FORMULAÇÕES E ALGORITMOS EM PROGRAMAÇÃO INTEIRA PARA O PROBLEMA DO CAIXEIRO VIAJANTE COM COLETA E ENTREGA SOBRE CARREGAMENTO LIFO

AFONSO HENRIQUE SAMPAIO

FORMULAÇÕES E ALGORITMOS EM PROGRAMAÇÃO INTEIRA PARA O PROBLEMA DO CAIXEIRO VIAJANTE COM COLETA E ENTREGA SOBRE CARREGAMENTO LIFO

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

ORIENTADOR: SEBASTIÁN ALBERTO URRUTIA

Belo Horizonte
Maio de 2014

AFONSO HENRIQUE SAMPAIO

FORMULATIONS AND ALGORITHMS IN INTEGER PROGRAMMING FOR THE PICKUP AND DELIVERY TRAVELLING SALESMAN PROBLEM WITH MULTIPLE STACKS

Dissertation presented to the Graduate Program in Computer Science of the Universidade Federal de Minas Gerais. Departamento de Ciência da Computação in partial fulfillment of the requirements for the degree of Master in Computer Science.

ADVISOR: SEBASTIÁN ALBERTO URRUTIA

Belo Horizonte
May 2014

Sampaio, Afonso Henrique

S237r Formulations and algorithms in integer programming for the pickup and delivery travelling salesman problem with multiple stacks / Afonso Henrique Sampaio. — Belo Horizonte, 2014

xxvi, 65 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais. Departamento de Ciência da Computação

Orientador: Sebastián Alberto Urrutia

1. Computação — Teses. 2. Otimização combinatória — Teses. 3. Programação inteira — Teses. 4. Problema do caixeiro viajante. I. Orientador. II. Título.

CDU 519.6*61 (043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Formulações e algoritmos em programação inteira para o problema do caixeiro viajante com coleta e entrega sobre carregamento lifo

AFONSO HENRIQUE SAMPAIO OLIVEIRA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

[illegible]

Prof. Sebastián Alberto Urrutia - Orientador
Departamento de Ciência da Computação - UFMG

[illegible]

Prof. Cid Carvalho de Souza
Instituto de Computação - UNICAMP

[illegible]

Prof. Geraldo Robson Mateus
Departamento de Ciência da Computação - UFMG

Johan Oppen

Prof. Johan Oppen
Molde University College

[illegible]

Prof. Ricardo Saraiva de Camargo
Departamento de Engenharia de Produção - UFMG

Belo Horizonte, 22 de maio de 2014.

Para a minha mãe e minha irmã. E para o meu pai.

Acknowledgments

Firstly, thanks, praise and love to my mother, Clarinda, and my sister, Tainara. Because of these two, I don't remember any moment of worry or fear during this work - I always could count with their support and it's hard to express how hugely I am grateful to them. For all his previous support and great incentive, from the very beginning of my life to the end of his, I would not be in the position of accomplish this work if was not by my father, Afonso — *only some time after a man's death we regard his absence as incomprehensible.*

I am hugely grateful to Sebastián for his tremendous support throughout all the developments of this project. I had the pleasure of work with him before and this time was no exception — he was not only a great supervisor but also a great friend!

For the philosophical discussions which helped endure in the face of failure but rejoice in the eventual success, for sharing some few bad but lots of great moments, for the companionship, my sincerely big thanks to Harlley Lima, Phillipe Samer, Henrique Chevreux, Felipe Repolês and Thiago Cardoso. *amigo — é que a gente seja, mas sem precisar de saber o por quê é que é!*

I had the chance of meet great folks at LaPO, including, but not limited to: Evelyn Cavalcante, Vitor Andrade, Vinícius Morais, Amadeu Almeida, Ramon Lopes, Luis Henrique, Cristopher Moreira, Tiago Januário and all others I forgot.

My grateful thanks to the Master's thesis committee for the feedback, corrections and suggestions, many of then I tried to follow in this final version of the text.

Additionally, I am grateful for the financial support given by CAPES through a scholarship.

"Muita coisa é inata, mas muito é feito pelo treinamento. Por isso, ninguém será bem-sucedido se se poupar, se não mergulhar fundo nos temas maiores e se não estiver em condições de se empenhar até o extremo por causas insignificantes."
(Walter Benjamim, Rua de Mão Única - Obras Escolhidas II.)

Abstract

This dissertation addresses the Pickup and Delivery Travelling Salesman Problem with Multiple Stacks and algorithmic approaches to obtain its exact solution. In this problem, a single vehicle must serve a set of customer requests defined by a pair of pickup and delivery destinations of an item. The vehicle contains a fixed number of stacks where each request is loaded at a pickup location and unloaded at the corresponding delivery location. Each stack has finite capacity, and its loading/unloading sequence must follow the last-in-first-out policy, i.e. for each stack, just the last item loaded can be unloaded at its corresponding delivery location.

We propose a new integer programming formulation for this problem with a polyhedral representation described by exponentially-many inequalities. In particular, we introduce a new set of variables used to model the last-in-first-out policy for loading and unloading items. With the inclusion of these new variables, finding violations concerning the capacity of each stack or the LIFO policy for a given tour can be done by solving polynomial problems. These ideas are used within a branch-and-cut algorithm to solve the proposed formulation.

Computational results show that our approach is competitive with the best algorithm in the literature, outperforming it for some benchmark instances. Also, two new certificates of optimality are provided.

Keywords: Travelling Salesman Problem, Vehicle Routing Problem, loading constraints, Integer Linear Programming, branch-and-cut..

Resumo

Nesta dissertação, abordamos o Problema do Caixeiro Viajante com Coleta e Entrega sobre Carregamento LIFO (PDTSPMS) e métodos para a obtenção de sua solução exata. O problema consiste em determinar o trajeto de menor custo de um veículo que deve atender um conjunto de requisições de clientes. Cada requisição é composta por uma localização de coleta, onde um determinado item é carregado no veículo, e por uma localização de entrega, onde esse item é descarregado. Para realizar o processo de carregamento e descarregamento dos items, o veículo conta com um conjunto de pilhas com capacidade finita. Um item quando carregado neste veículo ocupa o topo de uma das pilhas e apenas itens que estão no topo das pilhas podem ser descarregados nas correspondentes localizações de entrega.

Apresentamos um nova formulação em Programação Inteira para o problema e propomos um algoritmo *branch-and-cut* para obter a solução ótima dessa formulação. Em particular, utilizamos um conjunto exponencial de desigualdades para modelar a política de carregamento do veículo através da adição de um novo conjunto de variáveis. A partir da inclusão dessas novas variáveis, conseguimos identificar violações da política de carregamento (seja na ordem ou na capacidade) para um dado trajeto através da resolução de problemas polinomiais.

Resultados computacionais mostram que nosso algoritmo é competitivo em relação ao melhor algoritmo proposto na literatura, resolvendo algumas instâncias de teste com menor tempo computacional. Além disso, o algoritmo foi capaz de determinar certificados de otimalidade para duas instâncias não solucionadas anteriormente.

