first* variant, we first select the insertion position that leads to the smallest increase in the cost of the global solution. Then we select the best possible way to split t into feasible trips t_1 and t_2, trying both existing and newly created first-level stops.

When we create a new first-level stop to be connected to a second-level trip, it is likely that the best choice for the satellite will be close to the second vehicle. At the beginning of our algorithm we sort the satellites in order of distance for every pair of customers. When creating a new trip connected to a new first-level stop, we consider only the s satellites closest to the pair (predecessor of the trip, first customer in the trip). This is used for the *trip creation* and the variants of the *trip split* operator.

As shown in Section 2.6.3, using these restricted neighborhoods makes our algorithm about 2.3 times faster while maintaining the quality of the solution.

Repair methods

All the unplanned requests are stored in a request bank.

Best insertion: From the requests in the request bank, we insert the one with the cheapest insertion cost considering all possible insertion operators.

K-Regret: For each request in the request bank, let δ_r^i represent the gap between the insertion of r at its best position in its best trip and the insertion at its best position in its i^{th} best trip. We select the request where $\sum_{i=2}^{k} \delta_r^i$ is maximum. In other words, we maximize the sum of the differences of the cost of inserting request r into its best trip and its i^{th} best trip. To control the computational time, we use small values of k.

First-level routes

Inserting a request r by moving a first-level vehicle to a new satellite generates a large increase in the routing cost compared to transferring the request at an existing transfer point. Thus, it is rare for repair methods to choose such insertions. However, subsequent insertions may benefit from a new transfer point, because the second-level vehicle may have a smaller distance to travel. Therefore, when an insertion operator creates a new stop, we consider the following *biased* cost:

$$\begin{aligned}\text{Biased cost} = {} & \text{second-level insertion cost} \\ & + \text{first-level insertion cost} \times \max\left(\alpha, \frac{\text{load in second-level trip}}{\text{second-level vehicle capacity}}\right). \quad (2.17)\end{aligned}$$

In this biased cost, we acknowledge that if there is some room in the second-level trip, then it is likely that we will later use it for a customer. For our instances, after some

tuning, we have used $\alpha = 0.7$ for the *trip creation* operator and $\alpha = 0$ for the *trip split* operator and its variants.

2.5 Route scheduling and feasibility algorithm

For each performed insertion, repair methods evaluate thousands of insertions both in terms of profitability and feasibility. In this section we describe an efficient way to test if an insertion is feasible with respect to the temporal constraints of the problem. The proposed method is an adaptation of the feasibility algorithm designed by Masson et al [64] for the PDPT, which was based on the forward time slacks [88]. The main idea behind forward time slacks is first introduced in 2.5.1, then a way to model time constraints is presented in 2.5.2, and the efficient feasibility tests are detailed in 2.5.3. All along this section, we use the notation in Table 2.1.

Notation	Definition
$\psi_{u,v}$	ordered set of vertices on path $(u, \dots, v)$
$\delta^+(i)$	direct successors of a vertex i
$\Gamma^+(i)$	set of all successors of i
$\Omega_{u,v}$	set of paths from u to v

Table 2.1 – Notation in the temporal graph

2.5.1 Efficient feasibility test for the vehicle routing problem with time windows

Inserting a request into a route of a feasible VRP solution may postpone several deliveries later in this route, potentially violating time windows. For a given insertion, a way to check if such a violation occurs would be to reschedule downstream operations while taking into account the postponement created by the insertion. Such method has a linear complexity in the size of the route, but Savelsbergh [88] introduced a method that checks in constant time if an insertion generates a time-window violation in an *as early as possible* route schedule. It comes at the price of recomputing some coefficients every time an insertion is performed, but this proved to be a very efficient trade off, as insertion based algorithms usually test many insertions before actually performing one.
Savelsbergh's method computes for each delivery the maximum possible forward shift, its *forward time slack* (FTS), that does not lead to a time-window violation later in the route.

In detail, let u and v be two nodes in the same route, with v being delivered after u. For all vertices i in the route, let h_i be the current time of service at i, w_i be the waiting time before service and l_i be the latest time for service at vertex i. We define the total waiting time on path $(u, \dots, v)$ as

$$TWT_{\psi_{u,v}} = \sum_{i \in \psi_{\delta^+(u),v}} w_i. \tag{2.18}$$

The FTS at node u is

$$F_u = \min_{i \in \{u\} \cup \Gamma^+(u)} \{TWT_{\psi_{u,i}} + l_i - h_i\} \tag{2.19}$$

An intuitive explanation of formula (2.19) is the following. For every successor i of u, the path $(u, \ldots, i)$ can be decomposed into a working time and an idle/waiting time. If u is postponed, first the idle time will be reduced and then the time of service at node v will be postponed. So the margin for postponement of v that does not violate the time window at i is the total waiting time between u and i plus the difference between the current start time at i and the end of its time window. This should be true for every indirect successor of v thus the min.
And if the start time of service of u is postponed by δ, then the new start time of service of v, $\tilde{h_v}$, can be computed

$$\tilde{h_v} = h_v + \max(0, \delta - TWT_{u,v}) \tag{2.20}$$

By simply comparing the shift of the delivery right after the insertion with its forward time slack, we get a constant time feasibility check.

2.5.2 Modeling time constraints in the 2E-MTVRP-SS

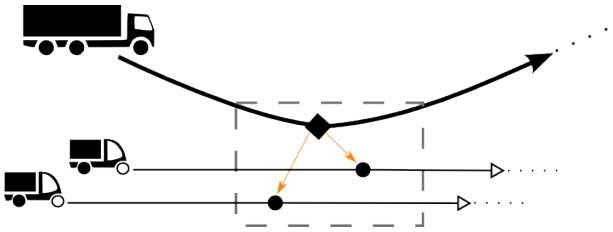
Contrary to the VRP, in problems with synchronization, a change in the schedule of one route may have effects on other routes, potentially making them timewise infeasible. For example, in the 2E-MTVRP-SS, if we insert a delivery in a trip of a second-level vehicle A, it may arrive later to its next transfer, thus delaying the first level-vehicle it is synchronized with. In turn, the first-level vehicle will spread its delay to second-level vehicles at its next transfer, and so on, eventually causing a time windows violation for a second-level vehicle B whose link with A was not obvious at first sight. This is known as *interdependence problem* [31], and we model it through a precedence graph.

Precedence graph definition

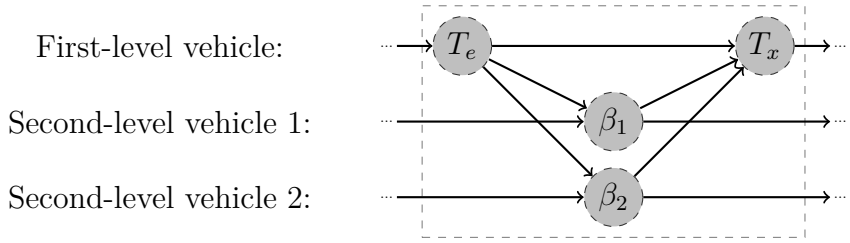
Given the routes in a solution of the 2E-MTVRP-SS, we can represent the time constraints as a directed acyclic graph, G_t. We refer to it as a *precedence graph* and it is built as follows: for every operation except transfers, we create a node and add an arc to each of its direct successors in the given route. Each arc (u, v) has a weight that corresponds to $t_{u,v}$. For a transfer we create three nodes: a transfer entrance node T_e, a transfer exit node T_x for the first-level vehicle, and a pick-up node β for the second-level vehicle. We create three arcs (T_e, T_x), (T_e, β), (β, T_x) with weight 0. If the first-level vehicle transfers its load to several second-level vehicles, we create only one pair (T_e, T_x). We assume that transfers to second-level vehicles can occur simultaneously. Figure 2.3 illustrates this transformation.

Route scheduling

G_t corresponds to a PERT chart. As mentioned in [17] (p. 657), scheduling tasks in such a diagram can be performed using a shortest-path algorithm, with linear complexity in the number of vertices. Furthermore, in an as-early-as-possible schedule, after a change, only the downstream operations have to be rescheduled, thus reducing the number of modifications to be performed.



(a) First-level stop with two transfers



(b) Precedence graph representation.

Figure 2.3 – A first-level stop with two transfers and its precedence graph representation.

2.5.3 Using the precedence graph for efficient feasibility testing

With G_t we would be able to check if an insertion is timewise feasible by rescheduling downstream operations. But this would be even more time consuming than in the VRP, as it would be linear in the number of operations in the solution (versus number of operations in one route in the VRP). Masson et al. [64] were faced with a similar problem for the PDPT. They introduced a directed acyclic graph to model their precedence constraints and extended the FTS obtaining a constant time feasibility check for time constraints. Hereafter we present their main results and how they can be applied to the 2E-MTVRP-SS.

Extension of FTS to directed acyclic graph modeling precedence constraints [64]

First, Masson et al. introduced the notion of *slack time*. It is a generalization of the total waiting time between two nodes:

$$ST_{u,v} = \min_{w \in \Omega_{u,v}} TWT_w.$$

Then the FTS at node u becomes:

$$F_u = \min_{i \in \{u\} \cup \Gamma^+(u)} \{ST_{u,i} + l_i - h_i\}. \tag{2.21}$$

The intuitive explanation given for equation (2.19) still holds for equation (2.21), with slack times accounting for the fact they may exist several time paths between two vertices. Masson et al. proved the above result in their Proposition 2. Their proof does not rely on the particular structure of their precedence graph for the PDPT, it is thus valid for any directed acyclic graph. As such, this result (as well as the one hereafter on start of service) is valid for our precedence graph.
Similarly to the VRP, if the start of service time of node u is postponed by δ, the new start of service time $\tilde{h}$ of any of its sucessor v, is

$$\tilde{h_v} = h_v + max(0, \delta - ST_{u,v}) \tag{2.22}$$

Note that in contrast to Masson et al., we recompute FTS using the Floyd–Warshall algorithm ([17], p. 693), which is faster in our case than the suggested shortest path method.

Efficient feasibility test for the 2E-MTVRP-SS

When evaluating an insertion of an unplanned request into a feasible solution, we need to ensure that the solution will remain timewise feasible after the insertion. With the insertion operators of Section 2.4.3, timewise infeasibility can occur in two ways: creating a cycle of precedence constraints or violating a time window.

Cycle detection Some insertion operators create new transfers, which may lead to infeasible precedence relations (see Figure 2.4 for an illustration). We extend the method of Masson et al. [64] for detecting cycles in constant time.

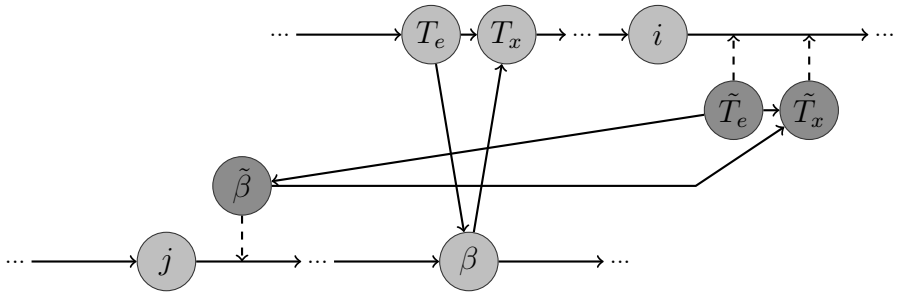


Figure 2.4 – Infeasible insertion: it creates a cycle in the precedence graph.

Proposition 1. *Synchronizing a first-level stop T and a second-level trip $(\beta \to d_1 \dots d_n)$ creates a cycle in the precedence graph if and only if*

$$T_e \in \Gamma^+(\beta) \text{ or } \beta \in \Gamma^+(T_x).$$

Proof. $\Rightarrow$ With the synchronization, only two arcs are created $(T_e \to \beta)$ and $(\beta \to T_x)$. One of them is responsible for the cycle. If it is $(T_e \to \beta)$, previously $T_e \in \Gamma^+(\beta)$. If it is $(\beta \to T_x)$, previously $\beta \in \Gamma^+(T_x)$.
$\Leftarrow$ If $T_e \in \Gamma^+(\beta)$, since $\beta \in \delta^+(T_e)$ by definition, a cycle is created. If $\beta \in \Gamma^+(T_x)$, since $T_x \in \Gamma^+(\beta)$ by definition, a cycle is created. □

Corollary 1. *Synchronizing a new first-level stop inserted after node i and a new second-level trip inserted after node j creates a cycle in G_t if and only if*

$$i \in \Gamma^+(j) \text{ or } j \in \Gamma^+(i).$$

Provided that we maintain a successor matrix, which can easily be computed together with slack times, we can check in constant time if an insertion creates a cycle in G_t.

Use of FTS The use of FTS for checking the feasibility with respect to time windows of an *insertion into an existing trip*, or an *insertion by creation of a new trip* is straightforward. For an *insertion into an existing trip*, we check if the shift of the operations of the second-level vehicle right after the insertion is smaller than its forward time slacks. For an *insertion by creation of a new trip*, we check if the shift of the operations of the second-level vehicle

right after the new delivery, and the shift of the operations of the first level vehicle right after the new transfer exit are smaller than their respective forward time slacks. The case of an *insertion by trip split* is more complex, it is illustrated in Figure 2.5 and its details are given in Algorithm 3.
Still, for all insertion operators, with FTS we get a constant time feasibility check.

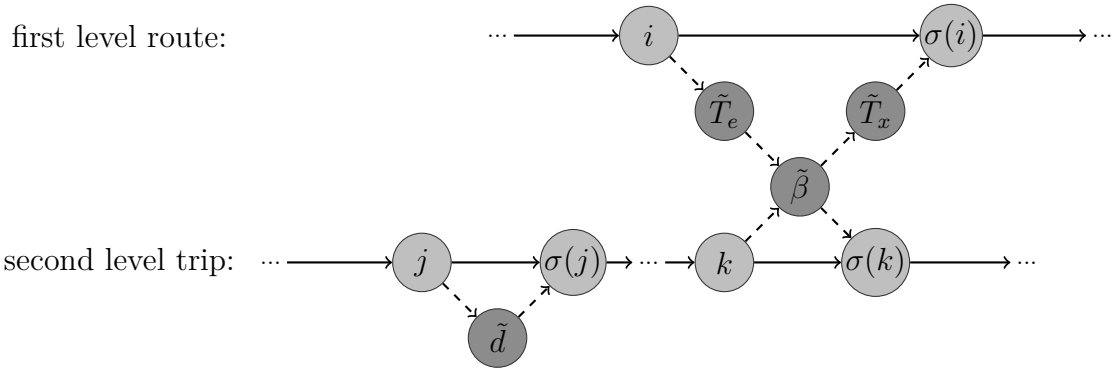


Figure 2.5 – Example of insertion with split trip in the precedence graph. Customer $\tilde{d}$ is inserted between vertices j and $\sigma(j)$ in a second level trip. This trip is split and connected at a new satellite $\tilde{\beta}$ between vertices k and $\sigma(k)$. The new trip is supplied at a new stop in a first level route, which is inserted between vertices i and $\sigma(i)$.

Efficiency of the method

We compare the runtime of the extension of FTS versus a check based on an incremental PERT: on average our algorithm is approximately 12 times faster with the FTS extension than with the PERT (see Section 2.6.3).

2.6 Computational experiments

In this section, we first describe the adaptation of some well-known VRPTW instances to the 2E-MTVRP-SS. Parameters configuration is discussed in Section 2.6.2. In Section 2.6.3 we show that our custom heuristics are efficient, and we present our results.

2.6.1 Instances

Since the 2E-MTVRP-SS is a new problem, there are no instances for it. We adapt the well-known Solomon's instances for the VRPTW [93]. We use a subset of these instances to tune our algorithm: the first two of every type, for a total of 12 tuning instances.

Geographical configuration

The customer requests are unchanged. The depot (node 0) is the base of second-level vehicles; the first-level vehicles are based at the CDC.
We adopt the following $X/Y/M/N$ notation to describe the position of the CDC and the satellites. X and Y give the position of the CDC expressed as a percentage of the size of the map. M and N describe the number of rows (resp. columns) of a grid. We locate a satellite at each exterior intersection of the grid. Figure 2.6 illustrates a -50 / 50 / 3 / 3 configuration.

Result : return **true** if the insertion illustrated in Figure 2.5 is feasible, **false** otherwise

$\bar{h}_{\tilde{T}_e} \leftarrow \max(h_i + s_i + t_{i,\tilde{T}_e}, e_{\tilde{T}_e})$
$\bar{h}_{\tilde{d}} \leftarrow \max(h_j + s_j + t_{j,\tilde{d}}, e_{\tilde{d}})$
if $\bar{h}_{\tilde{T}_e} > l_{\tilde{T}_e}$ **or** $\bar{h}_{\tilde{d}} > l_{\tilde{d}}$ **then**
  return false
end
$\bar{h}_{\sigma(j)} \leftarrow \max(\bar{h}_{\tilde{d}} + s_{\tilde{d}} + t_{\tilde{d},\sigma(j)}, e_{\sigma(j)})$
$\delta_{\sigma(j)} \leftarrow \bar{h}_{\sigma(j)} - h_{\sigma(j)}$
if $\delta_{\sigma(j)} > F_{\sigma(j)}$ **then**
  return false
end
$\bar{h}_k \leftarrow h_k + \max(\delta_{\sigma(j)} - ST_{\sigma(j),k}, 0)$
$\bar{h}_{\tilde{\beta}} \leftarrow \max(\bar{h}_{\tilde{T}_e}, \bar{h}_k + s_k + t_{k,\tilde{\beta}})$
$\bar{h}_{\sigma(k)} \leftarrow \max(\bar{h}_{\tilde{\beta}} + t_{\tilde{\beta},\sigma(k)}, e_{\sigma(k)})$
$\delta_{\sigma(k)} \leftarrow \bar{h}_{\sigma(k)} - h_{\sigma(k)}$
if $\delta_{\sigma(k)} > F_{\sigma(k)}$ **then**
  return false
end
$\bar{h}_{\tilde{T}_x} \leftarrow \bar{h}_{\tilde{\beta}}$
$\bar{h}_{\sigma(i)} \leftarrow \max(\bar{h}_{\tilde{T}_x} + t_{\bar{h}_{\tilde{T}_x},\sigma(i)}, e_{\sigma(i)})$
$\delta_{\sigma(i)} \leftarrow \bar{h}_{\sigma(i)} - h_{\sigma(i)}$
if $\delta_{\sigma(i)} > F_{\sigma(i)}$ **then**
  return false
end
return true

Algorithm 3 : Evaluation procedure for the feasibility of an insertion with trip split

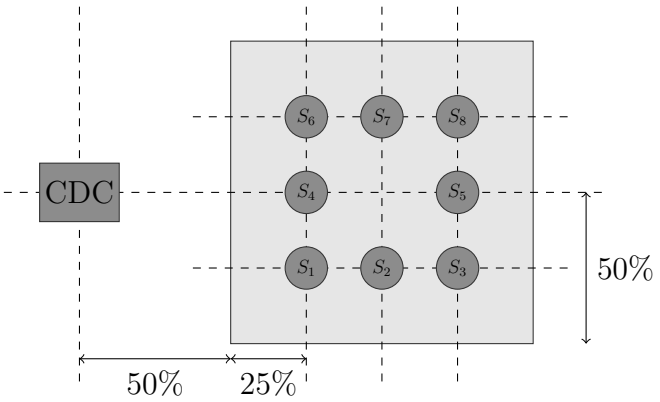


Figure 2.6 – Geometrical layout -50 / 50 / 3 / 3.

According to [23], the maximum benefit of the 2E-VRP compared to the VRP occurs when the CDC is *external* (outside the customer's zone), thus saving travel from the depot to the customers and back, and when the satellites are between the CDC and the customers. The appropriate satellite number is between 7 and 10 for instances with between 100 and 200 customers. In all the benchmarks we use a 50 / 150 / 3 / 3 configuration.

Because the CDC is situated farther from the customers than the depot, some orders may be impossible to deliver. We therefore add an offset δ to each time window, with $\delta = \lceil t_{CDC,Depot} \rceil$. Hence, a time window $[e_i, l_i]$ in a Solomon instance becomes $[e_i + \delta, l_i + \delta]$.

Vehicle configuration

According to Savelsbergh, the instances labeled with a 1 (R1, C1, RC1) have short scheduling horizons, whereas instances labeled with a 2 have long scheduling horizons. Thus, the former are more time-constrained, and the latter are more capacity-constrained. To preserve this idea we use the following ratios for first-level vehicles capacity/second-level vehicles capacity: 4/0.5 ratio for instances 1; 2/0.25 ratio for instances 2.

Objectives

As the number of vehicles is not fixed, our algorithm is used twice. In a first phase we try to minimize the number of vehicles used by minimizing first the number of first level vehicles and then the number of second level vehicles. In a second phase, given the fleet obtained in the first phase, we minimize the routing cost.

2.6.2 Parameters configuration

In this section we discuss the parameters used in our algorithm.

Parameters for fleet optimization phase

We use a sequential optimization scheme to reduce the number of vehicles. We start by reducing the number of first-level vehicles and then we reduce the number of second-level vehicles. When we find a feasible solution with $n + 1$ vehicles, we start looking for a feasible solution with n vehicles by calling the least-used-vehicle removal heuristic. $LB_1 = \lceil \sum_{r \in R} d_r / q_1 \rceil$ is a lower bound on the number of first-level vehicles. If we reach it, we switch to the reduction of the number of second-level vehicles. Overall, we perform a maximum of 25,000 iterations with no more than 12,500 dedicated to the reduction of the first-level fleet. As Ropke and Pisinger [82], we stop the search if 5 or more requests are unplanned and no improvement in the number of unplanned requests has been found in the last 2,000 iterations.

Parameters for cost optimization phase

In the cost optimization phase we use the following parameters (w, c, σ_1, σ_2, σ_3, r, ρ_{min}, ρ_{max})=(0.05, 0.99975, 33, 9, 13, 0.1, 10%, 40%). The notation is that of Ropke and Pisinger [82]. We perform 25,000 iterations, since this gives a compromise between runtime and solution quality.

Heuristics

For both the fleet optimization phase and the cost reduction phase, we use the following heuristics.
Destruction heuristics: random vehicle removal, least-used-vehicle removal, random trip removal, first-level stop removal, trip-related removal, synchronization-based trip removal, random customer removal, worst removal, distance-related removal, time-related removal,

history-based removal.
Repair heuristics: best insertion, 3-regret, 4-regret, 5-regret. Each was used in three variants: without split, existing stops, and customer first (see Section 2.4.3).

2.6.3 Results

The algorithm was coded in C++, and the experiments were conducted using a single core of an Intel Xeon X5675 @ 3.07 GHz under Linux. We report figures for a subset of 12 instances, and the best solutions values and average values found for all instances out of ten runs.

Impact of two-phase initialization

For constructing the initial solution, we compare the best insertion method of Section 3.5.1 and the dedicated two phase best insertion method of Section 2.4.1. As shown in Table 2.2, we observe that although the latter alone produce worst results, it is a better starting point for the fleet reduction.

	Average initial solution				Average solution after fleet opt.				Best solution after fleet opt.			
	Best Insertion		Two Phase		Best Insertion		Two Phase		Best Insertion		Two Phase	
Instance	FL	SL	FL	SL	FL	SL	FL	SL	FL	SL	FL	SL
c101	3.0	13.0	3.16	13.0	3.0	11.32	*3.0*	*11.28*	**3**	**11**	**3**	**11**
c102	3.0	13.0	3.04	12.0	**3.0**	**10.0**	**3.0**	**10.0**	**3**	**10**	**3**	**10**
c201	2.0	4.0	2.0	4.0	2.0	4.0	*2.0*	*3.72*	2	4	**2**	**3**
c202	2.0	4.0	2.0	4.0	2.0	3.24	*2.0*	*3.04*	**2**	**3**	**2**	**3**
r101	2.0	22.0	2.84	22.0	2.0	19.72	*2.0*	*19.64*	**2**	**19**	**2**	**19**
r102	2.0	21.0	2.88	20.0	2.0	18.08	**2.0**	**18.00**	**2**	**18**	**2**	**18**
r201	1.0	5.0	2.0	5.0	**1.0**	**4.0**	**1.0**	**4.0**	**1**	**4**	**1**	**4**
r202	1.0	5.0	2.43	5.0	*1.0*	*3.88*	1.0	3.90	**1**	**3**	**1**	**3**
rc101	3.0	21.0	4.28	20.0	*3.0*	*16.48*	3.0	16.56	**3**	**16**	**3**	**16**
rc102	3.0	18.0	5.24	17.0	3.0	14.4	*3.0*	*14.2*	**3**	**14**	**3**	**14**
rc201	1.0	6.0	3.0	5.0	**1.0**	**4.0**	**1.0**	**4.0**	**1**	**4**	**1**	**4**
rc202	1.0	5.0	2.0	5.0	**1.0**	**4.0**	**1.0**	**4.0**	**1**	**4**	**1**	**4**

Table 2.2 – Comparison of the results of the fleet optimization phase, using best insertion or two-phase best insertion as initialization methods. The FL (resp. SL) columns indicate the number of first-level (resp. second-level) vehicles. Bold figures indicate best solutions; italics indicate best average values.

Impact of FTS on reduction of runtime

On average the repair heuristics are far more time consuming than the destruction heuristics (96.1% versus 1.1% of the total runtime). This partly comes from the fact that the repair methods have to reschedule the solution every time an insertion is performed while the destruction methods reschedule the entire solution only when p customers have been removed from the current solution, and not after each removal.
In table 2.3, we compare the runtime of our algorithm using the extended FTS versus an incremental PERT to check the feasibility of an insertion. This clearly shows that FTS are keys to reducing the computational effort, as our algorithm is 91.9% faster with FTS. With FTS, time-related functions account for 37.7% of the total runtime of our algorithm.