Palavras-chave: Caixeiro Viajante, Roteamento de veículos, restrições de carregamento, Programação Linear Inteira, *branch-and-cut*..

List of Figures

List of Tables

List of Acronyms

B&C	branch-and-cut
ILP	Integer Linear Programming
LIFO	Last-in-First-Out
VRP	Vehicle Routing Problem
TSP	Travelling Salesman Problem
PDTSP	Pickup and Delivery Travelling Salesman Problem
PDTSPL	Pickup and Delivery Travelling Salesman Problem with LIFO Loading
PDTSPMS	Pickup and Delivery Travelling Salesman Problem with Multiple Stacks
DTSPMS	Double Travelling Salesman Problem with Multiple Stacks

Contents

Chapter 1

Introduction

One of the most studied problems in Combinatorial Optimization is the Vehicle Routing Problem (VRP). Its rich structure and economic importance in real world yield a variety of related problems in which more complex operating constraints such as vehicle capacity [Baldacci et al., 2004], precedence relations between clients [Santos et al., 2013], time windows [Azi et al., 2010], to cite just a few, are of interest.

Variants of the problem that integrate routing and loading issues have received special attention recently. In some transportation applications, not only the weight of items being transported are a concern in the design of feasible routes, but other characteristics such as dimensional shape, the manipulation of these items (for example, when dealing with fragile items), among others. In that sense, one operating constraint arises when dimensional sizes of items are considered and the cargo must fit inside the vehicle loading area. Another constraint emerges when unloading operations should be performed without rearranging the items, so the vehicle loading sequence determines which items can be unloaded at a given location.

The solution of vehicle routing problems in the literature mostly involves solving some variant of the famous Travelling Salesman Problem (TSP). Traditionally, the problem is stated as: given a set of cities and the cost of travelling between each pair of them, find a minimum cost tour that visits each city exactly once. Despite its simple statement, the optimal solution for the TSP is a classical combinatorial optimization challenge. The problem has been studied since the 1950 decade and it is intrinsically related to the developments in the fields of Mathematical Programming.

In this work, we tackle the Pickup and Delivery Travelling Salesman Problem with Multiple Stacks (PDTSPMS). Suppose that a single vehicle is available to serve a set of customer transportation requests defined by the pickup and delivery locations of an item. In order to perform this task, the vehicle uses a set of independent stacks of finite capacity.

Loading and unloading of the items being transported in each stack must follow the Last-in-First-Out (LIFO) policy. The objective is to serve all requests with a route of minimum cost that satisfies the loading/unloading policy.

1.1 Motivation

The problem finds applications in the routing of transportation vehicles where freight is loaded and unloaded in the vehicles from the rear. For instance, in the transportation of heavy or fragile items, or hazardous materials, reallocating the items en route can be prohibitive. In that sense, just the last item loaded in a stack is accessible and it must be delivered before all other items that were loaded before it.

Ladany and Mehrez [1984] show a real application at an expedition company in Israel. The problem was faced when planning the route of a truck collecting items in the city of Tel Aviv and, after all items were collected, delivering them in the city of Haifa. The truck could only be loaded through the rear door. Reshuffling of items inside the vehicle was a very cumbersome task, so that the unloading sequence was defined by the inverse order of loading.

The problem was also presented by Levitin and Abezgaouz [2003] in the context of a warehouse, where automated guided vehicles (AGV) are used for carrying multiple loads between workstations. They cite an example in manufacturing systems, where the loading and unloading of the items being transported by the AGV's consume more time than transportation itself. In this scenario, material are transported in pallets, and each one picked up by the AGV is placed on the top of a batch of pallets. When an AGV needs to deliver a pallet at some workstation and there are pallets addressed to other workstations above it, all of them need to be unloaded and loaded back after the desired pallet was unloaded. They proposed an algorithm to plan an optimal route for an AGV in such a way that only the last loaded pallet can be unloaded at the workstations, avoiding this rearrangement during the unloading process.

1.2 Contributions

The main concern in this dissertation is to describe a new Integer Linear Programming (ILP) formulation and a method for obtaining exact solutions to the PDTSPMS. Our approach will be tested with instances that have been tackled before in the literature. We build our ILP model based on a previous formulation recently presented for a more restricted case where the vehicle contains just one stack of unlimited capacity. In fact, the formulation presented in the literature for the PDTSPMS was built on the same work, but our approach differ from the one in the literature in the type of variables used and in the way the LIFO policy is tackled.

A branch-and-cut (B&C) approach is proposed and we show that, within our formulation, inequalities used to dictate the LIFO policy are simpler to express. This easiness in the expression decrease the computational effort involved in the separation of violated inequalities. The algorithm for separating them is described in this dissertation. Albeit the proposed formulation uses more variables, it will be shown that they are indeed more descriptive, potentially leading to a better description of the feasible set of the problem.

We assess the quality of our algorithm and compare the results with those obtained by the best algorithm available in the literature. Our approach proved to be competitive, solving some benchmark instances in less time than the state-of-the art algorithm. Also, we were able to achieve optimality certificates for two instances not previously solved.

We convey all these ideas throughout the text as follows:

Chapter 2 **The Pickup and Delivery Travelling Salesman Problem with Multiple Stacks** The precise definition of PDTSPMS is given. The mathematical notation adopted in the text and some combinatorial properties of the problem are presented. We review the literature on the problem, highlighting algorithmic approaches usually employed to achieve a solution as well as related work on problems resembling PDTSPMS.

Chapter 3 **Integer Programming Formulation for PDTSPMS** We propose a novel ILP formulation for PDTSPMS and develop three classes of valid inequalities. Differences between our approach and the literature are identified. We stress the weakness and advantages of our formulation vis-a-vis the formulation in the literature.

Chapter 4 **An Exact Algorithm for PDTSPMS** We employ the ideas developed in previous chapter within a *branch-and-cut* algorithm to solve PDTSPMS exactly. Further implementation issues are described.

Chapter 5 **Computational Results** Using our proposed algorithm, we solve instances from the literature and analyse the results as compared with those obtained with the state-of-the-art algorithm.

Chapter 6 **Conclusion and Future Work** Finally, we assess the contributions given by this work and discuss possible directions for further work.

Chapter 2

The Pickup and Delivery Travelling Salesman Problem with Multiple Stacks

In this chapter, the problem we address in this dissertation is formally stated. The adopted notation and definitions used in the mathematical context through the text are presented in Section 2.2. Also, we review the literature on the problem, pointing out solution strategies and other related problems in Section 2.3. Some aspects concerning the structure of the solutions are presented in Section 2.4.

2.1 Preliminary definitions

Given a set V of points in some space, we define $A = \{(i, j) | i, j \in V\}$ as the set of arcs between two points in V and use $G(V, A)$ to denote the directed graph G on V with arcs A, and use the terms points and vertices interchangeably. Also, we define the *cost* of arc $(i, j) \in A$ by c_{ij} as some constant value (for example, the distance between points i and j or the time spent traversing the arc).

A *walk* in G is a finite, non-empty sequence of vertices $W = v_0, v_1, ..., v_k$ such that $(v_i, v_{i+1}) \in A$ for $0 \leq i < k$. If all vertices in W are distinct, then we call W a *path*. If $v_0 = v_k$ then we call it a *tour*. The cost of a path W is the sum of the costs of its constituent arcs.

Define $\mathbf{x} = \{x_{ij} | (i, j) \in A\}$ as the incidence vector of a tour in G such that $x_{ij} = 1$ iff arc (i, j) is traversed by the tour and 0 otherwise. Let $\mathcal{F} = \{\mathbf{x}_1, ..., \mathbf{x}_r\}$ be the set of incidence vectors of tours in G. The *convex hull* of $\mathcal{F}$, $\mathcal{Q} = \text{conv}(\mathcal{F})$, is the polytope of tours in G. The polyhedral region obtained by replacing the integrality $x_{ij} \in \{0, 1\}$ by $0 \leq x_{ij} \leq 1$ is $\mathcal{P}$, the feasible set of the linear relaxation, such that $\mathcal{Q} \subset \mathcal{P}$.

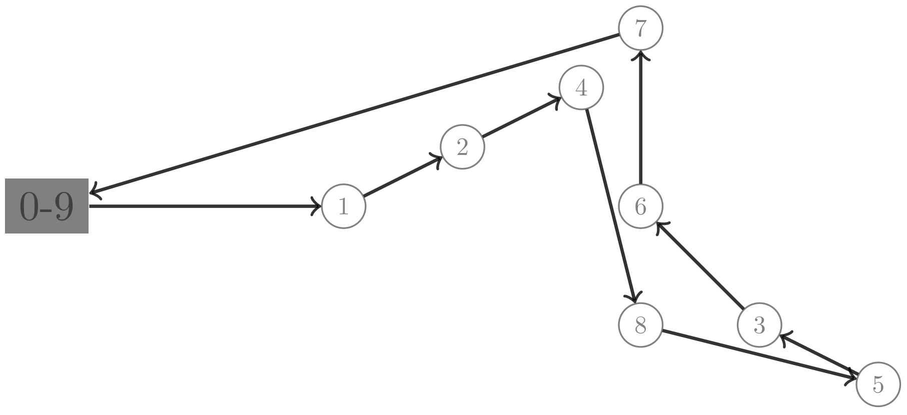
In this work, we consider Integer Linear Programming formulations whose feasible set are bounded and non-empty. Also, the cost c_{ij} is integer-valued, and we do not make any assumptions on the value of c_{ji} (the *Asymmetric TSP*) or assume the triangle inequality to hold.

The *separation problem* for $\mathcal{Q}$ is the problem of, given a vector $\mathbf{x} \in \mathbb{R}^{|A|}$, decide whether $\mathbf{x} \in \mathcal{Q}$ or find some vector $\mathbf{a} \in \mathbb{R}^{|A|}$ and a number $b \in \mathbb{R}$ such that $\mathbf{a}^T\mathbf{y} \leq b \ \forall \mathbf{y} \in \mathcal{Q}$ and $\mathbf{a}^T\mathbf{x} > b$. In the latter case, we say that the *valid inequality* $\mathbf{a}^T\mathbf{y} \leq b$ is violated by $\mathbf{x}$. A class $\mathcal{C}$ is a family of inequalities within the ILP formulation that models one characteristic that is satisfied by all elements of $\mathcal{F}$. A separation procedure for a class $\mathcal{C}$ and a vector $\mathbf{x}$ is a subroutine that finds one or more inequalities in class $\mathcal{C}$ violated by $\mathbf{x}$.

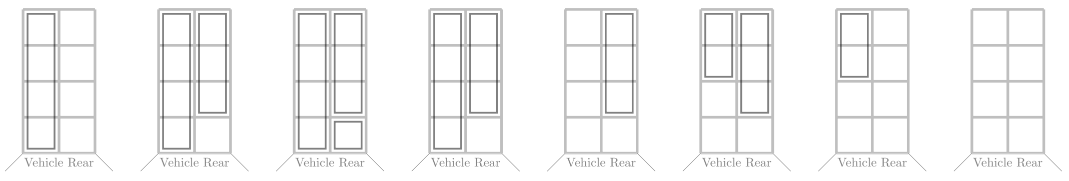
2.2 Problem definition

The Pickup and Delivery Travelling Salesman Problem with Multiple Stacks is defined on a direct (complete graph) $G = (V, A)$, where $V = \{0, 1, ..., 2n+1\}$ is a set of locations in some given space and $A = \{(i, j) : i, j \in V\}$ is the set of arcs between those locations such that c_{ij} is the cost of traversing arc $(i, j) \in A$. Locations 0 and $2n+1$ represent the initial and final depots, respectively. Locations $P = \{1, 2, ..., n\}$ represent pickup points while locations $D = \{n+1, ..., 2n\}$ represent delivery points. A single vehicle must visit each location attending the transportation requests, where n is the number of requests. Each request is defined by a location of pickup $i \in P$, where an item of size q_i is loaded into the vehicle and by a corresponding delivery location $n + i \in D$, where this same item is unloaded. The vehicle contains a number K of loading stacks, each of them with a constant capacity $Q \geq q_i \ \forall i \in P$. Loading and unloading operations for each stack must follow the LIFO policy: when loading an item, it is placed at the top of the stack and is the only item loaded in that stack which can be directly accessed from the rear of the vehicle. Thus, only items at the top of each stack can be unloaded and, consequently, if the vehicle visits a delivery location, only the corresponding delivery locations of those items can be visited. The aim of the problem is to determine a route of minimum total cost that starts at depot 0, transports each item between its associated pickup and delivery locations while satisfying the LIFO policy, and ends at depot $2n+1$. For the sake of simplicity, throughout the text we will refer to the item loaded into the vehicle at location $i \in P$ and unloaded at $n+i \in D$ as item i. Also, let $M = \{0, ..., K-1\}$ be the set of labels for each stack, and associate an item of size $-q_i$ to the delivery location $n + i \in D$.

In Figure 2.1, we depict an optimal solution for an instance of the problem consisting of four customer requests ($n = 4$) which must be attended by a vehicle containing two stacks of capacity $Q = 4$. The initial (location 0) and the final (location $2n + 1$) depots are the same location. The four items to be collected at locations 1, 2, 3, 4 and delivered at locations 5, 6, 7, 8 have lengths $q_1 = 4$, $q_2 = 3$, $q_3 = 2$ and $q_4 = 1$. In part 2.1a, we depict the route traversed by the vehicle and, from 2.1b to 2.1i, we illustrate how items are arranged inside the vehicle after the loading and unloading operations are performed. After leaving depot 0, the vehicle loads item 1 on the first stack. This configuration is illustrated in part 2.1b. Then, items 2 (2.1c) and 4 (2.1d) are loaded on stack 1. Item 4 is delivered at location 8 and item 1 at 5. Parts 2.1e and 2.1f show the stacks configuration immediately after those operations, respectively. Item 3 is loaded on stack 0, and item 2 is unloaded from stack 1 at location 6. Finally, the vehicle unloads item 3 from stack 0 at location 7 and returns to the depot (location 9) with all stacks empty.