Instance	c101	c102	c201	c202	r101	r102	r201	r202	rc101	rc102	rc201	rc202
PERT (in min)	529.2	527.1	768.1	912.0	330.9	270.4	499.0	937.5	403.0	522.0	687.8	1599.0
FTS (in min)	**42.4**	**56.2**	**55.4**	**77.9**	**26.0**	**32.8**	**42.6**	**72.9**	**26.2**	**42.8**	**50.6**	**63.5**
Difference (in %)	-92.0	-89.3	-92.8	-91.5	-92.1	-87.7	-91.5	-92.2	-93.5	-91.8	-92.6	-96.0

Table 2.3 – Computational time for incremental PERT versus FTS for 25,000 iterations in the cost optimization phase

Impact of reduced neighborhoods on solution quality and runtime

Table 2.4 shows the impact of the reduced neighborhoods on the solution quality and the runtime. We observe that using these neighborhoods has a limited impact on the solution quality, but the runtimes are significantly reduced (by 56.3%). The runtime reduction is larger for type-2 instances, which are capacity constrained, than for type-1 instances, which are time constrained. This is expected, because the tighter the time constraint the smaller the number of reachable satellites. Based on these results we have chosen the following configuration (neighborhood 4 in Table 2.4): we use *customer first* and *existing stops* instead of the full *trip split* operator, and we limit the number of satellites explored to three when creating a new stop for a first-level vehicle.

	Neighborhood 1		Neighborhood 2		Neighborhood 3		Neighborhood 4		Neighborhood 5	
Instance	avg. cost	avg. time	avg. cost	avg. time	avg. cost	avg. time	avg. cost	avg. time	avg. cost	avg. time
C101	2061.0	59.4	**2048.1**	48.3	2060.0	28.7	2053.7	33.5	2065.1	35.3
C102	1981.3	119.2	1984.0	61.7	1983.3	36.3	**1974.4**	42.7	1977.6	45.1
C201	1290.6	79.6	**1290.4**	38.4	1291.6	26.1	1293.8	27.7	1290.6	31.0
C202	1316.1	120.0	1311.3	64.8	**1309.4**	45.1	1311.4	48.9	1319.4	50.0
R101	**2343.5**	28.7	2352.0	27.8	2355.5	17.6	2353.2	20.8	2355.6	21.7
R102	2156.0	46.7	**2148.9**	31.7	2150.4	19.1	2158.1	23.0	2158.4	24.7
R201	1602.8	88.7	**1598.3**	33.1	1605.9	23.7	1604.5	26.8	1605.7	26.5
R202	1550.1	120.0	**1544.5**	49.9	1547.2	37.2	1546.1	40.0	1545.9	40.6
RC101	2624.0	34.2	2637.1	33.5	2616.3	20.7	**2609.0**	23.3	2613.0	23.3
RC102	2447.3	79.7	2451.2	46.9	2451.2	29.3	2446.9	33.9	**2435.1**	36.0
RC201	1821.3	104.8	1820.7	40.2	1812.5	29.2	**1809.3**	30.6	1811.0	32.0
RC202	1531.6	120.0	1534.8	49.5	**1525.0**	31.1	1527.1	37.8	1529.8	39.0
Dev. from N. 1	-	-	-0.05%	-40.4%	-0.10%	-61.6%	-0.16%	-56.3%	-0.07%	-54.5%

Table 2.4 – Comparison of the average cost and runtime over 10 runs for 5 different neighborhood configurations. Configuration 1 uses the entire *trip split* neighborhood. In configuration 2 the *trip split* operator is replaced by its variants *customer first* and *existing stops*. In configurations 3 to 5 the number of satellites explored when creating a new first-level stop is limited to s, with $s = 2$ in neighborhood 3, $s = 3$ in neighborhood 4, and $s = 4$ in neighborhood 5. Bold figures indicate best values.

Impact of custom destruction and repair methods on cost optimization phase

We now focus on the contribution of our new heuristics to the solution of the 2E-MTVRP-SS.
Introducing split-recreate heuristics leads to solution that are 0.5% cheaper on average, showing that this neighborhood, although time consuming, is worth considering.
Biased cost and synchronization based removal do not make a significant difference on the quality of the solutions found. But introducing them makes our method more stable, this can be shown by comparing the standard deviation. When normalizing the average results for each instance to 100, average standard deviations on all test instances are the following: 0.93 with all heuristics, 0.97 without the synchronization based removal and 1.10 without biased costs.

Solutions

In Table 2.5, we report the best and average results found based on 10 runs for each instance. In the fleet optimization phase, we observe that the number of first-level vehicles is always equal to $LB1$ and thus optimal. In the cost optimization phase, there is an average gap of 4.68% between the best solution and the average results. In Table 2.6, we evaluate the impact of the instance characteristics on the routing network used by the vehicles. We notice that there are on average more trips in second-level routes for

type-2 instances (capacity-constrained), and that more satellites are used in instances with clustering.

	Average					Best solution found				
Instance	FL	SL	T1 (min)	Cost	T2 (min)	FL	SL	T1 (min)	Cost	T2 (min)
c101	**3.0**	11.3	21.9	2057.4	30.5	**3**	11	8.3	2022.4	27.5
c102	**3.0**	*10.0*	48.4	1974.9	40.1	**3**	*10*	54.0	1947.6	45.6
c103	**3.0**	*9.0*	9.8	1946.8	46.9	**3**	*9*	13.4	1880.7	51.6
c104	**3.0**	*9.0*	8.4	1857.3	50.5	**3**	*9*	6.9	1811.1	51.5
c105	**3.0**	10.9	26.6	1959.2	35.1	**3**	10	26.4	1934.0	33.5
c106	**3.0**	10.8	32.2	1974.5	39.7	**3**	10	17.0	1945.0	34.0
c107	**3.0**	*10.0*	8.2	1902.8	35.5	**3**	*10*	5.8	1888.9	35.5
c108	**3.0**	*10.0*	8.8	1910.5	38.6	**3**	*10*	8.1	1875.3	34.1
c109	**3.0**	9.2	23.7	1921.3	46.1	**3**	9	5.2	1863.1	47.4
c201	**2.0**	3.5	27.8	1414.2	37.2	**2**	3	19.4	1389.3	33.9
c202	**2.0**	*3.0*	6.3	1317.9	45.4	**2**	*3*	3.6	1305.0	46.6
c203	**2.0**	3.1	9.2	1282.6	52.5	**2**	3	14.3	1272.7	51.8
c204	**2.0**	*3.0*	6.5	1261.3	57.5	**2**	*3*	9.1	1237.9	55.0
c205	**2.0**	3.1	6.8	1322.8	33.0	**2**	3	2.9	1312.1	33.0
c206	**2.0**	*3.0*	4.4	1328.6	36.1	**2**	*3*	4.8	1312.6	31.6
c207	**2.0**	*3.0*	3.9	1306.6	38.2	**2**	*3*	3.4	1280.4	39.5
c208	**2.0**	*3.0*	4.2	1300.0	37.3	**2**	*3*	3.2	1278.3	37.2
r101	**2.0**	19.6	28.9	2352.2	19.3	**2**	19	31.3	2333.5	17.0
r102	**2.0**	18.1	34.4	2152.8	21.7	**2**	18	30.5	2136.8	22.1
r103	**2.0**	13.8	45.2	1955.3	25.5	**2**	13	40.2	1942.7	22.2
r104	**2.0**	10.8	51.7	1804.8	47.2	**2**	10	44.4	1777.2	42.0
r105	**2.0**	14.9	27.7	2138.5	17.2	**2**	14	29.3	2096.8	11.5
r106	**2.0**	12.9	35.9	2028.0	26.8	**2**	12	23.8	1992.4	21.3
r107	**2.0**	11.2	37.7	1817.0	29.6	**2**	11	7.1	1779.2	29.9
r108	**2.0**	10.2	23.8	1702.2	36.6	**2**	10	17.8	1654.3	31.6
r109	**2.0**	12.9	30.3	1982.8	30.1	**2**	12	9.9	1925.9	26.6
r110	**2.0**	12.1	36.9	1888.9	33.6	**2**	12	39.6	1833.6	29.7
r111	**2.0**	*12.0*	44.3	1812.5	31.9	**2**	*12*	38.2	1770.8	29.7
r112	**2.0**	11.1	31.4	1800.9	46.9	**2**	11	54.5	1746.0	39.3
r201	**1.0**	*4.0*	8.2	1614.4	25.5	**1**	*4*	6.8	1587.8	26.6
r202	**1.0**	3.3	32.8	1548.3	39.9	**1**	3	36.6	1530.8	42.8
r203	**1.0**	*3.0*	12.6	1278.3	43.5	**1**	*3*	18.3	1255.1	44.4
r204	**1.0**	2.8	73.4	1200.1	49.3	**1**	2	6.2	1191.7	51.2
r205	**1.0**	*3.0*	6.7	1358.0	29.8	**1**	*3*	8.6	1319.1	31.0
r206	**1.0**	*3.0*	13.0	1251.5	39.1	**1**	*3*	17.5	1228.3	41.4
r207	**1.0**	*3.0*	20.7	1157.7	44.3	**1**	*3*	10.7	1140.2	43.7
r208	**1.0**	*2.0*	10.2	1082.5	49.6	**1**	*2*	5.3	1050.2	50.2
r209	**1.0**	*3.0*	9.2	1275.6	37.1	**1**	*3*	7.8	1258.7	46.8
r210	**1.0**	*3.0*	11.1	1300.4	42.7	**1**	*3*	11.5	1279.8	43.3
r211	**1.0**	*3.0*	15.9	1149.0	49.8	**1**	*3*	7.2	1118.2	44.6
rc101	**3.0**	16.4	26.5	2613.4	20.3	**3**	16	25.7	2577.0	19.7
rc102	**3.0**	14.3	34.8	2460.6	31.4	**3**	14	33.8	2407.1	29.2
rc103	**3.0**	12.0	41.7	2541.9	80.1	**3**	11	48.6	2476.9	72.7
rc104	**3.0**	11.1	41.1	2163.5	41.5	**3**	11	41.4	2125.9	39.7
rc105	**3.0**	15.4	33.1	2602.4	28.3	**3**	15	28.9	2542.6	30.0
rc106	**3.0**	13.9	27.4	2584.7	47.0	**3**	13	36.0	2494.9	47.8
rc107	**3.0**	13.1	41.1	2311.1	39.4	**3**	13	38.0	2271.1	40.4
rc108	**3.0**	12.2	38.9	2229.5	45.8	**3**	12	52.6	2202.9	41.5
rc201	**1.0**	*4.0*	6.8	1834.4	29.5	**1**	*4*	8.2	1787.6	25.4
rc202	**1.0**	*4.0*	65.2	1544.2	37.3	**1**	*4*	68.1	1513.8	36.9
rc203	**1.0**	*3.0*	8.3	1450.7	45.4	**1**	*3*	9.1	1416.2	44.3
rc204	**1.0**	*3.0*	10.0	1211.8	50.1	**1**	*3*	8.6	1188.2	42.8
rc205	**1.0**	*4.0*	27.4	1714.7	34.3	**1**	*4*	5.9	1693.7	30.1
rc206	**1.0**	3.4	34.8	1627.1	41.4	**1**	3	4.3	1583.1	35.3
rc207	**1.0**	*3.0*	10.0	1509.2	50.7	**1**	*3*	11.5	1449.8	49.8
rc208	**1.0**	*3.0*	9.8	1301.4	56.5	**1**	*3*	6.0	1257.3	55.9

Table 2.5 – Average and best solutions. FL (resp. SL) columns indicate the number of first-level (resp. second-level) vehicles. $T1$ (resp. $T2$) is the time spent in the fleet optimization (resp. cost optimization) phase. Bold figures indicate optimal values. Italics indicate that the average value is equal to its equivalent in the best known solution.

Type	avg. no. stops in FL routes	avg. no. trips in SL routes	no. of satellites used
C1	2.6	2.5	6.1
C2	2.7	4.8	5.2
R1	2.2	1.5	3.5
R2	2.7	2.7	2.3
RC1	2.6	1.8	5.6
RC2	3.8	2.7	3.3

Table 2.6 – Comparison of the number of stops in first-level routes, the number of trips in second-level routes, and the number of satellites used for each type of instance.

2.6.4 Impact of time windows and synchronization on the cost of solutions

In what follows, we assess the impact of time windows and exact synchronization on the cost of solutions.
For testing the impact of time windows on the cost of solutions, we run our algorithm on the same instances while removing the time windows of customers. As Solomon's instances from the same type only differ in their time windows, when removing them, there are only six instances.
To evaluate the impact of exact synchronization on the cost of solutions, we run our algorithm on all instances of the 2E-MTVRP-SS while only imposing a precedence constraint for transfers at satellites: a second-level vehicle can pick up goods iff a first-level vehicle has already dropped them. This corresponds to the case in which satellites have a storage capacity.
For creating Table 2.7, we run our algorithm ten times for each instance. We select the best result for each instance and then compute the average of the second-level fleet size and cost for each type of instance. We do not report the results for first-level fleet size since for the 2E-MTVRP-SS we already reached the lower bound.

Type	2E-MTVRP-SS		No TW		Precedence	
	SL	Cost	SL	Cost	SL	Cost
C1	9.8	1911.0	9	1701.7	9.7	1911.8
C2	3.1	1284.2	3	1185.5	3	1268.4
R1	12.8	1913.3	9	1595.6	13	1911.5
R2	2.9	1268.3	2	975.1	2.9	1265.4
RC1	13.3	2348.6	10	2018.5	13.1	2343.2
RC2	3.4	1476.3	2	1061.2	3.25	1525.6

Table 2.7 – Comparison of the average best results for each instance type for the original 2E-MTVRP-SS, removing time windows and enforcing only precedence at transfers

From table 2.7, we observe that considering a precedence constraint instead of the exact synchronization constraint only marginally reduces the second-fleet size and the average solution cost. This is not very surprising since we observe in the solution for the 2E-MTVRP-SS that first-level vehicles tend to stay for rather long periods of time at the same satellite, acting as depot. A realistic perspective here would be to enforce a maximum stay at satellites and to add waiting stations. A second observation is that removing time windows drastically reduces the second-level fleet size and also the cost of the solution.

2.7 Conclusion

We have presented an extension of the well-known 2E-VRP that takes into account constraints arising in city logistics (time window constraints, synchronization constraints, and multiple trips for some vehicles). We have developed an ALNS to solve the 2E-MTVRP-SS, which has both custom destruction and repair heuristics and an efficient way to check if an insertion is feasible. These contributions help to find good solutions in a reasonable time, thus making it possible to consider this algorithm for other vehicle routing problems with synchronization constraints. As for city logistics, our comparison in 2.6.4, cleary outlines that time windows have a greater influence than exact synchronization on the cost of solutions.

3

Une matheuristique basée sur un algorithme de recherche à voisinage large pour le problème de tournées de véhicules avec cross-docking

Dans ce chapitre, nous nous intéressons au problème de tournées de véhicules avec cross-docking (Vehicle Routing Problem with Cross-Docking - VRPCD). Il s'agit d'un problème de collectes et de livraisons intégrant un unique cross-dock. Les véhicules partent du cross-dock, collectent les requêtes, reviennent au cross-dock où ils peuvent décharger/recharger des requêtes, enfin ils livrent les requêtes. Des fenêtres de temps existent aux points de collecte et aux points de livraison. Nous proposons une matheuristique basée sur une recherche à voisinage large pour le VRPCD. La méthodologie proposée combine une méthode de type LNS à la résolution d'un problème SPP. Notre méthode a été évaluée sur les instances de la littérature où elle améliore les meilleurs résultats connus pour 30 instances sur 35, en proposant une performance moyenne meilleure de 0,78% sur les petites instances et de 2,16% sur les grandes instances.

3.1 Article II: a large neighborhood search based matheuristic for the vehicle routing problem with cross-docking

The vehicle routing with cross-docking (VRPCD) consists in defining a set of routes that satisfy transportation requests between a set of pickup points and a set of delivery points. Vehicles first visit pickup locations, then a cross-docking platform, where items can be transferred during a consolidation process, and eventually delivery locations. In this paper we propose a new solution methodology to address this problem. It is based on large neighborhood search and periodically solving a set partitioning and matching problem with third party solvers. Our methods outperforms existing methods by improving the

best known solution in 30 out of 35 instances from the literature.

3.2 Introduction

Cross-docking is a distribution strategy, in which goods are brought from suppliers to an intermediate transshipment point, the so-called cross-dock, where they can be transferred to another vehicle before being delivered to the customers. Transfers are done based on consolidation opportunities. There is little to no storage capacity at the cross-dock, thus cutting down inventory holding costs compared to traditional distribution centers. It can also help reducing distribution costs by making it easier to consolidate shipments to full truck loads compared to point-to-point deliveries. Cross-docking has been successfully applied to several sectors, the canonical example being Walmart for which it is said to have been the key to the growth of the retailer in the 1980s [94].

Many cross-docking related problems exist such as : location, assignment of trucks to doors, inner flow optimization or routing. In particular, the *vehicle routing problem with cross-docking* (VRPCD) consists in designing routes to pick up and deliver a set of transportation requests at minimal cost using a single cross-dock. Trucks start by collecting items, then return to the cross-dock where they can unload some requests and load others before starting their delivery trips. The exchange of goods at the cross-dock is a consolidation process which aims to minimize the total delivery cost by collaboration between vehicles. The VRPCD can be seen as a *pickup and delivery problem with transfers* with only one compulsory transfer point.
In this paper, we propose a matheuristic, that relies on a large neighborhood search to create a pool of routes. These routes are then used in a set partitioning and matching problem. This problem is solved using a branch-and-check [98], a hybrid method that relies both on a mixed integer programming (MIP) solver and a constraint programming (CP) solver. Numerical results are presented and show that our method outperforms existing algorithms in the literature on most instances.

The remainder of this paper is organized as follows. A literature review is presented in Section 3.3, while the problem is defined in Section 3.4. Section 3.5 is devoted to the solution methodology. Lastly, computational results are presented in Section 3.6.

3.3 Literature review

The VRPCD is a routing problem and as such our literature review will be focused on routing. We refer the reader to [11, 1, 100] for a general overview of cross-docks and cross-docks related problems. Academic literature on cross-docking is fairly recent and only a few papers focus on the routing aspect. Lee et al. [57] solved a VRPCD variant in which all vehicles have to arrive simultaneously at the cross-dock. They proposed an exact formulation and developed a tabu-search heuristic to solve instance with 10, 30 and 50 nodes. This tabu-search heuristic was later improved by Liao et al. in [59]. Wen et al. [104] extended the problem by adding time windows on nodes and relaxing the constraint of simultaneous arrival at the cross-dock for all vehicles, only imposing precedence constraints based on the consolidation decisions. They presented a MIP formulation of that extension, and proposed a tabu-search embedded in an adaptive memory procedure. They solved instance with up to 200 requests originated from real-life data, and compared their results with a lower bound corresponding to two VRPTW problems. These results were improved

by Tarantilis [97] using a multi-start tabu-search and by Morais et al. [69] with a method based on an iterative local search and a set partitioning model. For this problem, Morais et al. introduced new instances with up to 500 customers. Petersen and Ropke [75] worked with a Danish company distributing flowers on a variant of the VRPCD with time windows, optional cross-dock return and multiple trips per day, that they refer to as the *vehicle routing problem with cross-docking opportunity*. They created a parallel ALNS to solve instances ranging between 585 to 982 requests. Santos et al. [84, 87] proposed two different branch-and-price approaches on a VRPCD variant for which there is a cost for transferring item at the cross-dock and there are no temporal constraints, they later extended their approach in [85] to a problem where return to the cross-dock for consolidation is optional. They refer to this problem as the *pickup and delivery problem with cross-docking* and show that it can reduce the routing cost between 3.1% and 7.7% on their instances, which are reductions of those of Wen et al. [104]. Dondo and Cerdà [29] considered a variant of the VRPCD in which they modeled each door at the cross-dock individually (handling speeds, travel times to other doors, ...) and with a smaller number of doors than the number of trucks. They proposed a solving methodoly based on a MILP formulation and a sweep heuristic and they solved instances with up to 70 requests. Enderer [36] studied the *dock-door assignment and vehicle routing problem* in which only the assignment of trucks to doors and the routing in the delivery part have to be done. He proposed and compared several exact and heuristic methods. In a recent survey on synchronization in cross-docking networks [14], Buijs et al. classified the VRPCD as a network scheduling problem with synchronization.
Closely related problems are problems with transfers such as the *pickup and delivery problem with transfers* (PDPT) for which both heuristics methods: ALNS [63], GRASP+ALNS [79], and exact method: branch-and-cut [18] have been proposed. The *two-echelon vehicle routing problem* [74], for which collaboration between the two echelons is at the heart of the delivery process, is also close to the VRPCD, in particular the *two-echelon multiple-trip vehicle routing problem with satellite synchronization* [40] which deals with temporal aspects. Lastly, cross-docking operations correspond to both *operations synchronization* and *resource synchronization* as described in [31].

A common technique for matheuristics is to have one or several (meta)heuristics generating a pool of routes and to solve a set partitioning problem (either during the search or as a postprocess). It has been applied to a large variety of routing problems, with many different heuristics to fill the pool. For example, it has been used: with a local search to solve the VRP [81], with a tabu search to solve the split delivery VRP [4], with an adaptive large neighborhood search for a technician routing problem [76], with a GRASP and local search procedures for the truck and trailer routing routing problem [102] and the VRP with stochastic demands [67], and with an iterated local search to solve seven VRP variants in [95]. This last method was applied to the VRPCD in [69]. For more details on matheuristics we refer the reader to the surveys on matheuristics in vehicle routing problems by Doerner and Schmid [28] and by Archetti and Speranza [3]. In these surveys, this technique falls under the category *set-covering/partitioning based approaches* in the former and *restricted master heuristics* in the latter. The LNS+SPM proposed in this paper can be classified in the aforementioned categories, because of the set partitioning and matching (SPM) component. However, the SPM is more than a set partitioning (hence the *matching*), in particular it encompasses matching and scheduling decisions. The SPM is solved using a *branch-and-check* approach [98], which is an hybrid technique integrating MIP and constraint programming (CP). We give more details on

branch-and-check in Section 3.5.2.

3.4 Problem formulation

In this section we present the VRPCD, with a special focus on the scheduling constraints at the cross-dock.

3.4.1 Problem statement

In the *vehicle routing with cross-docking* (VRPCD), we consider a cross-dock c, a set of requests R, and a homogeneous fleet of vehicles V, each of capacity Q and based at o. Each request $r \in R$ has to be picked up at its pickup location p_r within its pickup time window $[e_{p_r}, l_{p_r}]$, and has to be delivered at its delivery location d_r within its delivery time window $[e_{d_r}, l_{d_r}]$. In case of early arrival, a vehicle is allowed to wait, but late arrivals are forbidden. We denote by P the set of pickup locations and by D the set of delivery locations.
Each vehicle starts at o, then goes to several pickup locations, arrives at the cross-dock where it unloads/reloads some requests. A vehicle then visits delivery locations and eventually returns at o. Note that a vehicle has to visit the cross-dock even if it does not unload nor reload any requests there. The sequence of operations at the cross-dock is described in Section 3.4.2. We call *pickup leg* the sequence of operations performed by a vehicle between its departure from the cross-dock for pickups and its return to the cross-dock for consolidation operations (not including consolidation operations). Symmetrically, we call *delivery leg* the sequence of operations performed by a vehicle between its departure from the cross-dock after consolidation operations (not including them), and its return to the cross-dock at the end of the day.
The VRPCD is defined on a directed graph $G = (V, A)$, with $G = \{o\} \cup P \cup \{c\} \cup D$ and $A = \{(o,p)|p \in P\} \cup P \times P \cup \{(p,c)|p \in P\} \cup \{(c,d)|d \in D\} \cup D \times D \cup \{(d,o)|d \in D\} \cup \{(o,c),(c,o)\}$. With each arc $(i,j) \in A$ is associated a travel time $t_{i,j}$ and a travel cost $c_{i,j}$.
Solving the VRPCD involves finding $|K|$ routes, and a schedule for each route, such that capacity and time-related constraints are satisfied, at minimal routing cost. An arc-based mathematical formulation can be found in [104].

3.4.2 Operations at cross-dock

Following [104], if a vehicle k has to unload a set of requests R_k^- and reload a set requests R_k^+ at the cross-dock, the time spent at the cross-dock can be divided in up to four periods, as shown on Fig. 3.1:

- Preparation for unloading. The duration δ_u of this period is fixed.
- Unloading of requests. The duration of this period depends on the quantity of products to unload, so for a vehicle k: $(\sum_{i \in R_k^-} q_i)/s_u$, where s_u corresponds to the unloading speed in quantity per time unit. All unloaded requests become available for reloading at the end of this period.
- Preparation for reloading. The duration δ_r of this period is fixed.

- Reloading of requests. Similarly to unloading, the duration of this period depends on the quantity of products to unload. For a vehicle k: $(\sum_{i \in R_k^+} q_i)/s_r$, where s_r corresponds to the reloading speed in quantity per time unit. All requests to reload must have been unloaded before the beginning of the reloading operation (preemption is not allowed).

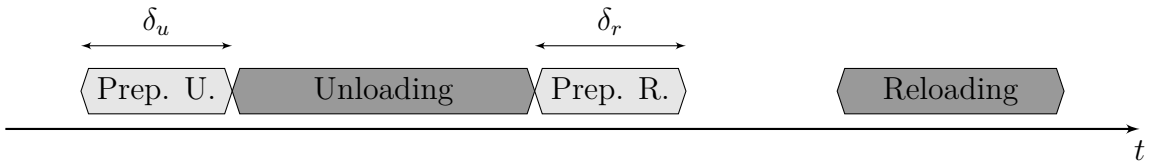


Figure 3.1 – Example of a time chart for a vehicle unloading and reloading at the cross-dock. The vehicle has to wait to be reloaded because some requests are not yet available when it is ready for reloading

Requests that are not transferred at the cross-dock remain in the vehicle, consequently if a vehicle does not unload (resp. reload) any requests it does not have to undergo the unloading (resp. reloading) process, thus also saving the preparation time. So, in the case where a vehicle would not unload nor reload any item it would just have to stop at the cross-dock and could leave immediately. We do not consider any limit on the number of available docks at the cross-dock.