(a)



(b) (c) (d) (e) (f) (g) (h) (i)

Figure 2.1: Example of a PDTSPMS route (a) and the stacks configuration during the loading and unloading process (from b to i).

PDTSPMS consists of the general optimization problem below. In Chapter 3, we describe ILP formulations for obtaining elements in $\mathcal{F}$ and how to impose both the precedence and the LIFO policy.

$$\left\{ \min \sum_{(i,j)\in A} c_{ij} x_{ij} | \mathbf{x} \in \mathcal{F} \text{ satisfying the precedences, LIFO and capacity of each stack.} \right\}$$

2.3 Literature review and related work

The PDTSPMS firstly appeared in the literature in a more relaxed version, namely, the Pickup and Delivery Travelling Salesman Problem with LIFO Loading (PDTSPL) in which the vehicle contains just one stack of unlimited capacity. This problem was tackled with heuristic approaches in Carrabs et al. [2007b] where instances varying from 375 to 500 requests were considered. Li et al. [2011] developed a *Variable Neighbourhood Search* (VNS) heuristic using search operators based on a tree data structure and applied the algorithm for solving instances with up to 500 requests. The first exact solution methods were based on branch-and-bound algorithms using TSP relaxations [Pacheco, 1997; Cassani, 2004]. Instances with up to 11 requests were solved. In Carrabs et al. [2007a] the size of solved instances was increased to 15, and some instances with up to 21 requests were also solved to optimality using a different branch-and-bound scheme with additive lower bounds based on relaxations of the TSP given by the assignment and shortest spanning r-arborescence problems. Recently, Cordeau et al. [2010] proposed three integer formulations for the PDTSPL, two of them for the capacitated case, and one for the uncapacitated case, with focus on the latter case. In particular, they proposed an exponentially-sized set of inequalities to impose the LIFO policy and some sets of valid inequalities used within a B&C algorithm capable of solving instances with up to 25 requests.

Another problem related to PDTSPMS is the Double Travelling Salesman Problem with Multiple Stacks (DTSPMS), introduced by Petersen and Madsen [2009]. In this problem, the vehicle also contains a number of stacks of limited capacity to store the items, but must collect all the items before delivering any of them. After collecting all the items, the vehicle returns to the depot and the delivery route must consider the LIFO policy of the stacks. The authors presented a mathematical formulation and some meta-heuristic solution approaches to the problem. Felipe et al. [2009] proposed four neighborhood structures and applied them to a VNS heuristic. Côté et al. [2012b] developed a *Large Neighbourhood Search* (LNS) heuristic for the PDTSPMS and applied it to DTSPMS instances. A *local search* approach was proposed by Urrutia et al. [2013]. Rather than applying the heuristic to construct the routes, the authors applied the local search to the construction of a feasible loading plan and used a *dynamic programming* algorithm to map the plan into corresponding optimal routes.

Concerning the exact solution of the DTSPMS, Petersen et al. [2010] presented several exact approaches, solving instances with up to 25 requests. An exact approach to the DTSPMS was also presented by Lusby et al. [2010]. The method is based on finding the k-best tours to each of the separate pickup and delivery routes and matching the solutions leading to a feasible loading plan. Borne et al. [2012] addressed the uncapacitated version of DTSPMS. They provide a polyhedral study of the problem and an ILP formulation for the case where two stacks are available and for which the linear programming relaxation is polynomial-time solvable.

As a more constrained Travelling Salesman Problem, the PDTSPMS can benefits from the advancements already made for this classical problem. Scientific literature for the TSP dates back to the 1950 decade and is intrinsically linked to the developments of Integer Programming itself [Chvátal et al., 2010]. For an extensive study on the aspects of the TSP solution, the reader is referred to Applegate et al. [2007].

The Pickup and Delivery Travelling Salesman Problem can also be viewed as a restricted version of PDTSPMS. A more general case is the *Precedence Constrained* TSP, in which each vertex has one or more predecessors, and was first addressed by Balas et al. [1995]. The authors presented a polyhedral study and derived several classes of facet inducing inequalities, some of them used in the formulation proposed in this dissertation. Ascheuer et al. [2000] proposed a B&C algorithm for this problem, solving instances of the asymmetric TSP varying from 18 to 101 vertices and containing from four to 131 precedence relations among the vertices. A polynomial formulation for the problem was proposed by Sarin et al. [2005]. Gouveia and Pesneau [2006] proposed new extended formulations for the Precedence Constrained TSP using extra binary variables to model the precedence relation among the vertices. They derived a set of valid inequalities and developed a B&C algorithm capable of obtaining the exact solution of one not previously solved instance. More recently, Dumitrescu et al. [2010] tackled the Pickup and Delivery Travelling Salesman Problem (PDTSP), in which a pick-up must precede its corresponding delivery. The authors presented polyhedral results and a B&C algorithm capable of solving instances not solved by previous approaches, involving up to 35 requests. The PDTSP was also solved with exact methods in Kalantari et al. [1985] and Hernández-Pérez and Salazar-González [2004]. For an extensive survey on pickup and delivery problems, the reader is referred to Berbeglia et al. [2007].

Combining routing and loading issues of vehicles has been the concern of many works in the VRP literature. In particular, two-dimensional and three-dimensional loading constraints, in which geometrical aspects are taken into account when loading an item inside the vehicle, are solved in Iori et al. [2007] and Tarantilis et al. [2009], respectively. Other kinds of loading constraints can be found on literature. For an extensive overview of problems arising from the combination of both routing and loading aspects, the reader is referred to Iori and Martello [2010].

Recently, Côté et al. [2012a] proposed the first exact algorithm for solving the PDTSPMS. They provided three formulations used within a B&C algorithm. The best results were obtained with a formulation using an exponential number of inequalities that extend the LIFO constraints proposed by Cordeau et al. [2010] for the case where more than one stack is available, and using some new capacity inequalities adapted from the VRP for the PDTSPMS. Instances with up to 21 requests are solved by the algorithm, where two classes of instances were considered, namely, instances with unit size items and with items varying in size from one to ten. They also used their algorithm effectively on DTSPMS instances.

2.4 The structure of PDTSPMS

We next illustrate how the values of K and Q, the number of stacks and the capacity of each, respectively, are crucial to determine the set of feasible solutions of PDTSPMS.

For a given problem instance, suppose that K equals (or is greater than) the number of requisitions, n. Then, an optimal route that complies with the precedence relations between each pickup and delivery locations is an optimal solution to PDTSPMS. Note that when visiting a pickup location, this vehicle always has an empty stack. Thus, items loaded into this vehicle are readily available for unload, and the corresponding deliveries locations can be visited. In that sense, whereas an optimal solution to the PDTSP provides a lower bound for PDTSPMS, an optimal solution for the latter tends to be more different from the former as K is decreased from n to 1.

Another remarkable issue concerns the total capacity available inside the vehicle, namely, $T = K \times Q$. The example below shows that it is possible to have optimal tours of different values even when the total loading area inside the vehicles are the same.