3.5 Solution approach

In this section we describe the solution approach methodology used for the VRPCD. The main component is Large Neighborhood Search (LNS), which is *boosted* by periodically solving a Set Partitioning and Matching problem (SPM). More precisely, every time the LNS finds a new solution, the legs in this solution are added to a pool of legs that acts as a long-term memory. The SPM is based on a set partitioning problem where the set to partition is $P \cup D$ and the candidate partitions are the legs in the pool. Thus the SPM assists the LNS by selecting good legs that have been previously discovered. We call this method Large Neighborhood Search with Set Partitioning and Matching (LNS+SPM). A sketch of LNS+SPM is given in Algorithm 4. Lines 2-15 are those of the LNS, described in detail in Section 3.5.1. The SPM is used during the search (l.18) and the set of legs that are used in each SPM problem is managed (l.20). SPM is discussed in Section 3.5.2.

3.5.1 Large Neighborhood Search

Large Neighborhood Search was introduced by Shaw [92]. It iteratively destroys (removes requests from the solution) and repair (reinserts requests) the current solution using heuristics. It has been widely used ever since its introduction, in particular its extension, the so-called Adaptive Large Neighborhood Search (ALNS) [82, 77], which selects the destroy and repair methods based on their past successes. In this paper we select destruction and repair methods randomly (our method is thus LNS-based), as preliminary experiments suggested that the adaptive layer has an extremely limited impact on the quality of the solutions for this problem. Note that Parragh and Schmid [72] took the same decision for their method for their dial a ride problem, which also makes use of LNS and set

Result : The best found solution $s^\star$
Pool of legs $\mathcal{L} := \emptyset$
Generate an initial solution s
$s^\star := s$
while *stop criterion not met* **do**
  $s' := s$
  Destroy quantity: select a number Φ of requests to remove from s'
  Operator selection: select a destruction method M^- and a repair method M^+
  Destruction : Apply M^- to remove Φ requests from s', and put them in the requests bank of s'
  Repair: Apply M^+ to reinsert the requests in the requests bank in s'
  if *acceptance criteria is met* **then**
    $s := s'$
  end
  if *cost of* s' *is better than cost of* $s^\star$ **then**
    $s^\star := s'$
  end
  Add legs of s' to $\mathcal{L}$
  if *set partitioning and matching condition is met* **then**
    Perform set partitioning and matching with the legs in $\mathcal{L}$
    Update $s^\star$ and s if a new best solution has been found
    Perform pool management
  end
end
return $s^\star$

Algorithm 4 : LNS+SPM

partitioning. We also do not noise the cost function.
In what follows, we present the proposed destruction and repair methods, and strategies to reduce the runtime.

Destruction methods

When partially destroying a solution, we select a destroy method M^- and a number Φ of requests to remove. Unless stated otherwise, this method is reused until Φ is reached. Random removal, worst removal, related removals and historical node-pair are inspired from [77], while we introduce the transfer removal for this particular problem.

Random removal: we randomly remove a request

Worst removal: we remove a request with a high removal gain. It is defined as the difference between the cost of the solution with and without the request. Then, we sort the requests in non increasing order of their removal gains and put them in a list N. We select the request to remove in a randomized fashion as in [82]: given a parameter p, we draw a random number y between 0 and 1. We then remove the request in position $y^p \times |N|$.

Historical node-pair removal: each arc $(u, v) \in G$ is associated with the cost of the cheapest solution it appears in (initially this cost is set to infinity). We then remove the

request which is served using the arcs with the highest associated costs. A randomized selection, similar to *worst removal*, is performed.

Related removals: these methods aim to remove related requests. Let the relatedness of requests i and j be $R(i, j)$. We use two distinct relatedness measures: distance and time. The distance measure between two requests is the sum of the distance between their pickup points and the distance between their delivery points. The time measure is the sum of the absolute difference between their start of service at their pickup points and the absolute gap between their start of service at their delivery point. In both cases a small $R(i, j)$ indicates a high relatedness. A randomized selection, similar to *worst removal* (albeit with a non decreasing ordering), is performed.

Transfer removal: for each pair of routes (v_i, v_j), with $v_i \neq v_j$ we compute the number of requests transferred from v_i to v_j. Then we iteratively apply a roulette wheel selection on the pairs of routes (the score of a pair being the number of requests transferred), and we remove the requests that are transferred between the routes in the selected pair. If there are less transferred requests than the target number Φ to remove, we remove all the transferred requests and switch back to random removal for the rest.

Repair methods

In LNS, the unplanned requests are stored in a so-called *requests bank*. In the following we explain how we insert these requests into a partial solution.

Best insertion from all the requests r in the requests bank, we insert the one with the cheapest insertion cost considering all possible insertion of p_r in pickup legs and d_r in delivery legs.

Regret Insertion for each request r in the requests bank and for each pair of vehicles (pickup vehicle, delivery vehicle), we compute the cost of cheapest feasible insertion (if any). Note that here the pickup vehicle and the delivery vehicle may actually be the same vehicle (to model the case of insertion without transfer). Then, with these insertion options, we compute the *k-regret value* of r, as $c_r^k = \sum_{i=1}^{k} f_i - f_1$, where f_1 is the cost of the cheapest insertion, f_2 is the cost of the second-cheapest insertion and so on, and k is a parameter. We insert the request with the highest regret value.
We use 2-regret to generate the inital solution in Algorithm 4.

We use best-insertion, 2-regret, 3-regret and 4-regrets as repair methods in LNS.

Reducing the runtime of repair methods

In LNS, repair methods take most of the runtime: more than 96% of the runtime in [40]. We have adopted two strategies to reduce the runtime of the repair methods: reducing the size of the neighborhoods and efficient time checking.

Size of the neighborhood Because each request can be transferred, the number of candidate insertions for a request is $\Theta(|V|^2)$, which is much larger than in the case of the traditional VRP, for which it is only $\Theta(|V|)$. For large instances, the quadratic size

of the reinsertion neighborhood becomes a problem, as it takes a lot of time to explore the entire neighborhood. This does not necessarily lead to better solution, because not all transfer opportunities are worth considering. Thus, for each request we evaluate all the insertions without transfers, and we only consider transfer opportunities among a subset of g vehicles. In details, for each request r in the requests bank we sort the vehicles for pickup (resp. delivery) in non decreasing order of the cheapest insertion of r in their pickup (resp. delivery) leg. Then we only consider insertion with transfers between the first g vehicles according to the previous order. A discussion on the appropriate value of g is presented in Section 3.6.2. We use this restricted neighborhood exploration in both repair methods defined in Section 3.5.1.

Efficient time checking When evaluating the insertion of a customer into a solution in a repair method, if this insertion is *promising* (i.e. worth considering with respect to its cost) according to the repair method, we need to make sure that it is feasible. Checking if an insertion is feasible with respect to capacity constraints can easily be done in constant time, but it is more complex regarding time constraints for which a straightforward implementation has a linear complexity. Savelsbergh [88] introduced the so-called *forward time slacks* that allow to check in constant time if an insertion is feasible in the VRP with time windows. Masson et al. [64] extended this constant feasibility check to the pickup and delivery problem with transfers (PDPT). This method has already been used several times [63, 40, 42] and has proven successful in significantly reducing the runtime, for example in [40]. We refer the reader to [64] for a detailed description of its implementation. As the VRPCD can be seen as a PDPT, we reuse it here.

3.5.2 Set partitioning and matching procedure

In the LNS, a solution is rejected solely based on its cost with respect to the best found solution so far. If a solution is rejected because it contains *bad* legs, that account for its high cost, it may also contain good legs, which will be lost once the solution is rejected. It may also happen that the matching of legs to form routes could be improved (giving more time flexibility and thus making further improvements easier for example). Storing the legs found by the LNS and solving a set partitioning based problem with them, can address these issues.

We call *set partitioning and matching* (SPM) the following problem: given a set of legs L, select a subset of legs $\tilde{L}$ such that (1) each request is picked up (resp. delivered) by exactly one leg in $\tilde{L}$ and (2) legs in $\tilde{L}$ can be matched to form routes that respect time constraints. Each leg $l \in L$ has an associated routing cost c_l, the objective in the SPM is to minimize the sum of the costs of the selected legs. In what follows, we present the branch-and-check method that we use to solve the SPM, its master problem (set partitioning), its subproblem (matching and scheduling) and eventually we detail the management of the pool of legs in LNS+SPM.

Notice that to some extent, our approach is here similar to that of Morais et al. [69], who used SPP as intensification phase of their SP-ILS. However there are two main differences: (1) their pool of legs is much smaller (the legs that appears in their ten best solutions) and (2) when they can not find a feasible matching (they did not detail their matching procedure), they perform a repair phase by moving requests from one route to another.

Branch-and-check

Branch-and-check was introduced by Thorsteinsson [98], we present it through the following optimization problem:

$$M1: \min c^\intercal x \tag{3.1}$$
$$Ax \leq b \tag{3.2}$$
$$H(x,y) \tag{3.3}$$
$$x \in \{0,1\}^n \tag{3.4}$$
$$y \in \mathbb{R}^m \tag{3.5}$$

Assume that $H(x,y)$ represents a set of constraints that have a limited impact on the LP relaxation and/or are difficult to efficiently model in a MIP, but that could be handled relatively easily by a CP solver. (3.1), (3.2) and (3.4) is a relaxation (M2) of (M1), that can be solved in branch-and-bound fashion. The general principle of branch-and-check is the following. To solve (M1), a branch-and-bound is performed on (M2). Whenever an integral solution of (M2) is found in the branch-and-bound process, a CP solver is called to check constraints (3.3). If they are satisfied, the best solution found so far for (M1) is updated accordingly. Otherwise, this solution is rejected. In both cases the branch and bound process continues.

In the set partitioning and matching procedure, we solve a classical set partitioning problem (SPP) where the set to partition is $P \cup D$ and the candidate partitions are the legs in the pool $\mathcal{L}$. A solution to the SPP is a solution to the VRPCD iff we can match legs to form a set of routes that respects time constraints. Since time aspects of consolidation operations are neglected in the SPP, it may be impossible. To determine that, we solve a dedicated matching and scheduling subproblem. Here, the SPP is the relaxation (M2) of the set partitioning and matching problem (M1) while the matching and scheduling scheduling subproblem play the role of $H(x,y)$.

Set partitioning

Given a set of legs $L = L_p \cup L_d$, where L_p is a set of pickup legs and L_d is a set of delivery legs. For each request $r \in R$, we can define a binary constant $\lambda_{r,l}$, that indicates whether this request is served by this leg. For each leg $l \in L$ we consider a binary variable x_l to determine if it is selected. The set partitioning problem (SPP) is then

$$\min \sum_{l \in L} c_l \times x_l \tag{3.6}$$
$$\sum_{l \in L_p} \lambda_{r,l} \times x_l = 1 \qquad \forall r \in R \tag{3.7}$$
$$\sum_{l \in L_d} \lambda_{r,l} \times x_l = 1 \qquad \forall r \in R \tag{3.8}$$
$$x_l \in \{0,1\} \qquad \forall l \in L \tag{3.9}$$

The objective (3.6) is to minimize the cost of the selected legs while constraints (3.7) (resp. (3.8)) ensure that each pickup point (resp. delivery point) is covered by exactly one leg.

Calling an auxiliary subproblem when an integral solution of a MIP is found is implemented by means of an *incumbent callback* in most MIP solvers. To save some time, we provide

the MIP solver with an initial solution (warm start): the best solution found so far ($s^\star$ in Algorithm 4).

If the matching and scheduling subproblem rejects a solution to the SPP (there exists no solution to the associated matching and scheduling subproblem), one could try to add some lazy constraints besides the rejection of that solution (it is common in branch-and-check). For example by detecting time incompatibilities: if, for given a pickup leg l and a given delivery leg l' that have some requests in common, there is not enough time to perform the associated operations at the cross-dock whether they are packed together or not, we could add $x_l + x'_l \leq 1$ as a lazy constraint in the SPP. We have tried such technique in early experiments, but we do not include it in our final algorithm as it performed poorly. We identify two reasons for this (1) the vast majority of incumbent callbacks are successful, (2) as of CPLEX 12.6.1 (which is the MIP solver that we use) adding lazy constraints callbacks disable dynamic search which seems to be a very useful feature to reduce CPLEX's runtime on the SPP.

Matching and scheduling subproblem

A solution of the previous SPP, which involves a set of pickup legs denoted $\tilde{L_p}$ and a set of delivery legs denoted $\tilde{L_d}$, is a solution to the VRPCD iff there exists a matching of pickup legs and delivery legs to form routes that respects time constraints.
For each pickup leg $l \in L_p$, we can compute its earliest feasible arrival time at the cross-dock a_l, and, for each delivery leg $l' \in L_d$, we can compute its latest feasible departure time from the cross-dock $b_{l'}$. Besides, for each pickup leg $l \in \tilde{L_p}$ we can determine the set of selected delivery legs T_l that deliver at least one request picked up by l. If we match a pickup leg l and a delivery leg l' together to create a route, we can define an associated unloading task $o^-_{ll'}$, with a set of requests $R^-_{ll'}$ being unloaded, and a reloading task $o^+_{ll'}$, with a set of requests $R^+_{ll'}$ being reloaded. These tasks have to be performed iff l and l' are in the same route.
The problem is modeled as a constraint satisfaction problem, represented using notation from OPL (Optimization Programming Language [101]). In particular the model is based on the notion of interval variables and uses alternative constraints. As used here (from [51]):

> 'An interval variable represents an interval of time during which a task happen, and whose position in time is an unknown of the scheduling problem. An interval is characterized by a start value, an end value and a size. (...) An interval variable can be optional, that is, one can decide not to consider [it] in the solution schedule.'

In this model, we model alternative activities [7] by using alternative constraints (from [51]):

> 'An alternative constraint between an interval variable a and a set of interval variables $b_1, \ldots, b_n$ models an exclusive alternative between $b_1, \ldots, b_n$. If interval a is present, then exactly one of intervals $b_1, \ldots, b_n$ is present and a starts and ends together with this specific interval. Interval a is absent if and only if all intervals in $b_1, \ldots, b_n$ are absent.'

We thus consider the following problem:

$$\text{Alternative}(t_l, \{o^-_{ll'}; \forall l' \in \tilde{L}_d\}) \qquad \forall l \in \tilde{L}_p \tag{3.10}$$

$$\text{Alternative}(t_{l'}, \{o^+_{ll'}; \forall l \in \tilde{L}_p\}) \qquad \forall l' \in \tilde{L}_d \tag{3.11}$$

$$\text{o}^-_{ll'}.IsFacultative \leftarrow True \qquad \forall l \in \tilde{L}_p, \forall l' \in \tilde{L}_d \tag{3.12}$$

$$\text{o}^+_{ll'}.IsFacultative \leftarrow True \qquad \forall l \in \tilde{L}_p, \forall l' \in \tilde{L}_d \tag{3.13}$$

$$\text{o}^-_{ll'}.IsPresent \iff o^+_{ll'}.IsPresent \qquad \forall l \in \tilde{L}_p, \forall l' \in \tilde{L}_d \tag{3.14}$$

$$\text{t}_{l'}.Start \geq t_l.End \qquad \forall l \in \tilde{L}_p, l' \in T_l \tag{3.15}$$

$$\text{o}^+_{ll'}.Start \geq o^-_{ll'}.End + \delta_r \qquad \forall l \in \tilde{L}_p, \forall l' \in \tilde{L}_d \text{ s.t. } R^+_{ll'} \neq \emptyset \tag{3.16}$$

$$\text{o}^+_{ll'}.Start \geq o^-_{ll'}.End \qquad \forall l \in \tilde{L}_p, \forall l' \in \tilde{L}_d \text{ s.t. } R^+_{ll'} = \emptyset \tag{3.17}$$

$$\text{o}^-_{ll'}.Start \geq a_l + \delta_u \qquad \forall l \in \tilde{L}_p, \forall l' \in \tilde{L}_d \text{ s.t. } R^-_{ll'} \neq \emptyset \tag{3.18}$$

$$\text{o}^-_{ll'}.Start \geq a_l \qquad \forall l \in \tilde{L}_p, \forall l' \in \tilde{L}_d \text{ s.t. } R^-_{ll'} = \emptyset \tag{3.19}$$

$$\text{o}^+_{ll'}.End \leq b'_l \qquad \forall l \in \tilde{L}_p, \forall l' \in \tilde{L}_d \tag{3.20}$$

For each pickup leg l we create an interval variable t_l that represents the associated unloading task that takes place at the cross dock. Alternative constraints (3.10) and (3.12) ensure that for each pickup leg l exactly one unloading task $o_{ll'}$ is scheduled and that it is equal to t_l. The same holds for delivery legs and reloading operations through variables $t_{l'}$ and constraints (3.11) and (3.13). Constraints (3.14) ensure that the unloading operation associated with the matching of pickup leg l and the delivery leg l' in the same vehicle is present iff the corresponding reloading operation is present as well. Constraints (3.15) ensure that all the reloading operations that depend on a pickup leg l start no earlier than the end of the unloading task associated with l. Constraints (3.16) and (3.17) ensure that when two legs are packed together, the delay between the two tasks respects the model presented in Section 3.4.2. Constraints (3.18) and (3.19) ensure that for each pickup leg, its corresponding unloading operation cannot start before the earliest feasible arrival time at the cross-dock. Constraints (3.20) ensure that for each delivery leg, its corresponding reloading operation is done by its latest feasible departure time.

Set partitioning and matching criterion and pool of legs management

When solving the SPP, one only needs to consider the legs in the pool of legs $\mathcal{L}$ (from Algorithm 4), that are non-dominated. A pickup (resp. delivery) leg l_i is said to be dominated by a leg l_j iff: l_i and l_j serve the same set of requests, $c_j < c_i$ and $a_j \leq a_i$ (resp. $b_j \geq b_i$). It is clear that for every solution of the SPP that contains a dominated leg, there exists a solution with a smaller cost that do not contain it.
Regarding the management of the pool of legs, it is important to observe two things. First, the more we progress in our algorithm, the more we can provide the MIP solver with a good starting solution, thus improving its cutoff capacities. As such SPPs tends to be easier to solve at the end of our algorithm than at the beginning. Second, providing too many legs to the MIP solver can be a problem as it may fail to improve the initial solution within its given time limit. So we implement the following policy: every time the SPM is solved, if the MIP solver is able to find an optimal solution and prove its optimality within its time limit, we keep legs in the pool, otherwise the pool is cleared. In practice, this policy tends to empty the pool more frequently at the beginning of the search than at its end.

The set partitioning and matching procedure is called every k_{SPM} iterations. A discussion on the value of k_{SPM} is presented in Section 3.6.2.

3.6 Computational experiments

The algorithm is coded in C++ and uses CPLEX and CP Optimizer from IBM ILOG Cplex Optimization Studio 12.6.1 as MIP solver and CP solver respectively. The experiments were conducted under Linux using an Intel Xeon X5675 @ 3.07 GHz. Only one core is used both by our code and third party solvers.

3.6.1 Instances

Two sets of instances exist for the VRPCD. The first one was introduced by Wen et al [104], and contains instances ranging from 50 to 200 requests. It is based on real life data from a Danish logistics company. The second set of instances was introduced by Morais et al. [69], and contains instances ranging from 200 to 500 requests. It is derived from the Gehring and Homberger's instances for the VRPTW. In both sets there is no limit on the number of vehicles.

3.6.2 Parameters

The stopping criterion is set to 20 000 iterations which is a good compromise between quality and runtime. Following Masson et al. [63] and preliminary experiments, the number Φ of requests to remove in the repair phase of the LNS is drawn randomly in the interval [min(30, 10% of $|R|$), max(60, 20% of $|R|$)].
In the following subsection, we describe the tuning experiments we have conducted to set the acceptance criterion, the reduction of the transfer neighborhood and the SPM period. As training set, we select the following instances: 50b, 100b, 150b, 200b from Wen et al. and R1-4-1, R1-6-1, R1-8-1 and R1-10-1 from Morais et al.

Acceptance criterion

Three sorts of acceptance criteria have been tested: descent (a solution is accepted as current solution iff it is better than the current best solution), α threshold (every time a solution is less than α% more expensive than the best solution found so far, we accept it as the current solution) and simulated annealing (as implemented by Ropke and Pisinger [82]). Table 3.1 compares the performance of the following acceptance criteria: descent, 1% threshold, 3% threshold, 10% threshold and simulated annealing, with descent taken as reference. From this table we can conclude that except for 10% threshold, all the other acceptance criteria have similar performances at the end of the algorithm. We select descent as acceptance criteria as it requires no parameter.

Accept. Crit.	Descent	1% thresh.	3% thresh.	10% thresh.	Sim. Ann.
Gap (%)	0.00	-0.01	0.04	1.55	0.00

Table 3.1 – Comparison of the average performance for five different acceptance criteria; 10 runs were performed for each instance in the training set; descent is taken as reference.

Reduction of the transfer neighborhood

As mentioned in Section 3.5.1, for each request in the requests bank we only evaluate insertions with transfers between the g most promising vehicles. To determine g, we started by running experiments with a large value of g and we observed that in no best solution found during the search was a vehicle transferring items to more than 10 vehicles. In Table 3.2, we report the average results out of five runs for the training instances from Morais et al. for different values of g. We can see that already with $g = 5$, the quality of the solutions found is equivalent to the situation where all transfers are considered (case: *infinity*). This confirms our intuition that only a subset of transfers are worth considering. Figure 3.2 illustrates the reduction in runtime when g decreases, with a reduction of more than 50% for R1-10-1 (500 requests). According to this observation, g has been set to 5, in the following experiments.

g	5	10	20	∞
Gap (%)	-0.01	0.00	0.03	0.00

Table 3.2 – Comparison of the impact of g on the solution quality for 5 runs for each instance from Morais et al. in the training set. *Infinity* is taken as reference

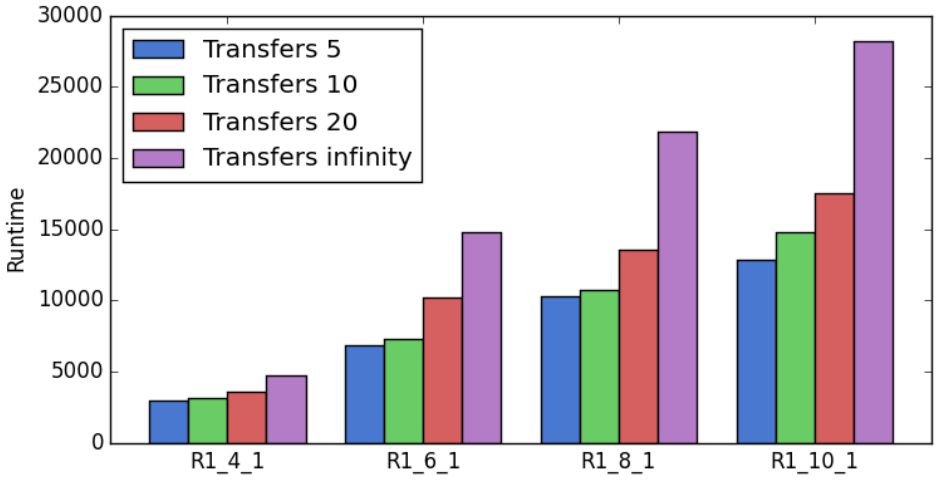


Figure 3.2 – Comparison of the impact of g on the average runtime (in seconds) for the training instances from Morais et al.; 5 runs were performed for each instance in each configuration

Set partitioning

In Table 3.3, we compare the influence of the set partitioning and matching period k_{SPM} for four different settings: calling SPM every 500, 1000, 2500, 5000 iterations. In order to maintain a fair balance in the time budget given to the SPM, and thus to see the influence of k_{SPM} on the quality and runtime of LNS+SPM, we set a time limit for each call to the SPM of 45, 90, 225 and 450 seconds respectively. We can observe that shorter periods help finding better solution, but that calling the SPM too often can increase the runtime. We thus set $k_{SPM} = 1000$ with a 90 seconds time limit for the MIP solver.

k_{SPM}	500	1000	2500	5000
Average gap (%)	0	-0.05	0.18	0.70
Average runtime (%)	0	-1.7	10.3	14.3

Table 3.3 – Comparison of the impact of period of the SPM procedure on the quality of the solution and on the runtime for 5 runs for each instance in the training set. *500* is taken as reference

3.6.3 Efficiency of set partitioning and matching

The SPM component constitutes the major contribution of this paper. To assess its efficiency, we compare the LNS+SPM with LNS, obtained by removing SPM (lines 16-23, and 23-24 in Algorithm 4). On Fig. 3.3 we present the convergence curve of LNS and LNS+SPM. On the training set, for 20 000 iterations, LNS finds solution that are 7.4% more expensive than LNS+SPM (4.2 % on the Wen et al. instances in the training set, and 10.6% on the Morais et al. instances in the training set). This improvement is observed very early in the search process. The SPM accounts for an increase in runtime of 20 % on average over the standard LNS. We can thus state that the SPM is a key feature of the proposed method that significantly increases the performance of the LNS for the VRPCD. On Fig. 3.3, we can also observe that after a few thousands iterations, the LNS is not able to improve the best solution by itself (there are plateaus in-between calls to the SPM). Thus the LNS contribution to the search lies in finding good legs that will then be matched by the SPM.

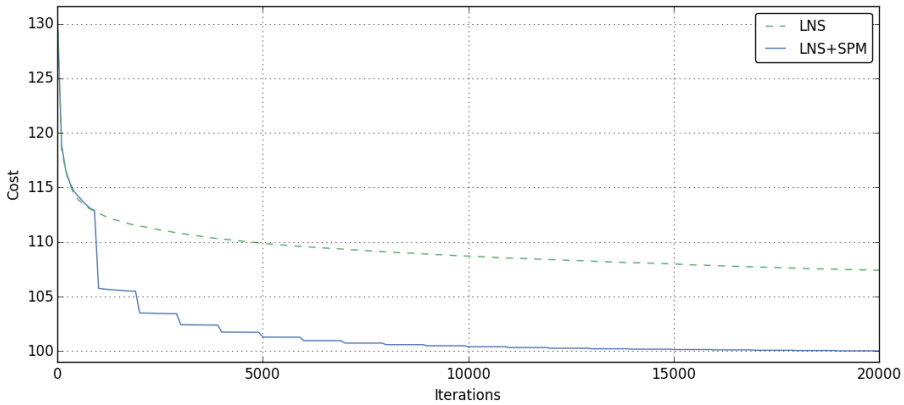


Figure 3.3 – Comparison of the evolution of the average solution quality for LNS and LNS+SPM; 10 runs were performed for each instance in the training set. The results have been normalized, with 100 representing the cost at the end for LNS+SPM

3.6.4 Results

In this section we recall the experiments conducted by other authors in the literature and we report our results.

Existing methods in the literature

There exists three methods for the VRPCD: that of Wen et al. [104], that of Tarantilis [97] and that of Morais [69]. It is worth pointing out that Wen at al.'s method was designed to find good solutions within 5 minutes of runtime (because of a real-life constraints). While they also report results for *no-limit* runs (the stopping condition was not based on time), one could think that design decisions they had to take for the 5 minutes constraint affects badly these results. On the other hand, Tarantilis and Morais et al. have greater time limit (3000 seconds). Our approach is similar to the latter with a greater emphasis on the quality of the solutions rather than small runtimes.

All methods have been evaluated on the Wen at al. dataset. Only the method of Morais et al. has been evaluated on the Morais et al. dataset. Wen et al. [104] performed 25 runs for each instance. They report average and best solution values within 5 minutes and best solution values when no time limit was considered. Tarantilis performed 3 runs for each instance with a time limit of 3000 seconds. He reports the best value found for each instance. Morais et al. performed 40 runs for each instance with a time limit of 3000 seconds for the Wen et al.'s instances and 1200 seconds for the Morais et al.'s instances. They report best and average values for each instances for four different variants of their algorithm. We choose to report the results of their SP-ILS variant as it is the best performing.

Best and average results

For each instance LNS+SPM was run ten times. In Table 3.4 and in Table 3.5 we report our average and best results for the Wen et al. data set and compare them to the existing methods. The lower bound are those reported in [104]. Columns *Gap* refer to gaps to lower bounds. For the methods of Wen et al, Tarantilis and Morais et al. the gaps are those reported in [69]. In Table 3.6 we present our best and average results for the Morais et al. data set and compare them to their results.
Our method outperforms existing methods with better average results for all instances, and by improving best known solutions for 30 instances out of 35. The proposed method improves the best known solutions by 0.78% on average on the Wen et al. dataset and by 2.16 % on average on the Morais et al. dataset. It is however impossible to predict what would have been the solution quality achieved by the previous methods if they would have been given more time.

Instance	LB	Wen et al.		Morais et al.		Grangier et al.		
		Value	Gap (%)	Value	Gap (%)	Value	Gap (%)	Time (s)
50a	6340.90	6534.2	3.05	6477.72	2.16	**6463.60**	**1.94**	209
50b	7201.89	7504.9	4.21	7443.92	3.36	**7427.45**	**3.13**	545
50c	7241.05	7440.0	2.75	7441.64	2.77	**7320.45**	**1.10**	261
50d	6887.93	7107.6	3.19	7063.17	2.54	**7040.59**	**2.22**	705
50e	7347.54	7629.4	3.84	7514.02	2.27	**7479.04**	**1.79**	272
100b	14200.48	14770.9	4.02	14498.69	2.10	**14376.71**	**1.24**	778
100c	13631.24	14145.0	3.77	13993.00	2.65	**13828.04**	**1.44**	1009
100d	13395.33	13949.6	4.14	13776.76	2.85	**13600.80**	**1.53**	738
100e	13745.60	14396.1	4.73	14159.96	3.01	**13958.75**	**1.55**	712
150a	19012.02	19871.3	4.52	19726.52	3.76	**19401.77**	**2.05**	1911
150b	20371.08	21284.0	4.48	20986.64	3.02	**20672.16**	**1.48**	1959
150c	19419.55	20320.5	4.64	20150.90	3.77	**19771.90**	**1.81**	1862
150d	20013.37	20891.3	4.39	20656.44	3.21	**20356.65**	**1.72**	1760
150e	19141.66	20034.6	4.66	19882.60	3.87	**19493.53**	**1.84**	1525
200a	26538.53	27683.9	4.32	27391.74	3.22	**26863.54**	**1.23**	2993
200b	26722.88	27989.1	4.74	27694.50	3.64	**27295.45**	**2.14**	2986
200c	25607.31	26654.1	4.09	26490.33	3.45	**26087.30**	**1.87**	2835
200d	26969.42	28088.2	4.15	27825.63	3.17	**27394.04**	**1.57**	2774
200e	25776.01	26868.6	4.24	26753.12	3.79	**26108.69**	**1.29**	2691

Table 3.4 – Comparison of average values for the Wen et al. dataset; LNS+SPM was run ten times for each instance with 20000 iterations as stop criterion

Instance	LB	Wen et al.		Tarantilis		Morais et al.		Grangier et al.		
		Value	Gap (%)	Value	Gap (%)	Value	Gap (%)	Value	Gap (%)	Time (s)
50a	6340.90	6471.9	2.07	**6450.28**	**1.73**	6453.08	1.77	6455.77	1.81	149
50b	7201.89	7410.6	2.9	7428.54	3.15	7434.90	3.24	**7320.77**	**1.65**	240
50c	7241.05	7330.6	1.24	**7311.77**	**0.98**	7317.35	1.05	**7311.77**	**0.98**	120
50d	6887.97	7050.3	2.36	**7021.39**	**1.94**	7035.50	2.14	7028.69	2.04	681
50e	7347.54	7516.8	2.30	**7451.42**	**1.41**	7482.01	1.83	7452.83	1.43	168
100b	14200.48	14526.1	2.29	14405.52	1.44	14441.01	1.69	**14349.60**	**1.05**	828
100c	13631.24	13967.8	2.47	13889.22	1.89	13932.78	2.21	**13784.70**	**1.13**	688
100d	13395.33	13763.3	2.75	**13564.23**	**1.26**	13708.81	2.34	13577.20	1.36	614
100e	13745.60	14212.7	3.4	14059.62	2.28	14122.32	2.74	**13943.10**	**1.44**	601
150a	19012.02	19537.3	2.76	19638.04	3.29	19532.28	2.74	**19358.90**	**1.82**	1886
150b	20371.08	20974.8	2.96	20922.27	2.71	20823.40	2.22	**20581.50**	**1.03**	2093
150c	19419.55	20126.5	3.64	20019.50	3.09	19964.59	2.81	**19726.80**	**1.58**	2187
150d	20013.37	20549.4	2.68	20600.33	2.93	20509.97	2.48	**20318.80**	**1.53**	1660
150e	19141.66	19848.5	3.69	19782.00	3.35	19716.87	3.01	**19449.50**	**1.61**	1546
200a	26538.53	27324.4	2.96	27397.31	3.24	27112.48	2.16	**26816.50**	**1.05**	2988
200b	26722.88	27637.7	3.42	27582.87	3.22	27509.08	2.94	**27215.10**	**1.84**	2804
200c	25607.31	26358.6	2.93	26425.29	3.19	26320.39	2.78	**25926.00**	**1.24**	2376
200d	26969.42	27749.7	2.89	27818.77	3.15	27686.75	2.66	**27328.70**	**1.33**	2457
200e	25776.01	26620.6	3.28	26704.81	3.60	26443.29	2.59	**26063.50**	**1.12**	2240

Table 3.5 – Comparison of the best solutions found for the Wen et al. dataset; LNS+SPM was run ten times for each instance with 20000 iterations as stop criterion

Instance	Morais et al.		Grangier at al.			
			Average		Best solution	
	Average value	Best value	Value	Time (min)	Value	Time (min)
R1-4-1	15530.10	15445.28	**15211.2**	59.6	**15170.0**	56.8
R1-4-2	14996.86	14850.75	**14666.2**	67.1	**14626.9**	57.2
R1-4-3	14414.90	14332.27	**14192.0**	63.6	**14146.6**	54.6
R1-4-4	15622.74	15521.49	**15336.4**	56.8	**15293.3**	56.3
R1-6-1	33776.88	33511.04	**32748.1**	153.4	**32598.1**	154.0
R1-6-2	33744.43	33540.56	**32726.5**	124.3	**32628.7**	123.4
R1-6-3	33478.77	33282.54	**32658.6**	145.9	**32571.5**	129.7
R1-6-4	33606.97	33468.72	**32850.6**	129.7	**32746.3**	139.9
R1-8-1	60611.89	60300.22	**59046.5**	181.3	**58831.1**	186.1
R1-8-2	58420.03	58113.83	**57137.6**	215.0	**56956.4**	179.2
R1-8-3	58859.03	58558.94	**57653.7**	248.2	**57421.7**	224.5
R1-8-4	60834.83	60502.26	**59427.9**	218.6	**59295.3**	233.7
R1-10-1	94687.60	94080.68	**92289.7**	302.5	**91949.2**	280.2
R1-10-2	93718.82	92792.34	**91360.2**	294.4	**91005.8**	297.7
R1-10-3	94200.82	93222.85	**91504.6**	217.1	**91317.0**	221.7
R1-10-4	94795.34	94372.82	**92804.0**	225.8	**92341.1**	227.3

Table 3.6 – Comparison of average values and best solution found for the Morais et al. dataset; LNS+SPM was run ten times for each instance with 20000 iterations as stop criterion

Note that runtimes for our method are higher than those of Tarantilis [97] and Morais et al. [69]. For each run, we have logged the evolution of the best known solution over time. Thus we can report the best known solution cost found by the LNS+SPM at any point in time. Taking into account the difference in speed of the processors used we can compare the proposed method with the previously mentioned methods. Following [56], we use CINT2000 and CINT2006 [1] to normalize the performance of processors: Tarantilis used a processor about 2.4 times slower than ours while Morais et al. used a processor about 2.0 slower than ours. For the Wen et al. instances, within comparable runtimes, the best values found by LNS+SPM are on average 0.4 % better than those reported by Tarantilis, while the average values by LNS+SPM are on average 1.26 % better than those reported by Morais et al.

3.7 Concluding remarks

This paper presents a new method based on large neighborhood search and periodic calls to a set partitioning based problem to solve the VRPCD. The set partitioning component is solved using both a MIP solver and a CP solver. Its addition help finding solutions significantly better than those obtained by the LNS alone. The proposed method has been tested on the instances of the literature and clearly outperforms existing methods by improving most of the previously best known results and all average results. Notably within comparable runtimes, it performs better than existing methods that focused on the quality of the solutions.
By solving, with a constraint programming solver, a dedicated subproblem, we propose a simple and efficient method to integrate precedence constraints in the SPP. As such it would be interesting to examine if this method could be adapted to solve other VRP with synchronization-related constraints.

[1] see the Standard Performance Evaluation Corporation web page: http://www.spec.org

4

Le problème de tournées de véhicules avec cross-docking et contraintes de ressources au cross-dock

Ce chapitre s'inscrit dans la suite du précédent. Au problème de tournées de véhicules avec cross-dock, nous rajoutons une contrainte de ressources, soit un nombre maximum de véhicules pouvant être traités simultanément au cross-dock. Nous définissons ainsi le problème de tournées de véhicules avec cross-docking et contraintes de ressources (Vehicle Routing Problem with Cross-Docking and Dock Resource constraints - VRPCDDR). Nous proposons une matheuristique inspirée de celle proposée pour le VRPCD. Une attention particulière a été portée au problème d'ordonnancement associé car, dans ce cas particulier, il est NP-Difficile. Deux techniques de vérification des contraintes temporelles ont été proposées : par des heuristiques de planification et par un modèle de programmation par contraintes. L'approche de set partitionning développée pour le VRPCD est modifiée pour intégrer la contrainte de capacité de traitement au cross-dock.

4.1 Article III: the vehicle routing problem with cross-docking and dock resource constraints

In logistics, cross-docking is a distribution strategy in which goods are brought from suppliers to an intermediate transshipment point, the so-called cross-dock, where they can be directly transferred (without storing) to another vehicle before being delivered. Compared to traditional distribution systems, cross-docking can help reducing delivery costs and delivery lead time, that is why it is used by many companies from different sectors: LTL, retail or automotive for example [100]. In the vehicle routing literature, the associated routing problem is called the Vehicle Routing Problem with Cross-Docking (VRPCD). It is a variant of the Pickup and Delivery Problem with Transfers with one compulsory transfer point: vehicles start by collecting items, then return to the cross-dock

where they unload/reload some items and eventually visit delivery locations. A few authors have proposed methods to solve this problem or variants of it, but to our knowledge, in most models there is no limit on the processing capacities of the cross-dock: as soon as a truck arrives, it immediately undergoes consolidation operations. However in practice, this may not be the case because of limited equipment or workforce, and trucks may wait before being unloaded. In fact, a wide range of the cross-docking literature is dedicated to the scheduling of operations at the cross-dock taking into account the cross-dock capacity. It has recently been pointed out [14, 55], that there is a need to consider the synchronization of local and network-wide cross-docking operations, in particular to take into account resource capacity at the cross-dock. To that end, in this paper, we introduce a new variant of the VRPCD in which the number of vehicles that can simultaneously be processed at the cross-dock is limited. We call it the Vehicle Routing Problem with Cross-Docking and Dock Resource Constraints (VRPCD-DR). The dock resource constraint, is a *resource synchronization constraint* as defined by Drexl [31] as vehicles compete to access a scarce resource: the processing capacity of the cross-dock. Very often resource synchronization constraints imply a difficult scheduling problem which is embedded within the vehicle routing problem. VRPCD-DR is no exception and the main contribution of this paper is on the integration of the scheduling problem associated with the dock resource constraints within a recently proposed large neighborhood search based method [41] for the VRPCD.

The remainder of this paper is organized as follows. A literature review is presented in Section 4.2, while the problem is defined in Section 4.3. In Section 4.4, we recall the method of [41], and Section 4.5 is devoted to the its adaptation to VRPCD-DR. Eventually, computational results are presented in Section 4.6.

4.2 Literature review

In this section we review the literature on two related vehicle routing problems: the vehicle routing problem with cross-docking and vehicle routing problems with resource synchronization.

4.2.1 The vehicle routing problem with cross-docking

A lot of cross-docking related problems exist such as: location, assignment of trucks to doors, inner flow optimization or routing. In particular, the vehicle routing problem with cross-docking consists in designing routes to pick up and deliver a set of transportation requests at minimal cost using a single cross-dock. It was introduced by Lee et al. [57] in a variant which imposes trucks to arrive at the exact same time at the cross-dock. Wen et al. [104] relaxed this last constraint only imposing precedence constraints based on the consolidation decisions and added time windows. This is the most studied variant, and it is the one we will refer to as the *vehicle routing with cross-docking* (VRPCD). Several heuristics have been proposed to solve it: based on tabu-search [104, 97], iterated local search [69] and large neighborhood search [41]. Other variants have been studied by Santos et al. [84, 87] which integrate a cost for transferring item at the cross-dock and have no temporal constraints. It was later extended in [85] with optional cross-dock return. These three articles proposed methods based on branch-and-price. The work of Petersen and Ropke [75] considers optional cross-dock return and multiple trips per day. They propose a parallel adaptive large neighborhood search to solve large real-life instances with up to 982 requests. Finally Dondo and Cerdà [29] consider a case where the number of

doors is fixed and smaller than the number of trucks. In particular each door is modeled individually: a time matrix models the time spent by a truck for moving from an inbound door a to an outbound door b. They solve two randomly generated instances with up to 70 requests with a mathematical model combined with a sweep heuristic. The VRPCD can be viewed as a special case of the pickup and delivery problem with transfers with only one compulsory transfer point [18]. Recently Guastaroba et al. [46] released a survey on intermediate facilities in freight transportation. For a general overview of cross-docking and cross-dock related problems we refer the reader to [11, 1, 100]

4.2.2 Resource synchronization

The expression *resource synchronization* appears in [31] as a way to model the following constraint:

> 'The total consumption of a specified resource by all vehicles must be less than or equal to a specified limit.'

Of course, this resource has to be scarce to be constraining, as vehicles *compete* to access it. Such resource constraints arise in many different vehicle routing problems usually when a special infrastructure or equipment is required: a docking station or parking space in airport cargo system [32], a berth in maritime transportation [45], a forest loader in forestry [34], a pump in ready mix-concrete delivery [90], an asphalt paver in public works [42]. Limited storage [32] or processing capacities [48] can also account for resource synchronization constraints. Hempsch and Irnich [48] also mention a situation where only a fixed number of vehicles (smaller than the total number) can perform *long* routes.
Many approaches have been applied to deal with these resource constraints in vehicle routing problems. In [32], Ebben et al. sequentially insert requests and check resource constraints in a predefined order. In [34], El Hachemi et al. use a dedicated constraint programming model, later combined with a greedy scheduling heuristic in [35], to ensure that resource constraints are satisfied during their entire solving process. In ready-mix concrete routing problems as well as in public works routing problem, orders are larger than truck capacity, as such they have to be split into several delivery operations that should not overlap. Resource synchronization constraints arise at pickup sites or at delivery sites or at both. Asbach et al. [5], Schmid et al. [90], Schmid et al. [89] and Grimault et al. [43] impose precedence constraints on the sequencing of operations to handle precedence constraints. Gronhaug et al. [45] rely on a time-discretized formulation in which resource synchronization constraints are easily expressed. Hempsch and Irnich [48] proposed a generic modeling for inter-routes constraints via the use of Resource Extension Functions (REF). In particular they focused on the use of REF as efficient feasibility tests in local search based algorithms when the solution is represented by a giant tour. From this literature review, it is clear that most resource synchronization constraints are in fact complex scheduling problems integrated into a vehicle routing problem. However the precise definition of the scheduling problem depends largely on the vehicle routing problem at stake.

4.3 The vehicle routing problem with cross-docking: model and resource synchronization constraint

This section presents the vehicle routing problem with cross-docking, and in particular the cross-dock model, as defined by Wen et al. [104]. It also introduces the two considered cross-dock resource models.

4.3.1 The vehicle routing problem with cross-docking

In the VRPCD, we consider a cross-dock c, a set of requests R, and a homogeneous fleet of vehicles K, each of capacity Q and based at o. Each request $r \in R$ has to be picked up at its pickup location p_r within its pickup time window $[e_{p_r}, l_{p_r}]$, and has to be delivered at its delivery location d_r within its delivery time window $[e_{d_r}, l_{d_r}]$. In case of early arrival, a vehicle is allowed to wait, but late arrivals are forbidden. We denote by P the set of pickup locations and by D the set of delivery locations.
Each vehicle starts at o, then goes to several pickup locations, arrives at the cross-dock where it unloads/reloads some requests. A vehicle then visits delivery locations and eventually returns at o. Note that a vehicle has to visit the cross-dock even if it does not unload nor reload any requests there. The sequencing of operations at the cross-dock is described in 4.3.2. The VRPCD is defined on a directed graph $G = (V, A)$, with $G = \{o\} \cup P \cup \{c\} \cup D$ and $A = \{(o, p) | p \in P\} \cup P \times P \cup \{(p, c) | p \in P\} \cup D \times D \cup \{(d, e) | d, e \in D\} \cup \{(d, o) | d \in D\} \cup \{(o, c), (c, o)\}$. With each arc $(i, j) \in A$ is associated a travel time $t_{i,j}$ and a travel cost $c_{i,j}$.
Solving the VRPCD involves finding $|V|$ routes, and a schedule for each route, such that the capacity and time-related constraints are satisfied, at minimal routing cost. An arc-based mathematical formulation can be found in [104].

4.3.2 Precedence constraints at the cross-dock

Following [104], if a vehicle k has to unload a set of requests R_k^- and reload a set requests R_k^+ at the cross-dock, the time spent at the cross-dock can be divided in up to four periods:

- Preparation for unloading. The duration δ_u of this period is fixed.
- Unloading of requests. The duration of this period depends on the quantity of products to unload, so for a vehicle k: $(\sum_{i \in R_k^-} q_i)/s_u$, where s_u corresponds to the unloading speed in quantity per time unit. All unloaded requests become available for reloading at the end of this period.
- Preparation for reloading. The duration δ_r of this period is fixed.
- Reloading of requests. Similarly to unloading, the duration of this period depends on the quantity of products to unload. For a vehicle k: $(\sum_{i \in R_k^+} q_i)/s_r$, where s_r corresponds to the reloading speed in quantity per time unit. All requests to reload must have been unloaded before the beginning of the reloading operation (preemption is not allowed).

Note that if a vehicle does not unload (resp. reload) any item at the cross-dock it does not have to undergo the preparation for unloading (resp. reloading) process, thus saving preparation time. So, in the case where a vehicle would not unload nor reload any item it would just have to stop at the cross-dock and could leave immediately.

4.3.3 Resource constraints models at the cross-dock

According to Van Belle et al. [100], the most common service mode of a cross-dock is called *exclusive*. In an exclusive mode, a dock door is either exclusively dedicated to unloading (inbound operations) or reloading (outbound operations). Such assignment is a decision taken at a strategical or tactical level. It cannot be modified in the VRPCD, which is an operational problem. In practice most cross-docks are I-shaped [100], with inbound doors on one side and outbound doors on the other. The flow of items is thus uni-directional, which can be easier to manage. This is a common situation but it is not mandatory.
Cross-docks usually have many doors (typically ranging from 40 to 150 according to [100]). However, as mentioned in [48] or [58], processing capacities may actually be lower than the number of doors. This comes from a limited workforce or special equipment to move the items within the cross-dock.
Because of the previous two considerations regarding operations at the cross-dock, we will consider two exclusive cross-dock models that integrate resource constraints:

- a case in which the total number of doors (both inbound and outbound) that can be processed simultaneously is limited to a number S. This is a simple way to model resource constraints due to limited workforce. We refer to this case as *shared*.

- a case in which the number of inbound doors (resp. outbound doors) that can be processed simultaneously is limited to a number I (resp O). This is a simple way to model a resource constraints due to special equipment. We refer to this case as *separated*.

Provided that at least two dock doors can be processed simultaneously, the scheduling problems at the cross-dock is NP-Hard as the scheduling problem $P2||C_{max}$ is included in them. In the rest, we will simply use *dock* to refer to the capacity in the number of docks processed simultaneously.
We call the VRPCD with these dock resource constraints, vehicle routing problem with dock resource constraints (VRPCD-DR). Figure 4.1 illustrates the sequencing of consolidation operations for a vehicle that unloads and reloads items at the cross-dock in the VRPCD-DR. When the vehicle arrives at the cross-dock, it can immediately be prepared for unloading, but it has to wait before being actually unloaded (W_U) because of a lack of available resources (note that such waiting time does not exist in the VRPCD). Once the unloading operation is done, the vehicle can proceed and move to an outbound door. Again preparation operation can be performed immediately, but it may wait before being actually reloaded (W_R). This waiting time can have two origins: first, not all items are available when it is ready (such situation can also arise in the VRPCD), second, there maybe a lack of available resources (which cannot occur in the VRPCD).

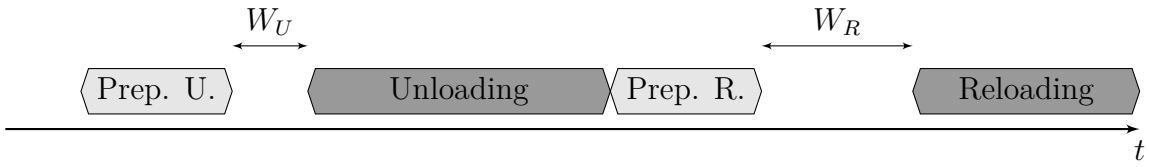


Figure 4.1 – Example of a time chart for a vehicle unloading and reloading at the cross-dock in the VRPCD-DR

4.4 Matheuristic for the VRPCD

In a previous paper [41], we proposed a matheuristic to solve the VRPCD, called LNS+SPM (Large Neighborhood Search + Set Partitioning and Matching). To our knowledge, this method is currently the best heuristic for the VRPCD. A sketch of LNS+SPM method is given in Algorithm 5. In details, it is based on large neighborhood search (l. 4-15) and periodically solving a set partitioning and matching problem (l.18). Every time the LNS finds a new solution, the pickup part and delivery part, called *legs*, of each route are added to a memory component (l. 16), called a *pool of legs*. The SPM aims to create the best possible solution from the legs in the pool. As shown in [41], this component significantly improves the quality of the solution compared to LNS alone. Hereafter, we recall the methods that are used in the LNS component and the SPM component.

Result : The best found solution $s^\star$
1 Pool of legs $\mathcal{L} := \emptyset$
2 Generate an initial solution s
3 $s^\star := s$
4 **while** *stop-criterion not met* **do**
5 $\quad s' := s$
6 $\quad$ **Destroy quantity:** select a number Φ of requests to remove from s'
7 $\quad$ **Operator selection:** select a destruction operator M^- and a repair operator M^+
8 $\quad$ **Destruction :** apply M^- to remove Φ requests from s', and put them in the requests bank of s'
9 $\quad$ **Repair:** apply M^+ to reinsert the requests in the requests bank in s'
10 $\quad$ **if** *acceptance criteria is met* **then**
11 $\quad\quad s := s'$
12 $\quad$ **end**
13 $\quad$ **if** *cost of* s' *is better than cost of* $s^\star$ **then**
14 $\quad\quad s^\star := s'$
15 $\quad$ **end**
16 $\quad$ Add legs of s' to $\mathcal{L}$
17 $\quad$ **if** *set partitioning and matching condition is met* **then**
18 $\quad\quad$ Perform set partitioning and matching with the legs in $\mathcal{L}$
19 $\quad\quad$ Update $s^\star$ and s if a new best solution has been found
20 $\quad\quad$ Perform pool management
21 $\quad$ **end**
22 **end**
23 **return** $s^\star$

Algorithm 5 : LNS+SPM of Grangier et al.