Consider an instance of PDTSPMS with n requests, where the pickup and delivery locations are given by the vertices of a regular $(2n+1)$-gon inside the $\mathbb{R}^2$, and assume the location of vertex 0 equals the location of vertex $2n+1$ (that is, the initial and the final depots are the same location). Suppose that the optimal solution for the PDTSP in this instance is the perimeter of the polygon, and that all items have capacity one. Starting at 0, the next nine vertices represent six pickup locations followed by three delivery locations. The dashed arc represents the path p from $n+1$ to the final depot, such that $p = n+6, n+5, n+4, 7, n+7, ..., n, 2n, 0$. That is, p can be attended using just one position of a stack. If a vehicle with total capacity $T = 6$, where $K = 2$ and $Q = 3$, was to follow the path from 0 to 6, then a configuration of the stacks immediately after loading item 6 is illustrated in part (b) of Figure 2.2. Accordingly, after leaving 6, this vehicle can continue as in the optimal solution and follow the path $n+3, n+2, n+1, p$. On the other hand, for a vehicle with $K = 3$ and $Q = 2$ it is not possible to follow this same path. After loading items 1 to 6, in order for this vehicle follows the path $n+3, n+2, n+1$, items 3, 2 and 1 should be at the top of some stack when the vehicle visit each respectively delivery location. But no such configuration is possible when $K = 3$ and $Q = 2$ for the loading sequence 1, 2, 3, 4, 5, 6, as we depict in part (c) of Figure 2.2. The value of the optimal tour using the first vehicle, z_1^*, is the same value of the optimal PDTSP tour, whereas an optimal tour using the second vehicle does use arcs outside the perimeter of the polygon. The value of this tour, z_2^*, satisfy $z_2^* > z_1^*$.

If one of the dimensions K or Q is held constant, the value of an optimal solution for a PDTSPMS instance using a vehicle with total capacity $\bar{T}$ is indeed a lower bound on the value for this same instance using a vehicle with total capacity inferior to $\bar{T}$. The smaller loading area can be seen as included in $\bar{T}$. If K is held constant, for example, then an optimal solution for a vehicle with stack capacity two is still valid for a vehicle with stack capacity three (that is, the extra capacity is not used).

Finally, we observe that the solution of the PDTSPMS is highly symmetrical. All stacks have the same capacity and since we do not constrain any item to be load or unload from a specific stack, they are all identical. Thus, each solution of the PDTSPMS has a set of K! equivalent solutions, one for each permutation of the K stacks labels.

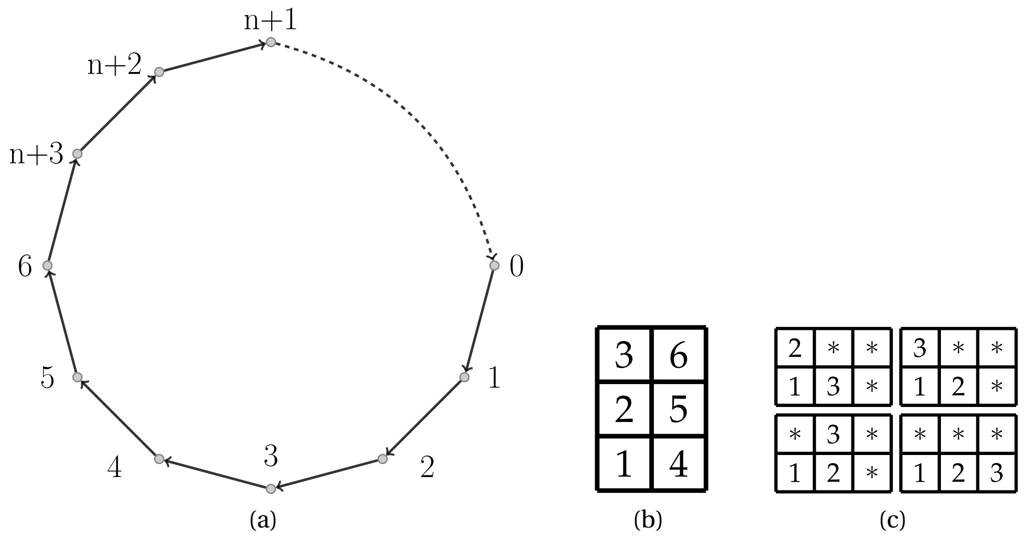


Figure 2.2: An optimal route for the PDTSP is depicted in (a). The route starts at depot 0, visits six pickup locations and three delivery locations. The dashed arc represents the path p. In (b) we illustrate a possible configuration of the load area after items 1-6 were loaded inside a vehicle with $K = 2$ and $Q = 3$. All possible configurations for loading these items on a vehicle with $K = 3$ and $Q = 2$ are shown in (c). Observe that this vehicle can not follow the path from 0 to $n+1$ in the perimeter, as any of these configurations allows the path $n+3, n+2, n+1$ to be visited. Thus, the value of the optimal solution for the PDTSPMS using this vehicle is strictly greater than for the vehicle with $K = 2$ and $Q = 3$. ($* \in \{4,5,6\}$)

Chapter 3

Integer Programming Formulations for the PDTSPMS

This chapter presents the proposed ILP formulation for the Pickup and Delivery Travelling Salesman Problem with Multiple Stacks. Since our work builds on formulations from previous work in the literature, we firstly review an ILP model for the Pickup and Delivery Travelling Salesman Problem. Then, we show how the LIFO policy can be imposed in the previous model using an exponential-sized class of inequalities, considering a vehicle with just one stack of unlimited capacity (PDTSPL). This class is then extended to cope with the more general case where the vehicle contains multiple stacks of limited capacity (PDTSPMS). Finally, based on all these previous work, we describe the formulation we propose in Section 3.2.

Some additional notation is needed before we introduce the formulations. In what follows, we use the notation below:

- $\bar{S} = V/S,\ S \subseteq V$;
- $x(S) = \sum_{i,j \in S} x_{ij}$;
- $x(S,T) = \sum_{i \in S, j \in T} x_{ij}$;
- $x(i,S) = x(\{i\},S)$;
- $x(S,i) = x(S,\{i\})$;
- $\mathcal{S}$ is the collection of subsets $S \subset V$, such that $0 \in S$, $2n+1 \notin S$ and there exists $i \in P$ for which $i \notin S$ and $n+i \in S$;
- Ω is the collection of subsets $S \subset P \cup D$, such that there exists at least a $j \in P$ for which $j \in S$ and $n+j \notin S$, or $n+j \in S$ and $j \notin S$;

- $i \prec j$ if the vehicle visits location i before location j;
- $\pi(S) = \{i \in P : n+i \in S \subset V\}$ denotes the *predecessors* of subset S;
- $\sigma(S) = \{n+i \in D : i \in S \subset V\}$ denotes the *successors* of subset S;
- $q(S) = \sum_{i \in S} q_i$.

The path from the initial depot 0 to the final depot $2n+1$ is a tour if they are the same location. However, it will be referred as a *tour* even if they are distinct locations.