4.4.1 Large neighborhood search

Large Neighborhood Search [92] iteratively destroys (removes several requests from the solution) and repair (reinserts requests) the current solution using heuristics. In what follows, the destruction and repair methods used in [41] are summarized.

Destruction operators

When partially destroying a solution, a destruction operator M^- and a number Φ of requests to remove are selected. Unless stated otherwise, this operator is reused until Φ is reached. Random removal, worst removal, related removals and historical node-pair are inspired from [77], while transfer removal has been introduced for the VRPCD.

Random removal: a request is removed at random.

Worst removal: a request with a high removal gain is removed. The removal gain is defined as the difference between the cost of the solution with and without the request. Then, the requests are sorted in non increasing order of their removal gains and put in a list N. The request to remove is selected in a randomized fashion as in [82]: given a parameter p, a random number y between 0 and 1 is drawn. The request in position $y^p \times |N|$ is then removed.

Historical node-pair removal: each arc $(u, v) \in G$ is associated with the cost of the cheapest solution it appears in (initially this cost is set to infinity). For each request, the sum of the cost of its associated arcs in the current solution is computed. A randomized selection, similar to *worst removal*, is performed.

Related removals: these methods aim to remove related requests. Let the relatedness of requests i and j be $R(i, j)$. Two distinct relatedness measures are used: distance and time. The distance measure between two requests is the sum of the distance between their pickup points and the distance distance between their delivery points. The time measure is the sum of the absolute difference between their start of service at their pickup points and the absolute gap between their start of service at their delivery point. In both cases a small $R(i, j)$ indicates a high relatedness. A randomized selection, similar to *worst removal* (albeit with a non decreasing ordering), is performed.

Transfer removal: for each pair of routes (v_i, v_j), with $v_i \neq v_j$ the number of requests transferred from v_i to v_j is computed. Then a roulette wheel selection is applied on the pairs of routes (the score of a pair being the number of requests transferred), and the requests that are transferred between the routes in the selected pair are removed. If there are less transferred requests than the target number Φ to remove, the rest of the removals is performed with random removal.

Repair operators

In LNS, the unplanned requests are stored in a so-called *requests bank*. In the following, the operators to reinsert them in a solution are described. Best-insertion, 2-regret, 3-regret and 4-regrets are used as repair methods in LNS+SPM.

Best insertion: from all the requests r in the requests bank, the one with the cheapest insertion cost considering all possible insertion (with and without transfer) is performed.

Regret Insertion: for each request r in the requests bank and for each pair of vehicles (pickup vehicle, delivery vehicle), the cost of cheapest feasible insertion (if any) is computed. With these insertion options, the *k-regret value* of r is defined as $c_r^k = \sum_{i=1}^{k}(f_i - f_1)$, where f_1 is the cost of the cheapest insertion, f_2 is the cost of the second-cheapest insertion and so on, and k is a parameter. The cheapest insertion of the request with the highest regret value is performed.

Feasibility tests and reduction of neighborhood

In [41], we used an adaptation of forward time slacks [88] due to Masson et al. [64] to check time feasibility of insertions in constant time (capacity check in constant time is straightforward). To reduce the runtime, we do not consider all insertions with transfer. For a request r we only consider transfer from the five closest vehicles to its pickup point, to the five closest vehicles from its delivery point. For large scale instances, this halves the runtime.

4.4.2 Set Partitioning and Matching

Given a set of legs L, set partitioning and matching (SPM) aims to select a subset $\tilde{L}$ of L such that (1) each request is picked up and delivered by exactly one leg in $\tilde{L}$ and (2) legs in $\tilde{L}$ can be matched to form routes that respect time constraints. Each leg $l \in L$ has an associated routing cost c_l, the objective in the SPM is to minimize the sum of the costs of the selected legs. Notice that only non-dominated legs in $\mathcal{L}$ have to be considered in the SPM. A pickup (resp. delivery) leg l_i is said to be dominated by a leg l_j iff: l_i and l_j serve the same set of requests, $c_j < c_i$ and $a_j \leq a_i$ (resp. $b_j \geq b_i$) where a represents the arrival time at the cross-dock and b represents the departure time from the cross-dock.
The SPM is solved using a technique called *branch-and-check*, presented in Section 4.4.2. Its subproblem is a dedicated matching and scheduling subproblem, detailed in Section 4.4.2. The SPM is solved every thousand iterations with a time limit of ninety seconds. If the SPM is solved to optimality within the time limit, the legs in memory are kept, otherwise the pool is cleared.

Branch-and-check

We present *branch-and-check* [98] through the following optimization problem:

$$M1: \min c^\intercal x \tag{4.1}$$
$$Ax \leq b \tag{4.2}$$
$$H(x, y) \tag{4.3}$$
$$x \in \{0, 1\}^n \tag{4.4}$$
$$y \in \mathbb{R}^m \tag{4.5}$$

Assume that $H(x, y)$ represents a set of constraints that have a limited impact on the LP relaxation and/or are difficult to efficiently model in a MIP, but that could be handled relatively easily by a constraint programming (CP) solver. (4.1), (4.2) and (4.4) is a relaxation (M2) of (M1), that can be solved in branch-and-bound fashion. The general principle of branch-and-check is the following. To solve (M1), a branch-and-bound is performed on (M2). Whenever an integral solution of (M2) is found in the branch-and-bound process, a CP solver is called to check constraints (4.3). If they are satisfied, the

best solution found so far for (M1) is updated accordingly. Otherwise, this solution is rejected. In both cases the branch and bound process continues.

Application of branch-and-check to the VRPCD

For the SPM in the VRPCD, a classical set partitioning problem (SPP) is used as relaxation. For each request $r \in R$ and each leg $l \in L$, let $\lambda_{r,l}$ be a binary constant that indicates whether this request is served by this leg, and for each leg, let x_l be a boolean variable that indicate whether this leg is selected. The SPP on legs is then :

$$\min \sum_{l \in L} c_l * x_l \tag{4.6}$$

$$\sum_{l \in L_p} \lambda_{r,l} \times x_l = 1 \qquad \forall r \in R \tag{4.7}$$

$$\sum_{l \in L_d} \lambda_{r,l} \times x_l = 1 \qquad \forall r \in R \tag{4.8}$$

$$x_l \in \{0, 1\} \qquad \forall l \in L \tag{4.9}$$

The objective (4.6) is to minimize the cost of the selected legs while constraints (4.7) (resp. (4.8)) ensure that each pickup point (resp. delivery point) is covered by exactly one leg.

A solution to the SPP on legs, which involves a set of pickup legs denoted $\tilde{L_p}$ and a set of delivery legs denoted $\tilde{L_d}$, is a solution to the VRPCD iff there exists a matching of pickup legs and delivery legs to form routes that respects time constraints. For each pickup leg $l \in \tilde{L_p}$, let T_l be the set of delivery legs that deliver at least one request picked up by l. If a pickup leg l and a delivery leg l' are matched together to create a route, there are an associated unloading task $o^-_{ll'}$, with a set of requests $R^-_{ll'}$ being unloaded, and a reloading task $o^+_{ll'}$, with a set of requests $R^+_{ll'}$ being reloaded. These tasks have to be performed iff l and l' are in the same route.
The matching and scheduling problem is modeled as a constraint satisfaction problem, represented using notation from OPL (Optimization Programming Language [101]). In particular the model is based on the notion of interval variables and uses alternative constraints. As used here (from [51]):

> 'An interval variable represents an interval of time during which a task happen, and whose position in time is an unknown of the scheduling problem. An interval is characterized by a start value, an end value and a size. (...) An interval variable can be optional, that is, one can decide not to consider [it] in the solution schedule.'

In this model, we model alternative activities [7] by using alternative constraints (from [51]):

> 'An alternative constraint between an interval variable a and a set of interval variables $b_1, \ldots, b_n$ models an exclusive alternative between $b_1, \ldots, b_n$. If interval a is present, then exactly one of intervals $b_1, \ldots, b_n$ is present and a starts and ends together with this specific interval. Interval a is absent if and only if all intervals in $b_1, \ldots, b_n$ are absent.'

The matching and scheduling problem is then:

$$\text{Alternative}(t_l, \{o^-_{ll'}; \forall l' \in \tilde{L}_d\}) \qquad \forall l \in \tilde{L}_p \tag{4.10}$$

$$\text{Alternative}(t_{l'}, \{o^+_{ll'}; \forall l \in \tilde{L}_p\}) \qquad \forall l' \in \tilde{L}_d \tag{4.11}$$

$$\text{o}^-_{ll'}.IsFacultative \leftarrow True \qquad \forall l \in \tilde{L}_p, \forall l' \in \tilde{L}_d \tag{4.12}$$

$$\text{o}^+_{ll'}.IsFacultative \leftarrow True \qquad \forall l \in \tilde{L}_p, \forall l' \in \tilde{L}_d \tag{4.13}$$

$$\text{o}^-_{ll'}.IsPresent \iff o^+_{ll'}.IsPresent \qquad \forall l \in \tilde{L}_p, \forall l' \in \tilde{L}_d \tag{4.14}$$

$$\text{t}_{l'}.Start \geq t_l.End \qquad \forall l \in \tilde{L}_p, l' \in T_l \tag{4.15}$$

$$\text{o}^+_{ll'}.Start \geq o^-_{ll'}.End + \delta_r \qquad \forall l \in \tilde{L}_p, \forall l' \in \tilde{L}_d \; s.t. \; R^+_{ll'} \neq \emptyset \tag{4.16}$$

$$\text{o}^+_{ll'}.Start \geq o^-_{ll'}.End \qquad \forall l \in \tilde{L}_p, \forall l' \in \tilde{L}_d \; s.t. \; R^+_{ll'} = \emptyset \tag{4.17}$$

$$\text{o}^-_{ll'}.Start \geq a_l + \delta_u \qquad \forall l \in \tilde{L}_p, \forall l' \in \tilde{L}_d \; s.t. \; R^-_{ll'} \neq \emptyset \tag{4.18}$$

$$\text{o}^-_{ll'}.Start \geq a_l \qquad \forall l \in \tilde{L}_p, \forall l' \in \tilde{L}_d \; s.t. \; R^-_{ll'} = \emptyset \tag{4.19}$$

$$\text{o}^+_{ll'}.End \leq b'_l \qquad \forall l \in \tilde{L}_p, \forall l' \in \tilde{L}_d \tag{4.20}$$

For each pickup leg l, t_l is an interval variable that represents the associated unloading task that takes place at the cross dock. Alternative constraints (4.10) and (4.12) ensure that for each pickup leg l exactly one unloading task $o_{ll'}$ is scheduled and that it is equal to t_l. The same holds for delivery legs and reloading operations through variables $t_{l'}$ and constraints (4.11) and (4.13). Constraints (4.14) ensure that the unloading operation associated with the matching of pickup leg l and the delivery leg l' in the same vehicle is present iff the corresponding reloading operation is present as well. Constraints (4.15) ensure that all the reloading operations that depend on a pickup leg l start no earlier than the end of the unloading task associated with l. Constraints (4.16) and (4.17) ensure that when two legs are packed together, the delay between the two tasks respects the model presented in Section 4.3.2. Constraints (4.18) and (4.19) ensure that for each pickup leg, its corresponding unloading operation cannot start before the earliest feasible arrival time at the cross-dock. Constraints (4.20) ensure that for each delivery leg, its corresponding reloading operation is done by its latest feasible departure time.

4.5 Proposed matheuristic for the VRPCD-DR

In this section, we present the matheuristic that has been derived from [41] for the VRPCD-DR: Large Neighborhood Search+Set Partitioning and Scheduling (LNS+SPS). In Section 4.5.1 we focus on the feasibility test of insertions in the repair methods. Then we summarized the methods used in the LNS in Section 4.5.2. Because of the dock resource constraints, in place of SPM, we propose a Set Partitioning and Scheduling problem (SPS). We present it in Section 4.5.3. The overall method is summarized in Section 4.5.4.

4.5.1 Integration of dock resource constraints in LNS

In the repair methods of LNS, we need to ensure that the considered insertions are feasible both with respect to capacity and time-related constraints (time windows and dock resource). Capacity constraints can easily be checked in constant time, thus in what follows we focus on how to handle dock resource constraints. To that end, we start by presenting the scheduling model associated with dock resource constraints, then the

methods we propose to solve it and eventually the general structure of feasibility tests we use for maximal efficiency.

Scheduling problems associated with dock resource constraints

For each route $k \in K$, let a_k be the earliest feasible arrival time at the cross-dock and b_k the latest feasible departure time from the cross-dock. For an insertion that would insert request r in route k_1 for pickup and route k_2 for delivery, we can compute the new earliest feasible arrival time at the cross-dock of k_1: a'_1, and the new latest feasible departure time of k_2: b'_2 (provided that no time windows violation occurs in the pickup leg of k_1 and in the delivery leg of k_2, in which case we could immediately reject the insertion). Thus, dock resource constraints can be seen as a satisfaction scheduling problem at the cross-dock.
For each route $k \in K$, let T_k be the set of routes that deliver at least one request picked up by k; let t_k^- and t_k^+ be the associated unloading and reloading operations respectively; and let R_k^- and R_k^+ be the sets of requests R_k^- being unloaded and reloaded respectively. Let $isActive$ be an indicator function such that $isActive(o, h)$ is equal to 1 iff task o is being performed at instant h. Let H be the time horizon of the problem.
In the separated case the associated scheduling problem is:

$$\mathrm{t}_{k'}^+.Start \geq t_k^-.End \qquad \forall k \in K, k' \in T_k \tag{4.21}$$

$$\mathrm{t}_k^+.Start \geq t_k^-.End + \delta_r \qquad \forall k \in K; s.t.\ R_k^+ \neq \emptyset \tag{4.22}$$

$$\mathrm{t}_k^+.Start \geq t_k^-.End \qquad \forall k \in K; s.t.\ R_k^+ = \emptyset \tag{4.23}$$

$$\mathrm{t}_k^-.Start \geq a_k + \delta_u \qquad \forall k \in K; s.t.\ R_k^- \neq \emptyset \tag{4.24}$$

$$\mathrm{t}_k^-.Start \geq a_k \qquad \forall k \in K; s.t.\ R_{k'}^- = \emptyset \tag{4.25}$$

$$\mathrm{t}_k^+.End \leq b_k \qquad \forall k \in K \tag{4.26}$$

$$\sum_{k \in K} isActive(t_k^-, h) \leq I \qquad \forall h \in [0, H] \tag{4.27}$$

$$\sum_{k \in K} isActive(t_k^+, h) \leq O \qquad \forall h \in [0, H] \tag{4.28}$$

In the shared case the associated scheduling problem is:

$$(4.21 \text{ - } 4.26)$$

$$\sum_{k \in K} isActive(t_k^-, h) + isActive(t_k^+, h) \leq S \qquad h \in [0, H] \tag{4.29}$$

Constraints (4.21) ensure that all the reloading operations that depend on a route k start no earlier than the end of the unloading task associated with k. Constraints (4.22) and (4.23) ensure the delay between the unloading and reloading task of a route k respects the model presented in Section 4.3.2. Constraints (4.24) and (4.25) ensure that for each route, its corresponding unloading operation cannot start before the earliest feasible arrival time at the cross-dock. Constraints (4.26) ensure that for each route, its corresponding reloading operation is done by its latest feasible departure time. Constraints (4.27 and 4.28) models the separated case while constraint (4.29) models the shared case.

Proposed methods for the dock resource constraints

To solve the satisfaction scheduling problems of Section 4.5.1 we propose two methods: (1) using a third party CP solver or (2) using scheduling heuristics. When repeatedly calling a

third party solver we cannot neglect its overhead (e.g. model building, memory allocation of the solver, ...) potentially leading to very high runtimes. Scheduling heuristics are potentially faster, we could thus perform more LNS iterations within the same time budget, but they are likely to report more false negatives (stating that an insertion is infeasible although it is actually feasible). This is a runtime versus quality trade-off situation.

The scheduling heuristics we use are list heuristics: among a list of available tasks (a task is said to be available for scheduling if all its predecessors have already been scheduled) we select one task according to a given criterion and we try to schedule it. If we can schedule it, we update the resource constraints accordingly, we update the list of available tasks to schedule, and we repeat the process. If at one point we cannot schedule a task, we declare this insertion infeasible according to this scheduling heuristic. The four different selection criteria we use are listed hereafter.

First Come First Served (FCFS): we select the task with the earliest release date in the list of available tasks.

Earliest Due Date (EDD): we select the task with the earliest due date in the list of available tasks.

Most Successors First (MSF): we select the task with the largest number of successors tasks in the list of available tasks. By definition reloading tasks do not have successors, we break ties with the EDD rule.

Shortest Processing Time First (SPTF): we select the task with the shortest processing time in the list of available tasks.

Checking the feasibility of an insertion in the VRPCD-DR

First, observe that a necessary condition for an insertion to be feasible in the VRPCD-DR, is for it to be feasible in the VRPCD. As mentionned in Section 4.4.1, feasibility tests in the VRPCD can be performed in constant time. On the other hand feasibility test with respect to dock resource constraints in the VRPCD-DR cannot be done in constant time. Thus for maximal efficiency, we test the feasibility of an insertion as shown on Fig. 4.2: we start by checking if it is feasible for the VRPCD, if the insertion passes these tests, we test it with respect to dock resource constraints using either a CP solver or scheduling heuristics.

4.5.2 LNS operators

In the proposed LNS, we use the destruction operators of LNS+SPM (see Section 4.4.1). For repair operators, two facts should be taken into account: first, because of resource constraints, only a limited number of requests will be transferred (some requests that were transferred in the VRPCD may not be transferred in the VRPCD-DR). Thus, there is an incentive in creating routes without transfers. Second, as mentioned in Section 4.5.1, feasibility tests can no longer be performed in constant time, thus repair methods that check many insertions, such as k-regret with a high value of k, should be avoided. As such we use: best insertion, 2-regret, best insertion without transfer and 2-regret without

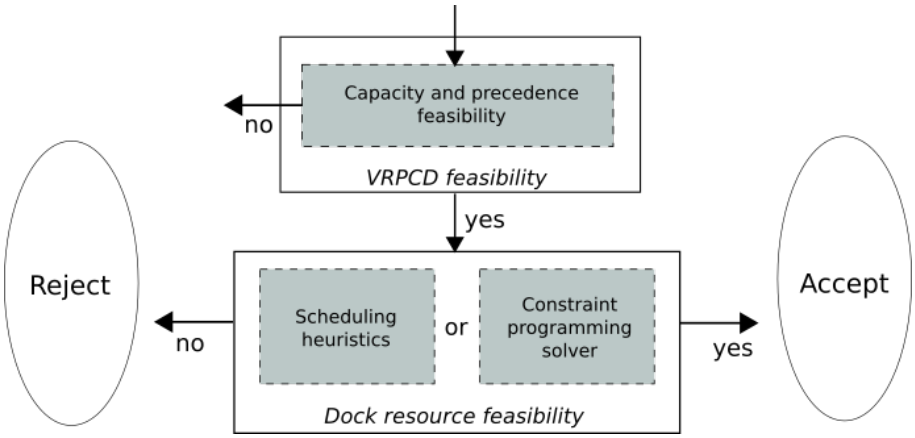


Figure 4.2 – Logical flow-chart of feasibility tests for the VRPCD-DR

transfer as repair methods. In the variants *without transfer*, only insertions without transfer are considered.

4.5.3 Integration of dock resource constraints in the periodical set partitioning based problem

For the VRPCD-DR, adding dock resource constraints makes the matching and scheduling subproblem of the SPM (see Section 4.4.2) significantly more difficult. Indeed preliminary tests showed that, very often, the CP solver could not find any solution to the subproblem within reasonable runtimes. On the other hand we noticed that the scheduling problems of Section 4.5.1 were solved by the CP solver within a relatively small time. Thus, because the set partitioning idea proved very efficient in [41], and to cope with the added complexity of the dock resource constraints, in place of SPM we propose to solve a Set Partitioning and Scheduling problem (SPS). It provides the best possible solution from a set of routes (instead of legs in the SPM). It is solved using branch-and-check. To that end a set partitioning problem on routes (similar to the SPP on legs of Section 4.4.2) is solved in a branch-and-bound fashion. At each integral node, the subproblem is a simpler satisfaction scheduling problem (no more matching) that corresponds to the one in Section 4.5.1.
This approach is efficient to select the best routes but the matching of legs in these routes may be improved. At the end of the SPS, we solve an optimization version of the matching and scheduling problem of Section 4.4.2, where the objective is to minimize the volume transferred at the cross-dock. In the shared case, the CP problem is:

$$\min \sum_{(l,l')\in\tilde{L}_p\times\tilde{L}_d} o^-_{ll'}.IsPresent \times \sum_{r\in R^-_{l,l'}} q_r \tag{4.30}$$

(4.10 - 4.20)

$$\sum_{l\in L_p} isActive(t_l,h) \le I \qquad \forall h\in[0,H] \tag{4.31}$$

$$\sum_{l'\in L_d} isActive(t'_l,h) \le O \qquad \forall h\in[0,H] \tag{4.32}$$

(4.30) minimizes the volume transferred at the cross-dock, while constraints (4.31) and (4.32) account for the capacity constraints and are similar to (4.27) and (4.28) of Section 4.5.1. A similar problem for the separated case can be formulated with an adaptation of (4.29).

4.5.4 Structure of the proposed method

Algorithm 6 and Algorithm 7 present a sketch of the proposed method: LNS+SPS. Algorithm 6 is similar to Algorithm 5 except for two changes. First, since feasibility tests are computationally more intensive, we consider a stop-criterion based on time, and enclose all instructions within time conditions. Second, the pool of routes $\mathcal{K}$ (l. 2 in Algorithm 7) plays the exact same role as the pool of legs $\mathcal{L}$ in Algorithm 5. The pool of legs $\mathcal{L}$ is still present, it now acts as a memory that can help improving routes in $\mathcal{K}$ before solving the SPS, as presented in Algorithm 7. We first apply dominance rules to the legs in $\mathcal{L}$ (see Section 4.4.2). Then we try to improve each route in $\mathcal{K}$ by replacing its pickup leg and/or its delivery leg by a non dominated equivalent in $\mathcal{L}$ (l. 2-4 in Algorithm 7). We remove dominated routes from $\mathcal{K}$ (l. 5 in Algorithm 7). A route is dominated iff there exists a route with a smaller cost that covers the same requests and can arrive earlier at the cross-dock and/or can leave later from the cross-dock. We solve the SPS, and eventually we try to improve the matching of legs in the solution. The initial solution is obtained by applying a 2-regret without transfer.

Result : The best found solution $s^\star$
1 Pool of legs $\mathcal{L} := \emptyset$
2 Pool of routes $\mathcal{K} := \emptyset$
3 Generate an initial solution s
4 $s^\star := s$
5 **while** *stop-criterion not met* **do**
6 $s' := s$
7 **Destroy quantity:** select a number Φ of requests to remove from s'
8 **Operator selection:** select a destruction operator M^- and a repair operator M^+
9 **Destruction :** apply M^- to remove Φ requests from s', and put them in the requests bank of s'
10 **Repair:** apply M^+ to reinsert the requests in the requests bank in s'
11 **if** *acceptance criteria is met* **then**
12 $s := s'$
13 **end**
14 **if** *cost of* s' *is better than cost of* $s^\star$ **then**
15 $s^\star := s'$
16 **end**
17 Add legs of s' to $\mathcal{L}$
18 Add routes of s' to $\mathcal{K}$
19 **if** *set partitioning and scheduling condition is met* **then**
20 Perform SPS($\mathcal{L}$, $\mathcal{K}$, $s^\star$, s');
21 **end**
22 **end**
23 **return** $s^\star$

Algorithm 6 : LNS+SPS

Input : pool of legs $\mathcal{L}$, pool of routes $\mathcal{K}$, solution $s^\star$, s' from Algorithm 6
Remove dominated legs from $\mathcal{L}$
for *each route* $k \in \mathcal{K}$ **do**
  Replace, if possible, its pickup leg and/or its delivery leg with a non-dominated equivalent in $\mathcal{L}$
end
Remove dominated routes from $\mathcal{K}$
Solve SPS with all routes in $\mathcal{K}$
if *a new best solution has been found* **then**
  Improve the matching of legs (as in Section 4.5.3)
  Update $s^\star$
end
if *Set partitioning was not solved to proven optimality* **then**
  Clear $\mathcal{K}$
end

Algorithm 7 : Perform SPS

4.6 Computational experiments

The algorithm is coded in C++ and uses CPLEX and CP Optimizer from IBM ILOG Cplex Optimization Studio 12.6.1 as MIP solver and CP solver, respectively. The experiments were conducted under Linux using an Intel Xeon X7350 @ 2.93 GHz. Only one core is used both by our code and third party solvers. We consider instances proposed by Wen et al [104], that range from 50 to 200 requests. They are based on real life data from a Danish logistics company. The termination criterion for all algorithms is based on time: 15 minutes for instances of size 50, 30 minutes for instances of size 100, 60 minutes for instances of size 150 and 120 minutes for instances of size 200. SPS time limit is 180 seconds. These time limits are between two and three times the runtimes reported for the VRPCD in [41]. As in [41], the number Φ of requests to remove in the repair phase of the LNS is drawn randomly in the interval [min(30, 10% of $|R|$), max(60, 20% of $|R|$)], acceptance criterion is descent.

4.6.1 Bound setting for the number of docks

Our aim was to determine when limiting the number of docks start being a constraint in the VRPCD-DR, that is to say the threshold dock value below which the VRPCD-DR could not no longer be solved with a combination of a VRPCD method and a post-process to minimize dock use. To that end we post-processed the solutions obtained in [41] for the VRPCD, and for all the ten solutions found for each instance, we solve optimization versions of the satisfaction scheduling problems introduced in Section 4.5.1. We take as objective: to minimize S in the shared case, and in the separated case, we only consider symmetric configurations where $I = O$ and we minimize I. In table 4.1, columns A correspond to the largest values obtained after five minutes of runtime for the CP solver, thus non constraining values as they can constantly be obtained by post-processing a VRPCD solution for a limited amount of time. As such, in our experiments for the VRPCD-DR, we will test dock values up to those reported in colums A. Columns B correspond to the smallest value obtained after two hours of post-process, that is to say the smallest dock value that can be obtained from a *good* solution to the VRPCD. We observe

that there is a small difference between columns A and B, thus showing a clear turning point after which dock resource start being a constraint. Overall resource constraints arise for dock values that corresponds to approximately 15% of the fleet size in the shared case and 10% of the fleet size in the separated case.

Instance	Avg. fleet size	Shared		Separated	
		A	B	A	B
50a	14.2	2	2	2	2
50b	16.1	2	2	2	2
50c	16.0	3	2	2	2
50d	15.0	2	2	2	2
50e	16.0	3	2	2	2
100b	31.0	4	4	3	3
100c	31.5	4	4	3	3
100d	29.2	4	4	3	3
100e	32.0	5	4	3	3
150a	45.4	7	5	5	4
150b	46.9	7	6	5	4
150c	45.9	7	6	5	4
150d	45.0	7	6	5	4
150e	46.0	7	6	5	4
200a	62.9	9	8	7	6
200b	62.0	9	8	7	6
200c	61.1	9	8	7	6
200d	62.0	9	8	7	6
200e	62.0	10	8	7	6

Table 4.1 – Dock value obtained when postprocessing for each instance all of ten solutions of [41]. Columns A refer to the to the worst solution (max dock use) obtained after five minutes, while columns B correspond to the best value (min dock use) obtained after two hours of post-processing.

4.6.2 Parameters tuning

In this section we evaluate and adjust several parameters. We start with the time limit for the CP solver in LNS feasibility tests, then we report the success rate of heuristics. After, we present the influence of the SPS frequency on the quality of solutions, and eventually we compare the performance of four possible configurations: with CP solver tests/with heuristic tests in LNS, with/without SPS. For tuning, we use instances 50b, 100b, 150b, 200b, and we consider the shared cross-dock configuration case.

CP solver time limit in feasibility tests

When checking the feasibility with respect to dock resource constraints, we need to set a time limit after which the CP solver will stop searching and declare the insertion infeasible. This avoids spending a large amount of time checking the feasibility of a single insertion. In Table 4.2, we compare four different time limits. In the final setting, the time limit of the CP Solver is set to 0.01 seconds as a compromise between the runtime (and thus the number of iterations that can be performed) and the percentage of time limit hit.

CP Time Limit (s)	1	0.1	0.01	0.001
Runtime	1	0.33	0.12	0.10
Time limit hit (%)	0.5	12.9	20.1	27.3

Table 4.2 – Comparison of four different time limits for the CP solver, when used as feasibility test in the shared case. Runtime is normalized with 1 representing the runtime for a CP solver time limit of 1s. *Time limit hit* represents the percentage of calls for which the CP solver could not find an answer within the time limit. Figures reported for one thousand LNS iterations, two runs were performed in each case.

Heuristics performance

As mentioned in Section 4.5.1, using CP versus using scheduling heuristics for feasibility tests is a quality/runtime trade-off. For each instance in the training set, we performed two runs with LNS, with the stop-criterion set to one thousand iterations. We count how many time insertions were reported feasible by heuristics and how many times they were reported feasible by the CP solver with a time limit of 1 second (according to Table 4.2, with this time limit we can consider that the CP solver gives an accurate answer most of the time). On average, 71.1% of feasible insertions are reported feasible by heuristics. Performing one thousand LNS iterations with feasibility tests performed by heuristics takes only 18.2% of the time taken with CP solver tests (with 0.01s as time limit).

SPS frequency

In [41], the SPM was solved twenty times per run (every thousand iterations with a stop-criterion of twenty thousands iterations). In the VRPCD-DR, feasibility tests are computationally intensive, and the number of iterations performed within a given time budget depends not only on the size of the instance but also on the dock value. As such, we propose both a stop-criterion and a SPS-criterion based on time. In Table 4.3 and Table 4.4, we compare three different settings for the SPS frequency : 10, 20 and 40 calls within the time limit. Accordingly, the number of calls per run is set to 20 for heuristic test and 10 for CP solver test.

Number of calls	10	20	40
Average gap (%)	0	-0.70	-0.29

Table 4.3 – Comparison of the impact of the number of SPS calls per run on the quality of the solution for heuristic feasibility tests for three different settings. Five runs were performed for each dock value for each instance in the training set. *10 calls* is taken as reference for the gap

Number of calls	10	20	40
Average gap (%)	0	+0.40	+0.68

Table 4.4 – Comparison of the impact of the number of SPS calls per run on the quality of the solution for CP solver tests for three different settings. Five runs were performed for each dock value for each instance in the training set. *10 calls* is taken as reference for the gap

Performance comparison

On Fig. 4.3 we present the convergence curves of LNS and LNS+SPS over time for both CP solver and heuristics feasibility tests. From this graph, we can observe two things. First, as in the VRPCD, periodically solving a set partitioning based problem significantly improves performance compared to LNS alone: -6.96% for heuristic test and -7.92% for CP solver test on average. Second, the best performing method is LNS+SPS with heuristic feasibility tests, as it finds solutions that are 1.19% better on average than LNS+SPS with CP solver test. In the rest, we thus use LNS+SPS with heuristic feasibility test.

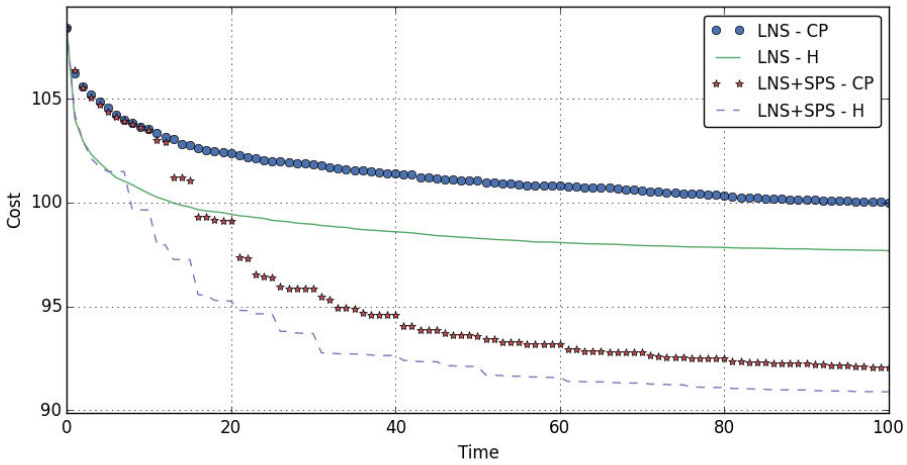


Figure 4.3 – Comparison of the evolution of the average solution quality over time (in percentage of time limit) for four different configurations: with CP solver/with heuristics tests, with/without SPS. The results aggregate 5 runs for each dock value in the shared case for instances in the training set. They have been normalized, first by instance then by method, with 100 representing the cost at the end of LNS with CP solver tests (LNS-CP)

4.6.3 Results

Table 4.5 and table 4.6 present the results in the shared case and in the separated case respectively. A dock value of 0 corresponds to the case where no transfer is allowed. These results highlight the increased complexity induced by integrating dock resource constraints. Indeed, for each instance, at maximum dock value, the VRPCD-DR can be solved as a VRPCD (see Section 4.6.1), as such a solution with 0% gap exists, and LNS+SPS finds solutions that are 1.6% more expensive in the shared case and 1.5% more expensive in the separated case. This remains satisfactory. The algorithm shows relatively good performance in terms of stability: the difference between the average value and the best value of the five runs for each instance and each dock value is 0.6% on average and at most 1.4% in the shared case and 0.6% on average and at most 3.2% in the separated case.
Regarding routing costs: integrating dock resource constraints implies an increase in cost. Comparing the two systems, the shared case costs are slightly smaller than separated costs (e.g, for instance 150b, 5.85% for 2 shared docks compared to 6.05 as shown on Fig. 4.4). This differences increases with the number of docks.

4.7 Conclusion

This paper presents an adaptation of the method proposed in [41], which is based on large neighborhood search and periodic calls to a set partitioning based problem, to solve an extension of the VRPCD that includes resource synchronization constraints at the cross-dock. To deal with these constraints, scheduling heuristics and a CP model have been used as feasibility tests for insertions in LNS. In this case, experiments have showed that heuristics are the most efficient compromise. Compared to [41], because of the increased complexity induced by resource synchronization constraints, the set partitioning

Instance	Dock	Average		Best	
		Value	Gap (%)	Value	Gap (%)
50a	0	6882.4	6.5	6871.93	6.4
	1	6682.72	3.4	6628.05	2.7
	2	6530.81	1.0	6471.48	0.2
50b	0	8027.87	8.1	8021.81	9.6
	1	7609.07	2.4	7541.33	3.0
	2	7481.99	0.7	7470.78	2.0
50c	0	7857.77	7.3	7827.67	7.1
	1	7505.34	2.5	7473.94	2.2
	2	7449.88	1.8	7397.38	1.2
	3	7338.88	0.3	7330.08	0.3
50d	0	7760.32	10.2	7760.11	10.4
	1	7272.57	3.3	7206.48	2.5
	2	7108.91	1.0	7076.13	0.7
50e	0	8157.77	9.1	8156.94	9.4
	1	7843.08	4.9	7772.39	4.3
	2	7616.54	1.8	7554.75	1.4
	3	7537.85	0.8	7473.56	0.3
100b	0	15636.26	8.8	15628.7	8.9
	1	15322.86	6.6	15234.2	6.2
	2	15181.04	5.6	15073.8	5.0
	3	14929.18	3.8	14810.3	3.2
	4	14700.96	2.3	14620.3	1.9
100c	0	14915.92	7.9	14915.6	8.2
	1	14654.16	6.0	14611.5	6.0
	2	14456.6	4.5	14395.0	4.4
	3	14353.9	3.8	14188.2	2.9
	4	14145.78	2.3	14081.3	2.2
100d	0	14860.46	9.3	14832.5	9.2
	1	14424.74	6.1	14338.9	5.6
	2	14316.7	5.3	14207.4	4.6
	3	14070.18	3.5	13976.2	2.9
	4	13866.24	2.0	13774.7	1.5
100e	0	15095.92	8.1	15091.5	8.2
	1	14819.26	6.2	14733.8	5.7
	2	14783.82	5.9	14622.8	4.9
	3	14515.0	4.0	14384.7	3.2
	4	14357.26	2.9	14258.4	2.3
	5	14281.0	2.3	14085.3	1.0
150a	0	20859.22	7.5	20807.7	7.5
	1	20534.8	5.8	20406.4	5.4
	2	20505.68	5.7	20342.5	5.1
	3	20248.54	4.4	20090.1	3.8
	4	20110.14	3.7	19988.6	3.3
	5	19822.08	2.2	19740.0	2.0
	6	19725.04	1.7	19629.7	1.4
	7	19603.78	1.0	19541.6	0.9
150b	0	22272.64	7.7	22236.5	8.0
	1	22018.68	6.5	21934.5	6.6
	2	21882.36	5.9	21825.0	6.0
	3	21651.48	4.7	21585.3	4.9
	4	21432.16	3.7	21360.2	3.8
	5	21227.66	2.7	21097.6	2.5
	6	21065.9	1.9	20986.2	2.0
	7	20963.44	1.4	20899.9	1.5
150c	0	21313.88	7.8	21295.7	8.0
	1	21095.74	6.7	20959.6	6.2
	2	20945.68	5.9	20854.4	5.7
	3	20869.2	5.5	20642.1	4.6
	4	20685.7	4.6	20559.7	4.2
	5	20475.68	3.6	20369.1	3.3
	6	20208.76	2.2	20155.8	2.2
	7	20153.68	1.9	20100.9	1.9
150d	0	21962.58	7.9	21951.3	8.0
	1	21735.46	6.8	21536.0	6.0
	2	21609.34	6.2	21442.6	5.5
	3	21448.08	5.4	21368.0	5.2
	4	21504.76	5.6	21289.8	4.8
	5	21040.52	3.4	20874.7	2.7
	6	20806.38	2.2	20642.2	1.6
	7	20715.78	1.8	20608.5	1.4

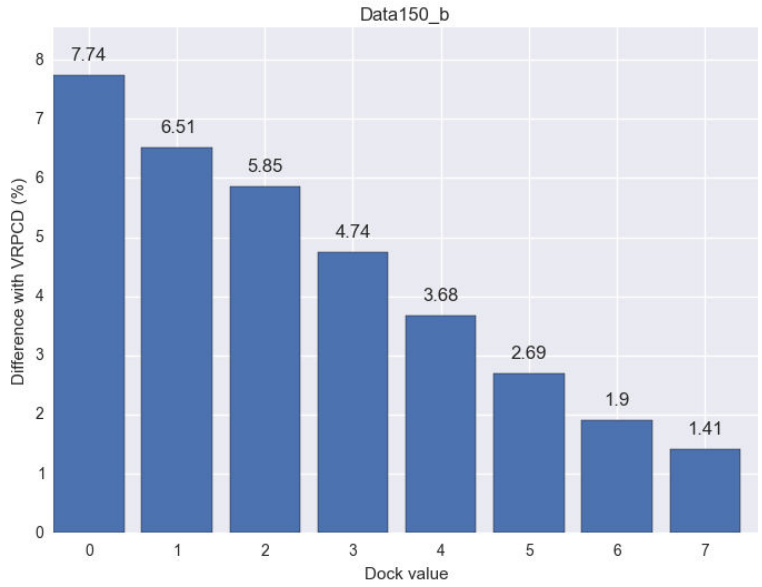
Instance	Dock	Average		Best	
		Value	Gap (%)	Value	Gap (%)
150e	0	21036.12	7.9	21019.3	8.1
	1	20827.96	6.8	20703.5	6.4
	2	20774.68	6.6	20587.3	5.9
	3	20599.16	5.7	20320.2	4.5
	4	20398.18	4.6	20341.0	4.6
	5	20346.68	4.4	20245.7	4.1
	6	20073.92	3.0	20025.5	3.0
	7	19915.4	2.2	19773.1	1.7
200a	0	28779.68	7.1	28760.6	7.2
	1	28507.12	6.1	28423.1	6.0
	2	28516.14	6.2	28263.0	5.4
	3	28273.0	5.2	28159.1	5.0
	4	28323.38	5.4	28146.5	5.0
	5	28151.24	4.8	28054.8	4.6
	6	28099.32	4.6	27959.8	4.3
	7	27922.8	3.9	27782.5	3.6
	8	27704.08	3.1	27527.8	2.7
	9	27497.36	2.4	27391.5	2.1
200b	0	29168.18	6.9	29082.8	6.9
	1	28962.52	6.1	28771.7	5.7
	2	28809.76	5.5	28751.4	5.6
	3	28784.88	5.5	28514.1	4.8
	4	28589.7	4.7	28546.4	4.9
	5	28374.36	4.0	28186.6	3.6
	6	28446.44	4.2	28234.3	3.7
	7	28156.7	3.2	28091.2	3.2
	8	27889.54	2.2	27677.2	1.7
	9	27630.92	1.2	27516.8	1.1
200c	0	28153.28	7.9	28119.3	8.5
	1	27996.14	7.3	27819.7	7.3
	2	27744.12	6.4	27555.7	6.3
	3	27629.96	5.9	27381.3	5.6
	4	27534.98	5.5	27207.7	4.9
	5	27522.48	5.5	27315.8	5.4
	6	27236.38	4.4	27135.5	4.7
	7	26921.14	3.2	26820.2	3.4
	8	26685.4	2.3	26570.1	2.5
	9	26583.44	1.9	26477.1	2.1
200d	0	29463.02	7.6	29446.8	7.8
	1	29328.88	7.1	29227.0	6.9
	2	29126.9	6.3	28993.7	6.1
	3	28962.52	5.7	28825.7	5.5
	4	28866.58	5.4	28776.3	5.3
	5	28777.12	5.0	28523.2	4.4
	6	28708.02	4.8	28546.4	4.5
	7	28467.38	3.9	28283.7	3.5
	8	28208.72	3.0	28129.6	2.9
	9	28002.7	2.2	27877.6	2.0
200e	0	28225.92	8.1	28139.3	8.0
	1	28106.28	7.7	28001.5	7.4
	2	27935.84	7.0	27659.2	6.1
	3	27789.42	6.4	27566.0	5.8
	4	27610.22	5.8	27432.2	5.3
	5	27421.46	5.0	27113.3	4.0
	6	27324.54	4.7	27067.2	3.9
	7	27061.8	3.7	26967.5	3.5
	8	26858.48	2.9	26772.0	2.7
	9	26665.24	2.1	26543.9	1.8
	10	26677.62	2.2	26600.4	2.1

Table 4.5 – Average values and best solution found in the shared cross-dock configuration case; LNS+SPS was run five times for each instance. Columns *Gap* refer to the gap to average values and best solutions reported in [41] for the VRPCD

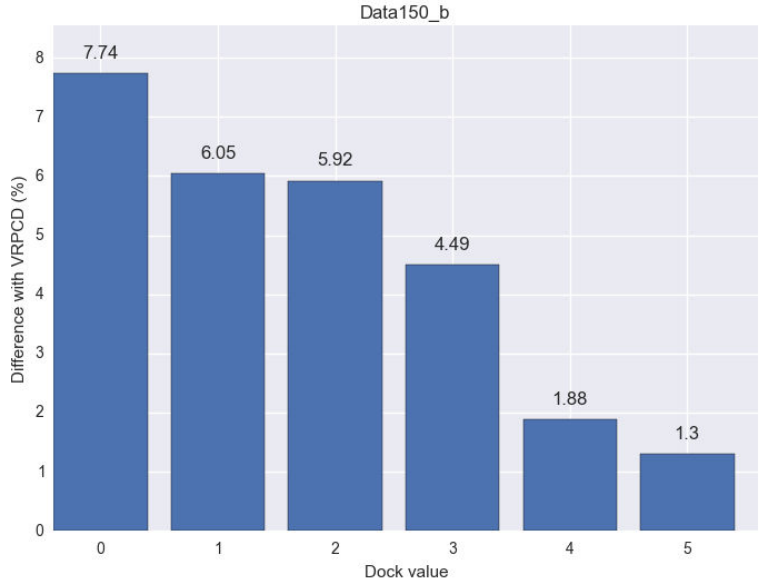
Instance	Dock	Average		Best	
		Value	Gap (%)	Value	Gap (%)
50a	0	6882.4	6.5	6871.93	6.4
	1	6614.32	2.3	6554.55	1.5
	2	6574.13	1.7	6545.34	1.4
50b	0	8027.87	8.1	8021.81	9.6
	1	7520.61	1.3	7496.96	2.4
	2	7462.01	0.5	7451.21	1.8
50c	0	7857.77	7.3	7827.67	7.1
	1	7428.2	1.5	7385.09	1.0
	2	7359.79	0.5	7335.97	0.3
50d	0	7760.32	10.2	7760.11	10.4
	1	7169.38	1.8	7127.77	1.4
	2	7078.23	0.5	7054.87	0.4
50e	0	8157.77	9.1	8156.94	9.4
	1	7705.03	3.0	7649.99	2.6
	2	7546.88	0.9	7478.02	0.3
100b	0	15636.26	8.8	15628.7	8.9
	1	15184.1	5.6	15088.4	5.1
	2	15163.32	5.5	14887.9	3.8
	3	14718.0	2.4	14618.6	1.9
100c	0	14915.92	7.9	14915.6	8.2
	1	14484.24	4.7	14360.5	4.2
	2	14470.17	4.6	14313.7	3.8
	3	14130.18	2.2	14029.0	1.8
100d	0	14860.46	9.3	14832.5	9.2
	1	14413.18	6.0	14331.3	5.6
	2	14120.35	3.8	14036.7	3.4
	3	13911.1	2.3	13762.2	1.4
100e	0	15095.92	8.1	15091.5	8.2
	1	14801.8	6.0	14736.4	5.7
	2	14643.54	4.9	14459.3	3.7
	3	14329.12	2.7	14251.0	2.2
150a	0	20859.22	7.5	20807.7	7.5
	1	20482.68	5.6	20397.9	5.4
	2	20430.4	5.3	20298.3	4.9
	3	20161.44	3.9	19866.7	2.6
	4	19818.47	2.1	19683.2	1.7
	5	19709.34	1.6	19599.3	1.2
150b	0	22272.64	7.7	22236.5	8.0
	1	21922.38	6.0	21770.5	5.8
	2	21896.35	5.9	21733.3	5.6
	3	21600.98	4.5	21422.4	4.1
	4	21060.7	1.9	20957.2	1.8
	5	20940.14	1.3	20877.8	1.4
150c	0	21313.88	7.8	21295.7	8.0
	1	21060.12	6.5	20986.5	6.4
	2	21057.4	6.5	20653.7	4.7
	3	20707.24	4.7	20594.7	4.4
	4	20431.26	3.3	20299.3	2.9
	5	20071.94	1.5	19936.7	1.1
150d	0	21962.58	7.9	21951.3	8.0
	1	21714.83	6.7	21549.1	6.1
	2	22019.32	8.2	21327.4	5.0
	3	21424.15	5.2	20967.3	3.2
	4	20964.9	3.0	20784.2	2.3
	5	20666.2	1.5	20553.8	1.2

Instance	Dock	Average		Best	
		Value	Gap (%)	Value	Gap (%)
150e	0	21036.12	7.9	21019.3	8.1
	1	20888.67	7.2	20774.5	6.8
	2	20722.35	6.3	20625.8	6.0
	3	20551.0	5.4	20280.7	4.3
	4	20347.46	4.4	20179.9	3.8
	5	19884.9	2.0	19789.0	1.7
200a	0	28779.68	7.1	28760.6	7.2
	1	28530.67	6.2	28419.3	6.0
	2	28435.3	5.9	28267.4	5.4
	3	28173.58	4.9	28063.0	4.6
	4	27994.62	4.2	27881.1	4.0
	5	27702.8	3.1	27555.8	2.8
	6	27458.1	2.2	27333.0	1.9
	7	27259.12	1.5	27177.7	1.3
200b	0	29168.18	6.9	29082.8	6.9
	1	28890.62	5.8	28695.7	5.4
	2	28784.8	5.5	28526.9	4.8
	3	28450.8	4.2	28329.2	4.1
	4	28200.94	3.3	28054.6	3.1
	5	28049.1	2.8	27940.8	2.7
	6	27653.76	1.3	27581.9	1.3
200c	0	28153.28	7.9	28119.3	8.5
	1	27735.95	6.3	27703.7	6.9
	2	27656.33	6.0	27451.2	5.9
	3	27435.67	5.2	27320.1	5.4
	4	27277.44	4.6	27083.6	4.5
	5	26846.82	2.9	26776.0	3.3
	6	26521.98	1.7	26387.4	1.8
	7	26362.82	1.1	26263.4	1.3
200d	0	29463.02	7.6	29446.8	7.8
	1	29172.35	6.5	29013.9	6.2
	2	28888.03	5.5	28775.9	5.3
	3	28907.9	5.5	28754.6	5.2
	4	28778.5	5.1	28633.4	4.8
	5	28403.72	3.7	28294.1	3.5
	6	27974.1	2.1	27846.6	1.9
	7	27762.08	1.3	27654.5	1.2
200e	0	28225.92	8.1	28139.3	8.0
	1	28046.3	7.4	27972.5	7.3
	2	27683.53	6.0	27555.4	5.7
	3	27360.9	4.8	27186.0	4.3
	4	27229.36	4.3	27025.6	3.7
	5	26943.35	3.2	26823.2	2.9
	6	26593.2	1.9	26530.5	1.8
	7	26687.88	2.2	26593.6	2.0

Table 4.6 – Average values and best solution found in the separated cross-dock configuration case; LNS+SPS was run five times for each instance. Columns *Gap* refer to the gap to average values and best solutions reported in [41] for the VRPCD



(a) *Shared* cross-dock configuration



(b) *Separated* cross-dock configuration

Figure 4.4 – Influence of the dock value for instance 150b. Five runs were performed for each dock value. The y-axis represents the average gap with respect to the average value reported in [41]

based problem had to be adapted with a simpler subproblem. As in [41], adding this component helps finding solutions significantly better than those obtained by LNS alone. The proposed method has been tested on instances from the literature, where it shows an increase in routing costs with the decrease in cross-dock capacity.

5

Un problème réel de chargement 3D

Cette contribution est le résultat d'un stage de cinq mois effectué au sein du JDA Labs de la société JDA Software à Montréal entre décembre 2014 et avril 2015. Le problème posé est un problème de chargement de container pour un grand fabricant de pneus : étant donné un ensemble de pneus de matériel agricole, existe-t-il un plan de chargement respectant les règles de chargement et de sécurité ? Ce problème se distingue des autres problèmes de chargements 3D de la littérature car les formes à charger ne sont pas des boites rectangulaires mais des cylindres. Par ailleurs, il existe une très grande variété de tailles de pneus de matériel agricole, les chargements sont donc très différents d'une instance à l'autre. La méthode de résolution proposée est une méthode en deux phases. Dans un premier temps, on génère l'ensemble des structures (soit le regroupement de plusieurs pneus) *intéressantes* puis un problème de couverture par ensembles visant à minimiser la surface est résolue. Les structures sélectionnées sont ensuite positionnées dans le container par un algorithme de programmation dynamique. Celui-ci exploite certaines caractéristiques du problème et permet d'obtenir de bonnes performances dans des temps jugés acceptables par l'entreprise. La méthode a été testée sur des instances réelles, où elle a prouvé son efficacité.