3.1 From the TSP to the PDTSPMS

We consider a standard ILP formulation for the asymmetric TSP, consisting of two equations (the *degree equations*) for each vertex and an exponentially-sized class of inequalities known as *subtour elimination constraints* (SEC). Formulations with polynomial sets of variables and constraints exist for the TSP (e.g. Miller et al. [1960]), although the polyhedra associated with their linear relaxation are usually weaker compared with the standard formulation (see [Padberg and Sung, 1991]). In any case, a polyhedral description $\mathcal{P}$ such that $\mathcal{P} = \text{conv}(\mathcal{F})$ for this problem is hard to find unless $\mathbf{P} = \mathbf{NP}$.

In the remainder of this section, we show how the basic TSP formulation is extended in order to impose the precedence among each pair of pickup and delivery (that is, a tour satisfying $i \prec n+i \ \forall i \in P$) and the LIFO policy for loading and unloading the items, first for the PDTSPL and then for the PDTSPMS. Valid inequalities used to strengthen the formulations and separation procedures used for identifying violated inequalities within a B&C algorithm for each exponentially-sized class are described in Chapter 4.

3.1.1 ILP formulation for PDTSP

First, we show a classic ILP formulation for the PDTSP. To this end, associate binary variables x_{ij} to each arc $(i,j) \in A$, taking value 1 if and only if location j is visited immediately after i, and 0 otherwise.

The following model formulates PDTSP:

$$\min \left\{ \sum_{(i,j) \in A} c_{ij} x_{ij} | \mathbf{x} \in \mathcal{P} \cap \mathbb{B}^{|A|} \right\} \tag{3.1}$$

where the polyhedral region $\mathcal{P}$ is given by:

$$x(i,V) = 1 \qquad i \in P \cup D \cup \{0\} \tag{3.2}$$

$$x(V,i) = 1 \qquad i \in P \cup D \cup \{2n+1\} \tag{3.3}$$

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

$$x(S) \leq |S| - 2 \qquad S \in \mathcal{S} \tag{3.5}$$

$$0 \leq x_{ij} \leq 1 \qquad (i,j) \in A \tag{3.6}$$

The objective function (3.1) minimizes the cost of the vehicle route. Constraints (3.2) and (3.3) (the degree constraints) ensure that each location is visited exactly once. Observe that the initial and final depots are different entities inside the model, though they may correspond to the same location. Constraints (3.4) are the *subtour elimination constraints* (SEC) and impose connectivity on the route by eliminating all cycles with vertices in $P \cup D$ since any cycle on the vertices in S requires $|S|$ arcs. Constraints (3.5) were proposed by Balas et al. [1995] in the context of the Precedence-constrained TSP. Observe that for $S \in \mathcal{S}$, there exists $i \in P$ for which $i \notin S$ and $n+i \in S$. Thus, inequalities (3.5) eliminate all paths with vertices in S that start in 0 and visiting $n+i$ before visiting i, hence violating the precedence of the pickup and delivery locations.

Observe that an integer feasible solution of $\mathcal{P}$ correspond to an incidence vector of a tour in G that satisfies the precedence of each pair of pickup and delivery.

3.1.2 The PDTSPL

Considering PDTSPL, in which the vehicle contains only one stack of infinite capacity, Cordeau et al. [2010] introduce a class of inequalities to impose the LIFO policy in the model (3.1)-(3.6). Their proposed constraints are of the form:

$$x(i,S) + x(S) + x(S,n+i) \leq |S| \qquad S \in \Omega, i, n+i \notin S, i \in P \tag{3.7}$$

Inequality (3.7) forbids a path from $i \in P$ to $n+i \in D$ going through a set $S \in \Omega$. Observe that in such a path, if $j \in S$ and $n+j \notin S$, the vehicle loads item i in the stack, then loads item j but unloads item i before item j, violating the LIFO policy. If $n+j \in S$ and $j \notin S$, the vehicle loads item i and then unloads item j before unloading item i. The authors show that inequalities (3.7) are sufficient to impose the LIFO policy.

3.1.3 The PDTSPMS

Note, however, that inequalities (3.7) are not necessarily valid for PDTSPMS when multiple stacks are available ($K > 1$). In particular, observe that items i and j in the above example could be loaded in different stacks. Thus, immediately after visiting the set $S \in \Omega$, both delivery locations, $n+i$ and $n+j$, could be visited by the vehicle. Let $\mathcal{L} = \{i_1, i_2, ..., i_l\}$ be a sequence of pickup locations for which $i_1 \prec \ldots \prec i_l \prec n+i_1 \prec \ldots \prec n+i_l$. Items in $\mathcal{L}$ are said to cross each other. Observe that, for a vehicle with K stacks, a sequence $\mathcal{L}$ with $K+1$ or more items leads to an infeasible solution concerning the LIFO policy.

Côté et al. [2012a] showed how to extend inequalities (3.7) for the case when $K \geq 2$. Consider a set $\mathcal{L} = \{i_1, i_2, ..., i_{K+1} \in P,\ i_j \neq i_k\}$ of pickup locations where each element in $\mathcal{L}$ crosses all other elements. Then, $i_1 \prec i_2 \prec ... \prec i_{K+1} \prec n+i_1 \prec ... \prec n+i_{K+1}$ is induced. The authors propose the following inequalities to forbid paths in which such a situation occurs:

$$\sum_{h=1}^{K} [x(i_h, S_{i_{h+1}}) + x(S_{i_{h+1}}) + x(S_{i_{h+1}}, n+i_h)] \leq \sum_{h=2}^{K+1} |S_{i_h}| + K - 1 \qquad (3.8)$$

where $S_{i_{h\geq 2}} = \{i_h, i_{h+1}, ..., i_{K+1}, n+i_1, ..., n+i_{h-2}\}$ ($n+i_0$ is defined as i_{K+1} and the sequence from $n+i_1$ to i_{K+1} is empty for $h=2$).

An example for a vehicle with two stacks is depicted in Figure 3.1. Inequality (3.8) forbids all paths from i_1 to $n+i_2$ going through sets $S_{i_2} = \{i_2, i_3 = n+i_0\}$ and $S_{i_3} = \{i_3, n+i_1\}$.

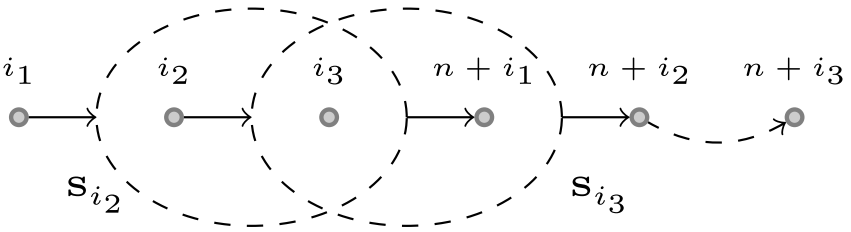


Figure 3.1: Example of a forbidden path for a vehicle with $K = 2$ stacks.