5.1 Article IV: A two phase algorithm for a real-life 3D container loading problem

The problem addressed in this paper is a real-life 3D container loading problem of agricultural tires occurring in one of the leading tire manufacturers. There exists many references of agricultural tires ranging from relatively small diameters (around 50 centimeters), for example for the front wheels of special tractors, to very large diameters (up to 2 meters), for example for the rear wheels of heavy duty tractors. As such, each shipment usually contains a heterogeneous set of tires and shipments differ from one to another. Those are the reasons why loading plans have to be determined on a per case basis. The problem at stake is: given a shipment and a container, is it possible to find a loading plan that respects all the positioning rules defined by the company? A shipment may be too large for a single container, but deciding which tire will be part of this shipment and which tire

will not, depends on sales forecasting and due dates (this is out of the scope of this study). As a result, the proposed algorithm has two functions: it establishes if a shipment can be loaded in a container, and, if so, it proposes a loading plan. Hence, the considered problem is a Constraint Satisfaction Problem (CSP).
In the typology on cutting and packing proposed by Wäscher et al. [103], this problem has the following features: three dimensional, strongly heterogeneous assortment, one large object and regular small items. Notice that it does not completely fit in this typology as it is a satisfaction problem and not an optimization problem. The majority of packing problems deal with orthogonal packing of rectangular items, for example the Three-Dimensional Bin Packing Problem [61]. Dealing with circular shapes has some important consequences: non-overlapping constraints are non-linear and some strategies for rectangular items, such as *touching perimeter* are not valid. For a recent survey on circle and sphere packing, we refer the reader to [50]. Note that, because tires can lie flat or vertically, the problem presented in this paper is different from any other problem we are aware of in the literature on circle packing. As a real-life problem, many constraints due to stability, and/or safety restrictions arise. We refer the reader to [10] for a recent survey on constraints in container loading. These constraints have guided our solving methodology. In particular, we first create shapes that then simplify the location part, this is similar to *templates* in [33] for furniture packing. Our location algorithm is based on recursive dynamic programming, which is a common technique for packing problems as mentioned in [49].

The remainder of this paper is the following. We present the problem in details in Section 5.2. Our solution methodology is described in Section 5.3, and eventually, we present results in Section 5.4.

5.2 Problem description

In what follows we call *shipment* a set of tires, each tire $t \in T$ is associated with a quantity q_t, and we call *loading plan* the precise position of each of these tires in a given container. The problem at stake is a CSP: for a given shipment and a given container, is it possible to find a valid loading plan, that is to say a loading plan that respects loading and safety rules.
In what follows, we first present conventions used throughout this paper, then the shapes that can be formed by grouping tires, and then we detail the constraints of the problem.

5.2.1 Conventions

For a clearer presentation, for a tire t we call diameter d_t and width w_t the measurements depicted on Fig. 5.1. The container is presented on Fig. 5.2. We call *length* its dimension H_x along the x-axis, *width* its dimension H_y along the y-axis and *height* its dimension H_z along the z-axis (not represented on Fig. 5.2). The container is rear loaded, and we assume that the door is located at $x = H_x$. As in most cases the container actually models a trailer, when talking about a position (x, y), we call *driver side* all positions whose abscissa x' is such that $x' < x$ and *door side* all the positions whose abscissa is such that $x' > x$. In case of a trailer, when in movement, the truck is driving along the x-axis towards negative values.

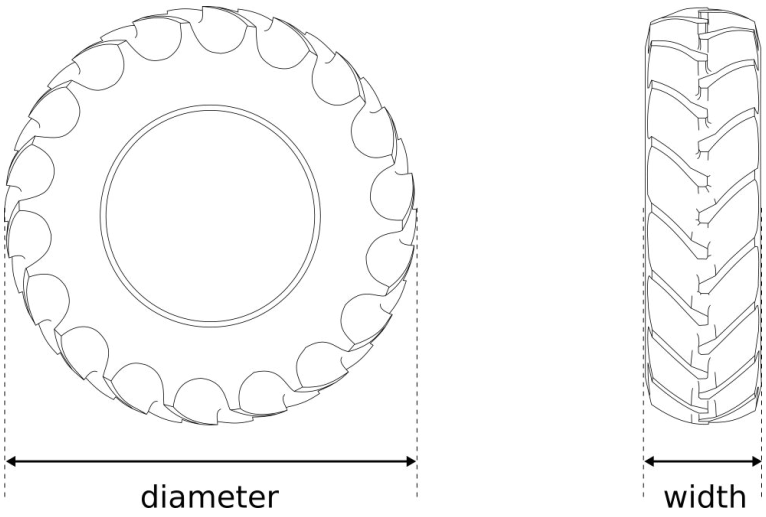


Figure 5.1 – Tires naming conventions (side view)

Figure 5.2 – Container naming conventions (top view)

5.2.2 Shapes

In this problem, for handling and safety reasons, tires cannot be freely located. Their layout must correspond to one of two particular shapes: stacks and individual tires.

Stack a stack is a group of tires, stacked on top of each other (see Fig. 5.3). Four rules must be respected when building a stack:

- tires are ordered from the largest diameter (bottom) to the smallest diameter (top). Ties are broken by comparing widths: the wider tire go below.
- the difference in diameter between two consecutive tires should be less than or equal to a parameter Δ_{Succ}. This is to prevent damages to sidewalls of tires. As a side effect it also avoids having a tire falling inside another one. Indeed, unlike in the packing of pipes [39], nesting tires inside one another is forbidden.
- the difference in diameter between the tire at the bottom and the tire at the top of the stack should be less than or equal to a parameter Δ_{Stack}.
- the height of the stack should be less than or equal to $H_z - \Delta_z$ where Δ_z is a parameter. This is a clearance required to load/unload stacks

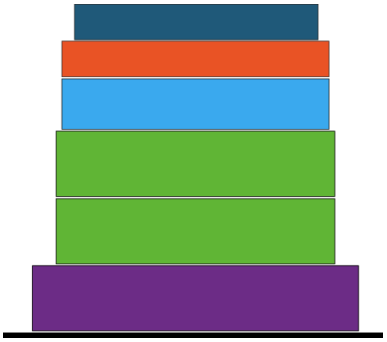


Figure 5.3 – A stack (side view)

Individual tire an individual tire is a tire which is loaded vertically (it lies on its width). We refer to it as a *side tire* when its diameter is parallel to the x-axis and as a *tail ender* when its diameter is parallel to the y-axis (the name comes from the fact that they can only be located near the door, see Section 5.2.3). Only a tire whose width is equal to or greater than a certain parameter W_{TE} can be a tail ender.

5.2.3 Positioning rules

Besides the rules that define shapes, there exists a set of positioning rules that define the position of shapes. We distinguish three kinds of rules: *two contact points*, *alternate* and *no escape*. A valid loading plan is depicted on Fig. 5.4.

Two contact points

This rule is an horizontal stability rule. Each shape is located by resting against two contact points that can be other shapes or walls. However some special conditions must be added depending on the shape.

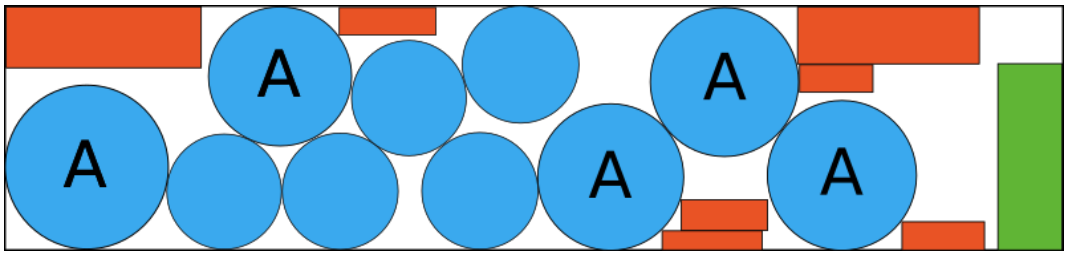


Figure 5.4 – A valid loading plan. Shapes in blue are stacks, those marked A should be alternated. Shapes in orange are side tires. And the shape in green is a tail ender. (top view)

For stacks: First, as containers are rear loaded and as stacks are loaded using clamp lift trucks, for a stack s whose center coordinates are (x, y), the contact points $c_1 = (x_1, y_1)$ and $c2 = (x_2, y_2)$, should be such that $x_1 \leq x$ and $x_2 \leq x$. Second a stack cannot rest against a vertical tire.

For side tires: Side tires should rest laterally against a side wall or another side tire (they cannot be in-between two stacks).

For tail enders: Tail enders should rest against the rear door or another tail ender and a side wall.

Alternate stacks

Stacks whose base diameters are greater than or equal to a parameter W_A should rest against sidewalls. Obviously these stacks are composed of the largest tires and, as such, are the heaviest. As a simple weight distribution constraint, they should rest alternately against the sidewall at $y = 0$ and the sidewall at $y = H_y$, as to be in a *zigzag* formation. We refer to this stacks as *alternated*, and this situation is depicted on Fig. 5.4.

No escape

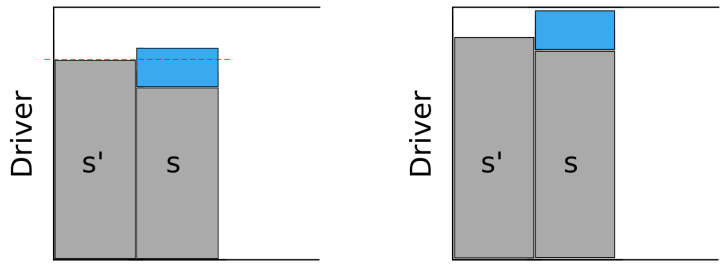
This rule exists to prevent tires on top of stacks from falling towards the driver when braking. When a stack s is resting against a stack s', then one of the following statement should be true:

- s' is high enough to cover at least 50% of the top tire of s,
- the gap between the top of stack s' and the ceiling of the container is smaller than the width of the tire on top of s.

The two cases are depicted on Fig. 5.5.

5.3 Solution methodology

From the description of the problem, we can observe that stacks play a very important role in loading plans. As such, instead of locating each tire individually, it may be interesting to group them to form *shapes* and then locate these shapes in the container. To that end, we propose a two phase algorithm. In the first phase, we enumerate some possible shapes



(a) s' covers more than 50% of the top tire of s

(b) The gap between the top of s' and the ceiling of the container is small enough to prevent the top tire of s from falling

Figure 5.5 – No escape rule valid configurations (side view)

and select a subset of them as to cover all the tires in the shipment. In a second phase, we locate the selected shapes in the container. This is done with a dedicated dynamic programming algorithm.

5.3.1 Shape selection algorithm

The shape selection algorithm aims at selecting a list of shapes (a shape can be selected more than once) that exactly covers the tires in the shipment. Although the problem at stake is a satisfaction problem, when two feasible loading plans exist for the same shipment, the one with less individual tires is preferred, as stacks are easier to manipulate. Building on that observation, in the shape selection phase we favor stacks over individual tires.

We start by enumerating feasible stacks (according to the rules defined in Section 5.2.2) that are higher than a parameter α (thus avoiding short stacks which are unlikely to be part of the loading plan). Experimentally α has been set to $0.75 \times (H_z - \Delta_z)$. We put all these possible stacks in a list C, to which we add all the tires types (to represent individual tires). We aim to select a subset of C that covers exactly T. The volume of tires to locate is constant, therefore, intuitively, a subset of shapes whose surface is small makes a better use of the volume of the container. As such, to select shapes, we minimize the surface covered by the set of selected shapes. For each shape $c \in C$, let a_c be its surface (surface of the base tire for stacks, and $diameter \times width$ for individual tires), and let x_c be the number of time it is selected, and for each tire $t \in T$, let $\lambda_{t,c}$ be a constant that indicates how many times tire t is in c. The shape selection problem can be formulated as a set partitioning problem:

$$\min \sum_{c \in C} a_c * n_c \tag{5.1}$$

$$\sum_{c \in C} \lambda_{c,t} \times n_c = q_t \qquad \forall t \in T \tag{5.2}$$

$$x_c \in \mathbb{N} \qquad \forall c \in C \tag{5.3}$$

The objective is to minimize the covered surface (Equation 5.1), while exactly covering each type of tire in the shipment (Equation 5.2). We choose not to consider a set covering

problem (relaxing Equation (5.2) to greater than or equal), because of the no escape rule. Indeed removing tires in excess could lead to violation of this rule, without any simple way to repair a solution.

The set partitioning problem is solved with a MIP solver. To reduce computation time, we provide it with two initial solutions. These solutions are build using list heuristics on C, such that individual tires are at the end of the list and thus less likely to be selected. We now detail how we sort stacks. In the first heuristic, called *largest first*, stacks in C are sorted such that stacks that contain larger tires come first. In the second heuristics, called *coherence*, stacks in C are sorted in non-increasing order according to the following measure:

$$coherence(c) = \beta \times \frac{\sum_{t \in T} (\lambda_{c,t} \times \frac{d_t}{d_{base}})}{\sum_{t \in T} \lambda_{c,t}} + \gamma \times \frac{\sum_{t \in T} \lambda_{c,t} \times w_t}{H_z}$$

where d_{base} represents the diameter of the tire at the bottom of the stack. This measure can be interpreted as a way to avoid selecting *pyramidal* stacks (first term) and short stacks (second term) that make poor use of the volume. Experimentally β and γ have been set to 0.4 and 0.6 respectively.

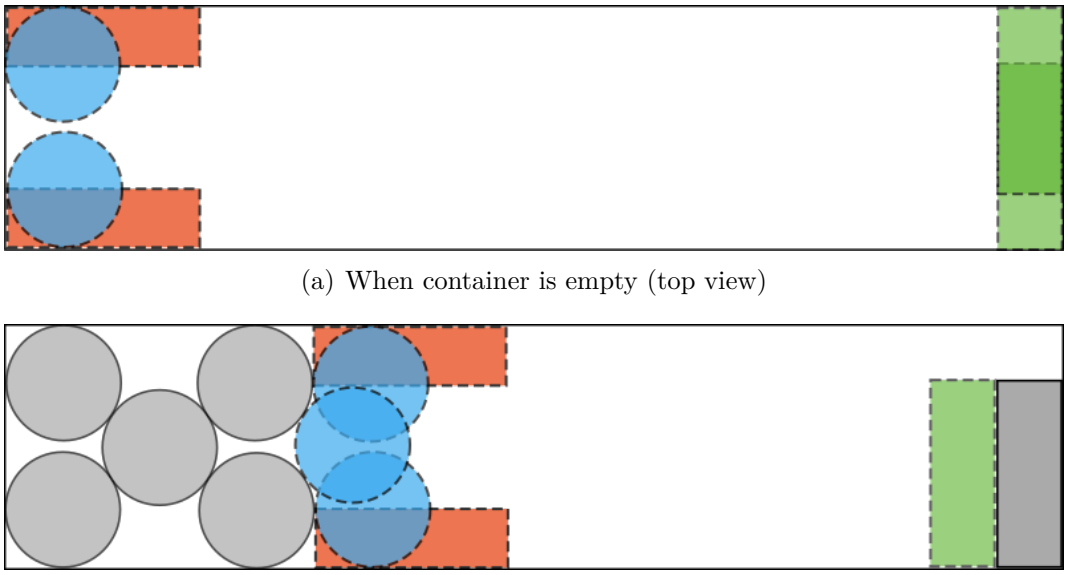
5.3.2 Location algorithm

Given a list of shapes $s \in L$, each associated with a quantity n_s, the location algorithm aims at positioning them in the container while respecting rules defined in Section 5.2.3. Because of the two contact points rule, it can be solved with a dynamic programming (DP) algorithm that sequentially determines the position of shapes by resting them against previously located shapes and/or walls.
In what follows we present (1) the general DP algorithm, (2) an incremental extension of it, designed to reduce runtime, (3) a memory component designed to avoid solving several times the same sub-problem and (4) the branching strategy used.

Dynamic programming algorithm

When the container is empty, stacks and side tires could be located: in the bottom left corner, or in the upper left corner. Tail enders could be located: in the bottom right corner or in the upper right corner. When some shapes have already been located in the container, the remaining shapes can be located, applying the two contact points rule, by resting against previously located shapes and/or walls. Both situations are depicted on Fig. 5.6

Let a shape-position p be the combination of: (1) a shape s_p, (2) a couple of coordinates (x_p, y_p), and (3) a rotation (for individual tires only: to model side tires and tail enders). The previous two situations can be seen as states in a dynamic programming algorithm. Each node n in the DP tree is fully described by: (1) a list of already fixed shape-positions F^n, (2) a list of shapes that have not yet been located U^n, such that $L = (\cup_{p \in F^n} s_p) \cup U^n$. A shape-position is said to be *valid* at a node n if fixing it would lead to a partial solution that respects all rules. All extensions of a node n can be obtained by generating all the valid shape-positions for all shapes in U^n and for all pairs of contact points (in F^n or walls). The coordinates of such shape-positions can be obtained by simple 2D geometry rules. Let V^n be the list of valid shape-positions at a node n. The recursive DP algorithm is presented in Algorithm 8. Algorithm 9 presents a straightforward version of how to generate all valid shape-positions at given node.



(a) When container is empty (top view)

(b) When some shapes have already been fixed (top view)

Figure 5.6 – Possible location of non yet located shapes, represented with dotted lines

Input : a list of fixed shape-positions F^n,
a list of shapes for which a position has yet to been determined U^n
if U^n *is empty* **then**
  A valid loading plan has been found
  End of DP algorithm
end
`Generate shape-positions`
while V^n *is not empty* **do**
  p:= select a shape-position in V^n
  Remove p from V^n
  Call Location($F \cup \{p\}$, $U \backslash \{s_p\}$)
end
Backtrack

Algorithm 8 : `Location` recursive part of the DP for a node n

Incremental dynamic programming algorithm

For a node n in the DP tree, let l^n be the latest fixed shape-position, and let $\delta(n)$ be its parent node. Generating possible extensions for n by nested for-loops as in Algorithm 9 does not make use of information we already know from $\delta(n)$. Indeed, many shape-positions at n are invalid because they overlap with a fixed shape-position that is not l^n. This means that we also generated them and rejected them at $\delta(n)$. Moreover, some valid shape-positions at n are obtained by resting a shape s against two contact points, none of which is l^n. This means that these shape-positions are also extension of $\delta(n)$. Hence, for a node, we save runtime by generating shape-positions incrementally by taking into account shape-positions at its parent node.
Nevertheless, as depicted on Fig. 5.7, a shape-position that is rejected at a node, because it violates the alternate rule, may be feasible at one of its child node. In an incremental DP algorithm, we thus need to identify this situation. To that end, we introduce the three

```
1  V^n := ∅
2  for shape s ∈ U^n do
3  |  for possible orientations of s do
4  |  |  for pair of contacts points (p_1, p_2) ∈ F^n ∪ walls do
5  |  |  |  Let s_p be the corresponding shape-position
6  |  |  |  Compute coordinates (x_p, y_p) resting against p_1 and p_2
7  |  |  |  if s_p does not overlap and respect positioning rules then
8  |  |  |  |  Add s_p to V^n
9  |  |  |  end
10 |  |  end
11 |  end
12 end
```

Algorithm 9 : `Generate shape-positions` for a node n (basic version)

following definitions. At a node n in the DP tree, we say that a shape-position s, for a shape $s_p \in U^n$, is :

- *valid* if fixing it would not violate any rule (as defined in Section 5.3.2)
- *not alternated* if fixing it would respect all rules but the alternate
- *invalid* in all other situations

Using these definitions, in Table 5.1 we present the possible changes in the status of a shape-position from a parent node to one its child nodes.

Status at parent $\delta(n)$	Possible status at node n
valid	• no longer needed ($s_p \notin U^n$) • invalid (overlaps with l^n) • not alternated (l^n makes it violate the alternate rule) • valid
not alternated	• no longer needed ($s_p \notin U^n$) • invalid (overlaps with l^n) • not alternated • valid (l^n makes it feasible w.r.t. the alternate rule)
invalid	• invalid

Table 5.1 – Possible status at node n for a shape-position p that was already available at parent node $\delta(n)$

For a node n, let V^n be the list of valid shape-positions, and let N^n be the list of not alternated shape-positions. We propose an incremental version of the DP algorithm by replacing Algorithm 9 with Algorithm 10. Lines (3-14) correspond to the new shape-positions that could be generated with l^n as contact point, while lines (15-22) correspond to the shape-positions that are carried over from $\delta(n)$.

Memory component and symmetry breaking

As depicted on Fig. 5.8, with small diameter tires, the same DP state can be achieved through different sequences of decisions. To prevent trying to solve twice or more the same

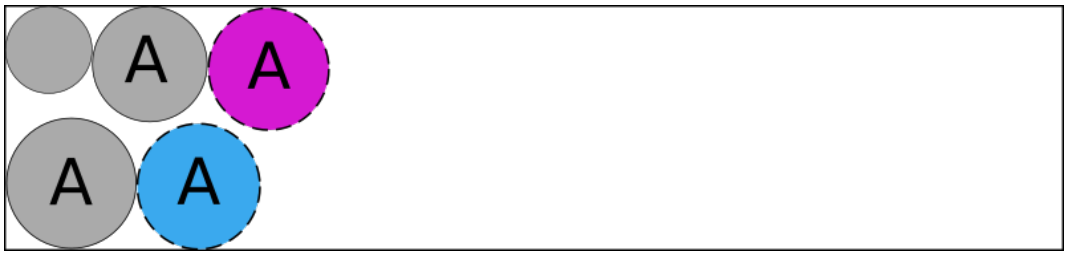


Figure 5.7 – A node in the DP tree. Shape-positions in grey have been fixed. The stack in blue is a valid shape-position at this node, but not the stack in purple as it violates the alternate rule. However, if we fix the stack in blue, then the stack in purple becomes valid in this child node. (top view)

subproblem, a memory component that detects if a state has already been visited is used. Preliminary experiments showed that, a central memory component, that would store all the visited states, was quickly taking too much memory. Instead we propose to create a memory per node, that store all its grand-children states. Before trying to extend a node (line 5 in Algorithm 8), we first check in the memory associated with its grand-parent if a similar state has already been explored. If so, we backtrack immediately otherwise we continue as in Algorithm 8. This can cover the case depicted on Fig. 5.8, while keeping a low memory footprint.

Figure 5.8 – Example of a situation where a memory component is needed: fixing (1) the shape-position in blue and (2) the shape-position in purple, or (1) the shape in position in purple and (2) the shape-position in blue lead to the same state. (top view)

Branching strategy

Regarding branching strategy (l.8 in Algorithm 8), in preliminary experiments, we observed that locating side tires early in the DP tree does lead to many early uninformative failures, since stacks cannot rest against side tires. We thus choose a branching strategy that locate stacks first. Shape-positions associated with stacks are sorted in the following lexicographic order: (1) non increasing base diameter, (2) increasing height, (3) non-increasing x-coordinates (4) non increasing y-coordinates. On top of that, we impose a symmetry breaking rule that enforce the first stacks to rest against the wall at $y = 0$.

5.4 Computational experiments

Currently, loading plans are created by hand. This can take a worker up to 45 minutes. As such, one of the goal for the proposed method is to come up with loading plan much faster. Another important aspect is to make sure that all safety rules are respected, as it

$V^n := \emptyset$
$N^n := \emptyset$
for *shape* $s \in U^n$ **do**
  for *a contact point* $p_1 \in (F^n \setminus \{l^n\}) \cup walls$ **do**
    Let p be the corresponding shape-position
    Compute coordinates (x_p, y_p) resting against l^n and p_1
    if p *is valid* **then**
      Add p to V^n
    end
    if p *is not alternated* **then**
      Add p to N^n
    end
  end
end
for *shape positions* $p \in V^{\delta(n)} \cup N^{\delta(n)}$ **do**
  if p *is valid at* n **then**
    Add p to V^n
  end
  if p *is not alternated at* n **then**
    Add p to N^n
  end
end

Algorithm 10 : `Generate shape-positions` for a node n (incremental version)

may happen in manual loading plans that some rules are violated. In what follows, we describe the test instances and present the results.

5.4.1 Instances

To test the proposed method, the company provided us with three different data sets corresponding to past shipments:

Set A: a set of shipments for which valid loading plans exist. They correspond to what is done on a regular basis. It consists of 7 instances.

Set B: a set of shipments for which very carefully optimized loading plans exist. These loading plans were created by experienced workers that spent a lot of time to find them. This set is meant to be more challenging than Set A. It consists of 3 instances.

Set C: a set of shipments for which manual loading plans that were executed did not respect all the positioning rules. As a consequence, we have no guarantee on whether valid loading plans for these shipments exist. This set is meant to test the impact of the strict enforcement of positioning rules on loading plans. It consists of 8 instances.

Instances contain between 7 and 40 different tire types, with an average of 22.9, and between 41 and 162 tires with an average of 96.2 . Containers are almost always the same. This illustrates that any algorithm designed to solve this problem should be really flexible to solve rather heterogeneous instances.

5.4.2 Results and observations

The algorithm is coded in C++ and uses CPLEX from IBM ILOG CPLEX Optimization Studio 12.6.1 as MIP solver in the shape selection phase. The experiments were conducted under Linux using an Intel Xeon X7350 @ 2.93 GHz. Only one core is used both by our code and CPLEX.
Within 4 minutes of runtime, we were able to find valid loading plans: for all instances in set A, 1 out of 3 in set B and for 4 out of 8 instances in set C. When a solution is found, it is composed of 17.6 shapes on average, 81 % of which are stacks. In set A, loading plans found by the proposed method tend to take less surface than what was found by hand. When carefully examining manual loading plans for set B, we observe that they were making use situations similar to that of Fig. 5.9 in presence of large tires. The proposed method cannot find such loading plans because of its two phase structure. Results for set C show the added safety associated with the use of an optimization software over manual optimization. For both set B and set C instances, when no solution was found after 4 minutes, no solution was found for longer runtimes either. This illustrates the efficiency of our branching strategy and the need for efficient bounds. From these results, we can conclude that the proposed method shows good performance and can significantly reduce the time spent on finding loading plans. It is considered satisfactory by the company.