Concerning the stacks capacity, Côté et al. [2012a] derived an adaptation of the classic *rounded capacity inequalities*, widely used in the context of the capacitated VRP, considering the whole vehicle loading area ($K \times Q$) instead of a single stack capacity, Q. Since those inequalities are not sufficient to guarantee that the capacity of each stack will not be exceeded, the authors also propose a set of inequalities of type (3.7), that is, eliminating all paths from a pickup $i \in P$ to $n+i \in D$ going through all vertices inside set S, but considering the case when $z(S) > Q(K-1)$, where $z(S) = max\{q(\pi(S)\backslash S), -q(\sigma(S)\backslash S)\}$. Items crossing item i cannot be loaded in the same stack as i, so they must fit in the remaining $Q(K-1)$ available space. Also, the authors extend these inequalities to deal with a set containing k items crossing each other, where $2 \le k \le K-1$ items. To reduce the complexity in the procedure for separating such inequalities, the authors only use values of $k = 2$ or 3. Again, these inequalities are not sufficient to ensure a solution not violating the stacks capacity.

To ensure a solution where the capacity of each stack is not exceeded, each time a tour satisfying the precedence of each pair of pickup and delivery is found, a *packing problem* is solved. The solution of this problem is an assignment of each item to a stack, such that the LIFO policy and the capacity constraints are satisfied. Suppose that the tour is fixed and known. Let $a_{ir} = 1$ if item i is loaded inside the vehicle when the location at position r of the tour is visited and $a_{ir} = 0$, otherwise. Let $z_{ik} = 1$ if the vehicle loads item i in stack k and $z_{ik} = 0$, otherwise. The packing problem is formulated as:

$$\sum_{k\in M} z_{ik} = 1 \qquad i \in P \qquad (3.9)$$

$$z_{ik} + z_{jk} \le 1 \qquad (i,j) \in I, k \in M \qquad (3.10)$$

$$\sum_{i\in P} a_{ir} q_i z_{ik} \le Q \qquad r \in \{1,2,\ldots,2n\}, k \in M \qquad (3.11)$$

$$z_{ik} \in \{0,1\} \qquad i \in P, k \in M \qquad (3.12)$$

where I is the set of all pair of items crossing each other in the given tour (computed in $O(n^2)$ time).

Each item is assigned to just one stack by (3.9). (3.10) impose items crossing each other to be loaded on different stacks. The capacity of each stack is preserved by (3.11), stating that at any location of the path p, the total length of items loaded in each stack do not exceed Q. If the solution of this packing problem is infeasible, an inequality is then inserted in the model to cut this path form the set of feasible solutions. Otherwise, the tour is feasible concerning both the LIFO policy and the capacity constraints.

To conclude this section, Côté et al. [2012a] also note that no polynomial time algorithm is expected for the packing problem. Indeed, in the particular case where all items are picked up before any delivery is made (DTSPMS), the packing problem is **NP**-Hard [Toulouse and Wolfler Calvo, 2009; Casazza et al., 2012].

3.2 The proposed ILP formulation for PDTSPMS

In this section, we propose an alternative ILP formulation for the PDTSPMS in which we explore model (3.1)-(3.6), adding new variables and constraints to handle the multiple stacks availability. Our main goal is to facilitate the expression of inequalities for the LIFO policy within this formulation and to avoid the need of solving a difficult packing problem to ensure a feasible solution. In particular, the model (3.1)-(3.6) gives the base of our formulation, providing the set of tours satisfying the PDTSP condition. To state the LIFO conditions, we utilize a new set of variables.

Consider the following binary variable vector $\mathbf{y} \in \mathbb{B}^{|A||M|}$:

$$y_{ij}^k = \begin{cases} 1 & \text{if immediately after visiting location } i \in V\text{, the vehicle loads } (j \in P) \\ & \text{or unloads } (j \in D) \text{ item } j \text{ to or from stack } k \\ 0 & \text{otherwise} \end{cases}$$

Using variables y_{ij}^k we have, at each arc $(i,j) \in A$, information concerning the operation performed by the vehicle when it arrives at location j coming from location i. Variables y_{ij}^k are linked to variables x_{ij} by the equations:

$$x_{ij} = \sum_{k \in M} y_{ij}^k \qquad (i,j) \in A \tag{3.13}$$

If an item is loaded in stack k, it must be unloaded from this same stack. This constraint can be stated through the following equations:

$$\sum_{i \in V/\{2n+1\}} y_{ij}^k = \sum_{i \in V/\{0,2n+1\}} y_{i,n+j}^k, \ j \in P,\ k \in M \tag{3.14}$$

Observe that depot $2n+1$ cannot be the predecessor of any location nor depot 0 can be the predecessor of a delivery location. In both cases, a precedence is violated by the vehicle.

Next, we show how to express the LIFO constraints using y_{ij}^k variables. Instead of finding a set of $K+1$ items that cross each other, thus violating the LIFO policy, and K subsets $(S_{h_2}, \dots, S_{h_{K+1}})$ to write inequalities (3.8), we adapt inequalities (3.7) using variables y_{ij}^k to impose that items crossing each other must be loaded on different stacks. Let $i,\ j \in P$ such that $i \prec j \prec n+i$, and $S \in \Omega$, $j \in S$, $i, n+i, n+j \notin S$. As noted earlier, a path from i to $n+i$ through S is LIFO infeasible when the vehicle contains only one stack. On the other hand, if $K \geq 2$ then items i and j must be loaded on different stacks. As we have information on which stack both items i and j are loaded, we can use variables $y^k(S,j)$ and $y^k(S,n+i)$ to remove from the solution space not all the paths from i to $n+i$ going through S, but the paths with a *stack configuration* that leads to infeasibility (that is, a configuration where the vehicle loads items i and j on the same stack when traversing that path).

In Figure 3.2 we depict a graphical representation. Observe that items i and j cross each other in this example. Thus, if the vehicle goes from i to $n+i$ through S (entering and leaving S only through i and $n+i$, respectively) then items i and j must be loaded on different stacks, or the vehicle must leave and enter S more than once. Using the new set of variables, we can state this condition with the following inequalities:

$$x(i, S\backslash\{j\}) + x(S, S\backslash\{j\}) + y^k(S \cup \{i\}, j) + y^k(S, n+i) \leq |S| \quad S \subset P \cup D,\ i,\ n+i \notin S, j \in S \tag{3.15}$$

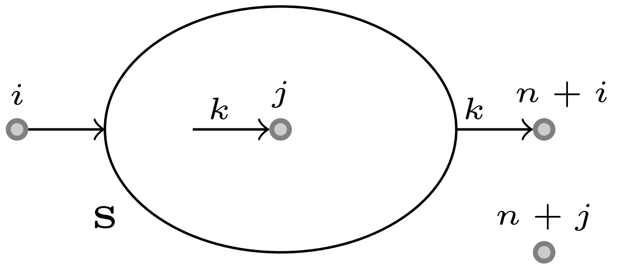


Figure 3.2: A forbidden path where items i and j cross each other.