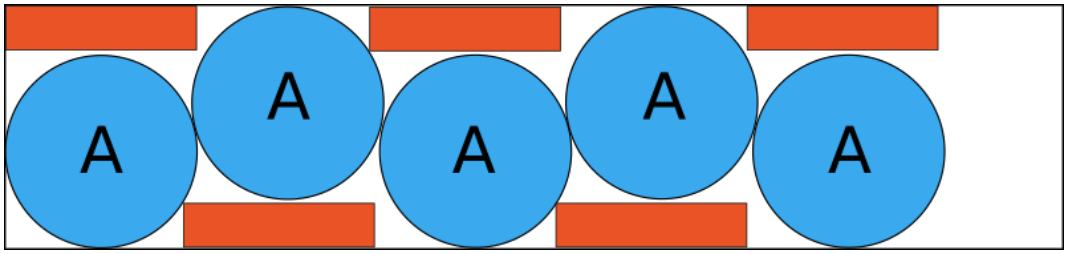


Figure 5.9 – Example of situation where, when minimizing the surface of the selected shapes in the first phase, side tires in orange would be grouped together to create a stack. However, in this configuration, intervals between alternated stacks and opposite side walls would remain empty and no loading plan could be found (top view)

Conclusion

In this article, we present a real-life 3D container loading problem of agricultural tires occurring in one of the leading tire manufacturers. It has an unique set of characteristics due to the tubular shape of tires and many positioning rules that define the problem. We have proposed a two phase method to solve this problem, that first creates structures with tires and then locates them with a dedicated dynamic programming algorithm. Results highlight the good performance of the overall method, although some challenging instances could not be solved. To overcome this issue, a method that would both create structures and locate them at the same time may be a direction for future research. This problem is part of a large project, and methods described in this article have been integrated to the global solution.

Table des matières

Liste des tableaux

Table des figures

Bibliographie

[1] Dwi Agustina, C. K. M. Lee, and Rajesh Piplani. A review : mathematical models for cross docking planning. *International Journal of Engineering Business Management*, 2(2) :47–54, 2010. 36, 55

[2] Claudia Archetti and Maria Grazia Speranza. Vehicle routing problems with split deliveries. *International Transactions in Operational Research*, 19(1-2) :3–22, 2012. 4

[3] Claudia Archetti and Maria Grazia Speranza. A survey on matheuristics for routing problem. *EURO Journal on Computational Optimization*, 2 :223–246, 2014. 7, 37

[4] Claudia Archetti, Maria Grazia Speranza, and Martin W. P. Savelsbergh. An Optimization-Based Heuristic for the Split Delivery Vehicle Routing Problem. *Transportation Science*, 42(1) :22–31, 2008. 37

[5] Lasse Asbach, Ulrich Dorndorf, and Erwin Pesch. Analysis, modeling and solution of the concrete delivery problem. *European Journal of Operational Research*, 193(3) :820–835, 2009. 55

[6] Nabila Azi, Michel Gendreau, and Jean-Yves Potvin. An adaptive large neighborhood search for a vehicle routing problem with multiple routes. *Computers & Operations Research*, 41 :167–173, jan 2014. 6, 15, 19, 20

[7] J. Christopher Beck and Mark S Fox. Scheduling Alternative Activities. *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence*, pages 680–687, 1999. 44, 61

[8] Robert E. Bixby. A Brief History of Linear and Mixed-Integer Programming Computation. *Documenta Mathematica*, I(Extra) :107–121, 2012. 7

[9] Andreas Bortfeldt, Thomas Hahn, Dirk Männel, and Lars Mönch. Hybrid algorithms for the vehicle routing problem with clustered backhauls and 3D loading constraints. *European Journal of Operational Research*, 243(1) :82–96, 2015. 19

[10] Andreas Bortfeldt and Gerhard Wäscher. Constraints in container loading – A state-of-the-art review. *European Journal of Operational Research*, 229(1) :1–20, aug 2013. 76

[11] Nils Boysen and Malte Fliedner. Cross dock scheduling : classification, literature review and research agenda. *Omega*, 38(6) :413–422, 2010. 36, 55

[12] David Bredström and Mikael Rönnqvist. Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. *European Journal of Operational Research*, 191(1) :19–29, 2008. 5

[13] Ulrich Breunig, Verena Schmid, Richard F. Hartl, and Thibaut Vidal. A fast large neighbourhood based heuristic for the Two-Echelon Vehicle Routing Problem. 2015. 15

[14] Paul Buijs, Iris F.A. Vis, and Héctor J. Carlo. Synchronization in cross-docking networks : A research classification and framework. *European Journal of Operational Research*, 2014. 10, 11, 37, 54

[15] Diego Cattaruzza, Nabil Absi, Dominique Feillet, and Jesus Gonzalez-Feliu. Vehicle routing problems for city logistics. *EURO Journal on Transportation and Logistics*, 2015. 4, 14, 15

[16] Diego Cattaruzza, Nabil Absi, Dominique Feillet, and Daniele Vigo. An iterated local search for the multi-commodity multi-trip vehicle routing problem with time windows. *Computers & Operations Research*, 51 :257–267, 2014. 15

[17] Thomas H. Cormen, Charles E. Leiserson, Rivest L. Ronald, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2010. 24, 26

[18] Cristián E. Cortés, Martín Matamala, and Claudio Contardo. The pickup and delivery problem with transfers : Formulation and a branch-and-cut solution method. *European Journal of Operational Research*, 200(3) :711–724, feb 2010. 4, 5, 37, 55

[19] Jean-françois Côté, Michel Gendreau, and Jean-Yves Potvin. Large Neighborhood Search for the Single Vehicle Pickup and Delivery Problem with Multiple Loading Stacks. *Networks*, 60 :19–30, 2012. 6

[20] Teodor Gabriel Crainic, Simona Mancini, Guido Perboli, and Roberto Tadei. Clustering-based heuristics for the two-echelon vehicle routing problem. Technical Report 2008-46, CIRRELT, 2008. 15

[21] Teodor Gabriel Crainic, Simona Mancini, Guido Perboli, and Roberto Tadei. Multi-start heuristics for the two-echelon vehicle routing problem. In Peter Merz and Jin-Kao Hao, editors, *Evolutionary Computation in Combinatorial Optimization*, pages 179–190. Springer, 2011. 15

[22] Teodor Gabriel Crainic, Simona Mancini, Guido Perboli, and Roberto Tadei. GRASP with path relinking for the two-echelon vehicle routing problem. *Advances in Metaheuristics*, 53 :113–125, 2013. 15

[23] Teodor Gabriel Crainic, Guido Perboli, Simona Mancini, and Roberto Tadei. Two-echelon vehicle routing problem : a satellite location analysis. *Procedia - Social and Behavioral Sciences*, 2(3) :5944–5955, 2010. 15, 28

[24] Teodor Gabriel Crainic, Nicoletta Ricciardi, and Giovanni Storchi. Advanced freight transportation systems for congested urban areas. *Transportation Research Part C : Emerging Technologies*, 12(2) :119–137, apr 2004. 14

[25] Teodor Gabriel Crainic, Nicoletta Ricciardi, and Giovanni Storchi. Models for evaluating and planning city logistics systems. *Transportation Science*, 43(4) :432–454, 2009. 8, 14, 15, 16

[26] Rosario Cuda, Gianfranco Guastaroba, and Maria Grazia Speranza. A survey on two-echelon routing problems. *Computers & Operations Research*, pages 1–15, jun 2014. 8, 15

[27] Emrah Demir, Tolga Bektaş, and Gilbert Laporte. An adaptive large neighborhood search heuristic for the Pollution-Routing Problem. *European Journal of Operational Research*, 223(2) :346–359, 2012. 6

[28] Karl F. Doerner and Verena Schmid. Survey : Matheuristics for rich vehicle routing problems. In *Hybrid Metaheuristics*, volume 6373 of *LNCS*, pages 206–221. Springer, 2010. 37

[29] Rodolfo Dondo and Jaime Cerdá. A monolithic approach to vehicle routing and operations scheduling of a cross-dock system with multiple dock doors. *Computers & Chemical Engineering*, 63 :184–205, apr 2014. 37, 54

[30] Michael Drexl. *On Some Generalized Routing Problems*. PhD thesis, Rheinisch-Westfälischen Technischen Hochschule Aachen, 2007. 4

[31] Michael Drexl. Synchronization in vehicle routing–A survey of VRPs with multiple synchronization constraints. *Transportation Science*, 46(3) :297–316, 2012. 3, 4, 11, 15, 24, 37, 54, 55

[32] M.J.R. Ebben, M.C. van der Heijden, and A. van Harten. Dynamic transport scheduling under multiple resource constraints. *European Journal of Operational Research*, 167(2) :320–335, 2005. 10, 55

[33] Jens Egeblad, Claudio Garavelli, Stefano Lisi, and David Pisinger. Heuristics for container loading of furniture. *European Journal of Operational Research*, 200(3) :881–892, 2010. 76

[34] Nizar El Hachemi, Michel Gendreau, and Louis-Martin Rousseau. A hybrid constraint programming approach to the log-truck scheduling problem. *Annals of Operations Research*, 184(1) :163–178, feb 2010. 55

[35] Nizar El Hachemi, Michel Gendreau, and Louis-Martin Rousseau. A heuristic to solve the synchronized log-truck scheduling problem. *Computers and Operations Research*, 40(3) :666–673, 2013. 10, 55

[36] Furkan Enderer. *Integrating dock-door assignment and vehicle routing in cross-docking*. PhD thesis, Concordia University, 2014. 37

[37] Alan L. Erera, Michael Hewitt, Martin W. P. Savelsbergh, and Yang Zhang. Creating schedules and computing operating costs for LTL load plans. *Computers & Operations Research*, 40(3) :691–702, mar 2013. 11

[38] B. A. Foster and D. M. Ryan. An Integer Programming Approach to the Vehicle Scheduling Problem. *Operational Research Quarterly*, 27(2) :367–384, 1976. 7

[39] John a. George, Jennifer M. George, and Bruce W. Lamar. Packing different-sized circles into a rectangular container. *European Journal of Operational Research*, 84(95) :693–712, 1995. 78

[40] Philippe Grangier, Michel Gendreau, Fabien Lehuédé, and Louis-Martin Rousseau. An adaptive large neighborhood search for the two-echelon multiple-trip vehicle routing problem with satellite synchronization. Technical report, CIRRELT 2014-33, 2014. 37, 41, 42

[41] Philippe Grangier, Michel Gendreau, Fabien Lehuédé, and Louis-Martin Rousseau. A large neighborhood search based matheuristic search for the vehicle routing problem with cross-docking. 2015. 54, 58, 60, 62, 65, 67, 68, 69, 70, 71, 72, 73, 74, 92, 93

[42] Axel Grimault, Nathalie Bostel, and Fabien Lehuédé. An Adaptive Large Neighborhood Search for the Full Truckload Pickup and Delivery Problem with Resource Synchronization. Technical report, Ecole des Mines de Nantes 15/4/AUTO, 2015. 42, 55

[43] Axel Grimault, Fabien Lehuédé, and Nathalie Bostel. A two-phase heuristic for full truckload routing and scheduling with split delivery and resource synchronization in public works. *2014 International Conference on Logistics Operations Management*, pages 57–61, 2014. 55

[44] Chris Groër, Bruce Golden, and Edward Wasil. A parallel algorithm for the vehicle routing problem. *INFORMS Journal on Computing*, 23(2) :315–330, 2011. 7

[45] Roar Gronhaug, Marielle Christiansen, Guy Desaulniers, and Jacques Desrosiers. A Branch-and-Price Method for a Liquefied Natural Gas Inventory Routing Problem. *Transportation Science*, 44(3) :400–415, 2010. 55

[46] Gianfranco Guastaroba, Maria Grazia Speranza, and Daniele Vigo. Intermediate Facilities in Freight Transportation Planning : A Survey. Technical Report 1, 2015. 55

[47] Vera C. Hemmelmayr, Jean-François Cordeau, and Teodor Gabriel Crainic. An adaptive large neighborhood search heuristic for two-echelon vehicle routing problems arising in city logistics. *Computers & Operations Research*, 39(12) :3215–3228, 2012. 15, 19

[48] Christoph Hempsch and Stefan Irnich. Vehicle routing problems with inter-tour resource constraints. In Bruce Golden, S. Raghavan, and Edward Wasil, editors, *The Vehicle Routing Problem : Latest Advances and New Challenges*, volume 43 of *Operations Research/Computer Science Interfaces*, pages 421–444. Springer US, Boston, MA, 2008. 4, 10, 55, 57

[49] Mhand Hifi. Exact algorithms for unconstrained three-dimensional cutting problems : a comparative study. *Computers & Operations Research*, 31(5) :657–674, 2004. 76

[50] Mhand Hifi and Rym M'Hallah. A Literature Review on Circle and Sphere Packing Problems : Models and Methodologies. *Advances in Operations Research*, 2009 :1–22, 2009. 76

[51] IBM Corporation. IBM ILOG CPLEX Optimization Studio V12.6.1 documentation, 2014. 44, 61

[52] Irina Ioachim, Jacques Desrosiers, François Soumis, and Nicolas Bélanger. Fleet assignment and routing with schedule synchronization constraints. *European Journal of Operational Research*, 119(1) :75–90, 1999. 5

[53] Mads Jepsen, Simon Spoorendonk, and Stefan Ropke. A branch-and-cut algorithm for the symmetric two-echelon capacitated vehicle routing problem. *Transportation Science*, 47(1) :23–37, 2013. 15

[54] Attila a. Kovacs, Sophie N. Parragh, Karl F. Doerner, and Richard F. Hartl. Adaptive large neighborhood search for service technician routing and scheduling problems. *Journal of Scheduling*, 15(5) :579–600, aug 2011. 19

[55] Anne-Laure Ladier and Gülgün Alpan. Cross-docking operations : Current research versus industry practice. *Omega*, 2015. 54

[56] Gilbert Laporte, Stefan Ropke, and Thibaut Vidal. Heuristics for the vehicle routing problem. In Paolo Toth and Daniele Vigo, editors, *Vehicle Routing Problems : Problems, Methods, and Applications*, chapter 4, pages 87–116. MOS-SIAM, second edition, 2014. 51

[57] Young Hae Lee, Jung Woo Jung, and Kyong Min Lee. Vehicle routing scheduling for cross-docking in the supply chain. *Computers & Industrial Engineering*, 51(2) :247–256, 2006. 36, 54

[58] Y Li, A Lim, and B Rodrigues. Crossdocking - JIT scheduling with time windows. *Journal of the Operational Research Society*, 55 :1342–1351, 2004. 57

[59] Ching-Jong Liao, Yaoming Lin, and Stephen C. Shih. Vehicle routing with cross-docking in the supply chain. *Expert Systems with Applications*, 37(10) :6868–6873, oct 2010. 36

[60] Simona Mancini. Multi-echelon distribution systems in city logistics. *European Transport/Trasporti Europei*, 54, 2013. 15

[61] Silvano Martello, David Pisinger, and Daniele Vigo. The Three-Dimensional Bin Packing Problem. *Operations Research*, 48(2) :256–267, 2000. 76

[62] Renaud Masson. *Problèmes de collectes et livraisons avec transferts*. PhD thesis, Ecole Centrale de Nantes, 2012. 4

[63] Renaud Masson, Fabien Lehuédé, and Olivier Péton. An adaptive large neighborhood search for the pickup and delivery problem with transfers. *Transportation Science*, 47(3) :344–355, 2013. 6, 15, 19, 20, 37, 42, 46

[64] Renaud Masson, Fabien Lehuédé, and Olivier Péton. Efficient feasibility testing for request insertion in the Pickup and Delivery Problem with Transfers. *Operations Research Letters*, 41(3) :211–215, 2013. 5, 9, 15, 23, 25, 26, 42, 60

[65] Renaud Masson, Fabien Lehuédé, and Olivier Péton. The dial-a-ride problem with transfers. *Computers and Operations Research*, 41(1) :12–23, 2014. 19

[66] Renaud Masson, Anna Trentini, Fabien Lehuédé, Nicolas Malhéné, Olivier Péton, and Houda Tlahig. Optimization of a city logistics transportation system with mixed passengers and goods. *EURO Journal on Transportation and Logistics*, pages 1–29, 2015. 15

[67] Jorge Mendoza, Louis-Martin Rousseau, and Juan G. Villegas. A hybrid meatheuristic for the vehicle routing problem with stochastic demand and duration constraint. *Journal of Heuristics*, 2015. 37

[68] Jorge Mendoza and Juan G. Villegas. A multi-space sampling heuristic for the vehicle routing problem with stochastic demands. *Optimization Letters*, 7(7) :1503–1516, 2013. 7

[69] Vinicius W.C. Morais, Geraldo Robson Mateus, and Thiago F. Noronha. Iterated local search heuristics for the Vehicle Routing Problem with Cross-Docking. *Expert Systems with Applications*, 41(16) :7495–7506, nov 2014. 9, 37, 42, 46, 49, 51, 54

[70] Phuong Khanh Nguyen, Teodor Gabriel Crainic, and Michel Toulouse. A tabu search for Time-dependent Multi-zone Multi-trip Vehicle Routing Problem with Time Windows. *European Journal of Operational Research*, 231(1) :43–56, nov 2013. 15

[71] Viet-Phuong Nguyen, Christian Prins, and Caroline Prodhon. Solving the two-echelon location routing problem by a GRASP reinforced by a learning process and path relinking. *European Journal of Operational Research*, 216(1) :113–126, 2012. 15

[72] Sophie N. Parragh and Verena Schmid. Hybrid column generation and large neighborhood search for the dial-a-ride problem. *Computers and Operations Research*, 40(1) :490–497, 2013. 7, 39

[73] Guido Perboli and Roberto Tadei. New families of valid inequalities for the two-echelon vehicle routing problem. *Electronic Notes in Discrete Mathematics*, 36 :639–646, 2010. 15

[74] Guido Perboli, Roberto Tadei, and Daniele Vigo. The two-echelon capacitated vehicle routing problem : models and math-based heuristics. *Transportation Science*, 45(3) :364–380, 2011. 4, 15, 37

[75] Hanne L Petersen and Stefan Ropke. The pickup and delivery problem with cross-docking opportunity. In *Computational Logistics*, pages 101–113. Springer, 2011. 10, 37, 54

[76] Victor Pillac, Christelle Guéret, and Andrés Medaglia. A parallel matheuristic for the technician routing and scheduling problem. *Optimization Letters*, 7(7) :1525–1535, 2013. 37

[77] David Pisinger and Stefan Ropke. A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8) :2403–2435, 2007. 6, 19, 39, 40, 59

[78] Eric Prescott-Gagnon, Guy Desaulniers, and Louis-Martin Rousseau. A branch-and-price-based large neighborhood search algorithm for the vehicle routing problem with time windows. *Networks*, 54(4) :190–204, 2009. 6

[79] Yuan Qu and Jonathan F. Bard. A GRASP with adaptive large neighborhood search for pickup and delivery problems with transshipment. *Computers & Operations Research*, 39(10) :2439–2456, 2012. 15, 37

[80] Roberto Roberti. *Exact algorithms for different classes of vehicle routing problems*. PhD thesis, Bologna, jun 2012. 15

[81] Yves Rochat and Éric D. Taillard. Probability Diversification and Intensification in Local Search for Vehicle Routing. *Journal of Heuristics*, 1 :147–167, 1995. 7, 37

[82] Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4) :455–472, 2006. 5, 6, 19, 20, 29, 39, 40, 46, 59

[83] Fernando Afonso Santos, Alexandre Salles Cunha, and Geraldo Robson Mateus. Branch-and-price algorithms for the two-echelon capacitated vehicle routing problem. *Optimization Letters*, 7(7) :1537–1547, oct 2012. 15

[84] Fernando Afonso Santos, Geraldo Robson Mateus, and Alexandre Salles da Cunha. A novel column generation algorithm for the vehicle routing problem with cross-docking. In *Network Optimization - INOC 2011*, volume 6701 of *LNCS*, pages 412–425, 2011. 37, 54

[85] Fernando Afonso Santos, Geraldo Robson Mateus, and Alexandre Salles da Cunha. The pickup and delivery problem with cross-docking. *Computers & Operations Research*, 40(4) :1085–1093, 2013. 37, 54

[86] Fernando Afonso Santos, Geraldo Robson Mateus, and Alexandre Salles da Cunha. A branch-and-cut-and-price algorithm for the two-echelon capacitated vehicle routing problem. *Transportation Science*, Articles i(April) :1–14, 2014. 15

[87] Fernando Afonso Santos, Geraldo Robson Mateus, and Alexandre Salles da Cunha. A branch-and-price algorithm for a vehicle routing problem with cross-docking. *Electronic Notes in Discrete Mathematics*, 37 :249–254, 2011. 37, 54

[88] Martin W. P. Savelsbergh. Local search in routing problems with time windows. *Annals of Operations Research*, 4(1) :285–305, 1985. 5, 23, 42, 60

[89] Verena Schmid, Karl F. Doerner, Richard F. Hartl, and Juan-José Salazar-González. Hybridization of very large neighborhood search for ready-mixed concrete delivery problems. *Computers and Operations Research*, 37(3) :559–574, 2010. 55

[90] Verena Schmid, Karl F. Doerner, Richard F. Hartl, Martin W. P. Savelsbergh, and Wolfgang Stoecher. A Hybrid Solution Approach for Ready-Mixed Concrete Delivery. *Transportation Science*, 43(1) :70–85, 2009. 10, 55

[91] Gerhard Schrimpf, Johannes Schneider, Hermann Stamm-Wilbrandt, and Gunter Dueck. Record Breaking Optimization Results Using the Ruin and Recreate Principle. *Journal of Computational Physics*, 159(2) :139–171, apr 2000. 6

[92] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint Programming — CP98*, volume 1520 of *LNCS*, pages 417–431, 1998. 5, 19, 20, 39, 58

[93] Marius M. Solomon. Algorithms for the vehicle routing and scheduling problem with time window constraints. *Operations Research*, 35(2) :254–265, 1987. 27

[94] George Stalk, Philip Evans, and Lawrence E. Schulman. Competing on capabilities : the new rules of corporate strategy. *Harvard Business Review*, 70(2) :57–69, 1992. 36

[95] Anand Subramanian, Eduardo Uchoa, and Luiz Satoru Ochi. A hybrid algorithm for a class of vehicle routing problems. *Computers and Operations Research*, 40(10) :2519–2531, 2013. 7, 37

[96] Éric D. Taillard, Gilbert Laporte, and Michel Gendreau. Vehicle routing with multiple use of vehicles. *Journal of the Operational Research Society*, 47 :1065–1070, 1996. 15

[97] Christos D. Tarantilis. Adaptive multi-restart tabu search algorithm for the vehicle routing problem with cross-docking. *Optimization Letters*, 7(7) :1583–1596, 2012. 9, 37, 49, 51, 54

[98] Erlendur S. Thorsteinsson. Branch-and-check : A hybrid framework integrating mixed integer programming and constraint logic programming. In Toby Walsh, editor, *Principles and Practice of Constraint Programming — CP 2001*, volume 2239 of *LNCS*, pages 16–30, 2001. 9, 36, 37, 43, 60

[99] Paolo Toth and Daniele Vigo, editors. *Vehicle Routing : Problems, Methods, and Applications*. Society for Industrial and Applied Mathematics, 2014. 6, 7

[100] Jan Van Belle, Paul Valckenaers, and Dirk Cattrysse. Cross-docking : state of the art. *Omega*, 40(6) :827–846, 2012. 10, 36, 53, 55, 57

[101] Pascal Van Hentenryck. *The OPL optimization programming language*. MIT Press, 1999. 44, 61

[102] Juan G. Villegas, Christian Prins, Caroline Prodhon, Andrés Medaglia, and Nubia Velasco. A matheuristic for the truck and trailer routing problem. *European Journal of Operational Research*, 230(2) :231–244, 2013. 37

[103] Gerhard Wäscher, Heike Haußner, and Holger Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3) :1109–1130, 2007. 76

[104] Min Wen, J Larsen, J Clausen, Jean-François Cordeau, and Gilbert Laporte. Vehicle routing with cross-docking. *Journal of the Operational Research Society*, 60(12) :1708–1718, 2008. 9, 10, 36, 37, 38, 46, 49, 54, 56, 67

[105] Zheng-Yang Zeng, W Xu, Zhi-Yu Xu, and Wei-Hui Shao. A Hybrid GRASP+ VND heuristic for the two-echelon vehicle routing problem arising in City Logistics. Technical report, School of Electronics and Information Engineering, Tongji University, 2014. 15

Thèse de Doctorat

Philippe GRANGIER

Résolution de problèmes de tournées avec synchronisation : applications au cas multi-échélons et au cross-docking

Solving vehicle routing problems with synchronization constraints: applications to multi-echelon distribution systems and to cross-docking

Résumé
L'interconnexion croissante dans les systèmes de transports a conduit à la modélisation de nouvelles contraintes, dites contraintes de synchronisation, dans les problèmes de tournées de véhicules. Dans cette thèse, nous nous intéressons à deux cas dans lesquels ce type de problématiques apparaît. Dans un premier temps, nous proposons une méthode heuristique pour un problème à deux échelons rencontré pour la distribution de marchandises en ville. Dans un second temps, nous étudions l'intégration d'un cross-dock dans des tournées de collectes et livraisons. Une première contribution à ce sujet concerne le problème de tournées de véhicules avec cross-docking, et une seconde contribution intègre, en plus, des contraintes de ressources au cross-dock dans le problème de routage. Une méthode pour un problème de chargement 3D, étudié lors d'un stage doctoral en entreprise, est également présentée.

Mots clés
recherche opérationnelle, optimisation, tournées de véhicules, synchronisation, matheuristiques.

Abstract
Transportation systems are more and more interconnected, this has lead to a new kind of constraints, called synchronization constraints, in vehicle routing problems. In this thesis, we study two cases in which this type of constraints arises. First, we propose a heuristic method for a two-echelon problem arising in City Logistics. Second, we study the integration of a cross-dock in pickup and delivery vehicle routing problems. To that end we propose a matheuristic for the vehicule routing problem with cross-docking, and we propose an extension of this problem that integrates specific resource synchonization constraints arising at the cross-dock. A method for a 3D loading problem is also presented.

Key Words
operations research, optimization, vehicle routing problems, synchronization, matheuristics.