Therefore, inequalities (3.15) cut all paths from i to $n+i$ in which the vehicle (i) loads item i on stack k, (ii) goes through the clients in S (without visiting clients outside S) and, in particular, loads (or unloads) item j on stack k, (iii) leaves S and immediately unloads item i from stack k at location $n+i$.

However, we note the following. Consider the example in Figure 3.3, and suppose that items i and j are in the same stack. In the path from i to $n+i$ going through the set S, the subpath in between i and j is not infeasible in the loading/unloading sequence. In other words, even if we change this subpath, the whole path would still be infeasible, as item i would still be unloaded after the load of item j (and $n+j$ is not in this path). In fact, that is where the real problem lies: item j is loaded and then the vehicle attempts to deliver item i before delivering item j. In Figure 3.3a, we illustrate this situation. Instead of using the set S, we just consider a subset $S' \subseteq S$ containing j (but not containing i, $n+i$ or $n+j$) in our inequalities.

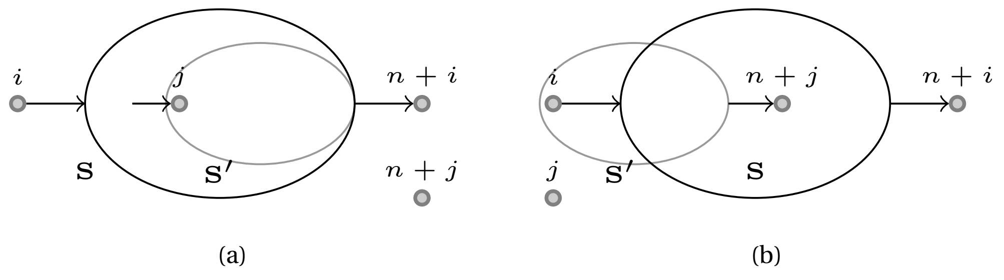


Figure 3.3: The proposed LIFO inequalities.

Now we describe how the LIFO policy can be imposed within our formulation.

$$y^k(\bar{S}', j) + x(S') + y^k(S', n+i) \le |S'| \qquad i, n+i, n+j \notin S',\ j \in S',\ k \in M \tag{3.16}$$

Observe that, for the case where $n+j \in S$ and $j \notin S$, as we depict in 3.3b, then the roles of i and j are interchanged in inequalities (3.16).

Variables **y** are used when we want to state explicitly the stack in which the vehicle loads an item, and variables **x** when we just want to specify an arc in the path. More specifically, we are concerned about the loading/unloading of items i and j in stack k and not about items in $S' \backslash \{j\}$. Note that when items i or j are not loaded in stack k the inequality is trivially satisfied assuming all other constraints in (3.1)-(3.6) are in place. Therefore, inequalities (3.16) cut all paths from j to $n+i$ in which the vehicle (i) loads item j in stack k, (ii) goes through the clients in S' (without visiting clients outside S') and (iii) unloads item i from stack k at location $n+i$.

Our proposed ILP model can be seen as composed of two levels. In the upper level, the PDTSP is formulated and $\mathbf{x} \in \mathbb{B}^{|A|}$ provides a solution for this problem. In the lower level, we use variables $\mathbf{y} \in \mathbb{B}^{|A|K}$ to model the LIFO policy in the tours. The two levels are coupled by constraints (3.13).

In Chapter 4 we describe the separation procedure for the class of inequalities (3.16).

3.2.1 Valid Inequalities

Here, we introduce some valid inequalities within our formulation that explore the particular structure of the PDTSPMS. Valid inequalities used in the context of the PDTSP, that is, not considering the loading and unloading of the items, are presented in Chapter 4.

Successor Inequalities

Suppose that item i is loaded on stack k. Then, if the vehicle visits a delivery location $n+j$ immediately after visiting location i, $j \neq i$, it cannot perform the delivery operation through stack k, as the item on the top of stack k is i. That is, the only item loaded on stack k that can be delivered is i. Then, we can derive the following set of valid inequalities:

$$y^k(V,i) + y^k(i, D\backslash\{n+i\}) \leq 1 \; i \in P, k \in M \tag{3.17}$$

Capacity subtour elimination constraints (CSEC)

Let $S \subset P \cup D$ be a set such that $q(\pi(S)\backslash S) > Q$. If the vehicle performs all pickups in $\pi(S)\backslash S$ on stack k, it must enter and leave the set S more than once. This holds because, if the vehicle enters and leaves S just once, when the vehicle enters set S, the items to be delivered in $\pi(S)\backslash S$ are all loaded in stack k overloading its capacity. This leads to the following valid inequality:

$$x(S \cup \{v\}, S\backslash\Delta) + y^k(S \cup \{v\}, \Delta) \leq |S| - 1 \; v \in P \cup D, \; k \in M, \; q(\pi(S)\backslash S) > Q \tag{3.18}$$

where set $\Delta \subset D$ is composed by the deliveriy locations in S with corresponding pickup locations outside S, and $v \in P \cup D$ is the vertex visited immediately before the vehicle enters the set S.

Figure 3.4 illustrates the idea of inequalities (3.18). In this example, $Q = 10$, $\pi(S)\backslash S = \{a,b,c\}$, $q_a = q_b = 4$ and $q_c = 3$, thus $q(\pi(S)\backslash S) = 11$ and $\Delta = \{n+a, n+b, n+c\}$. If vehicle leaves v, enters and leaves S just once (that is $x(S) = |S| - 1$), items a, b and c can not be all loaded on the same stack, or the vehicle needs to enter and leave S more than once in order to satisfy the inequality.

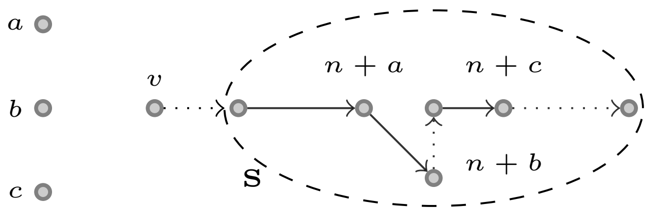


Figure 3.4: A violated capacity subtour elimination constraint. Solid arcs represent unload operations from stack K.

A similar inequality can be derived for the successors $\sigma(S)\backslash S$ of a set S when $-q(\sigma(S)\backslash S) > Q$:

$$x(S \cup \{v\}, S \backslash \Pi) + y^k(S \cup \{v\}, \Pi) \leq |S| - 1 \;\; v \in P \cup D \cup \{0\},\; k \in M,\; -q(\sigma(S) \backslash S) > Q \tag{3.19}$$

where set $\Pi \subset P$ is composed by the pickup locations in S with corresponding delivery locations outside S.

Infeasible path inequalities

The infeasible path inequalities presented for PDTSPMS in Côté et al. [2012a] are used for eliminating tours that violate the LIFO loading policy or the stacks capacity. Given an infeasible path W with $|W|$ arcs, let $x(W)$ denote the sum of x_{ij} variables representing the arcs in W. A path inequality is given by:

$$x(W) \leq |W| - 1,\; W \in \Phi \tag{3.20}$$

where Φ is the set of paths that violate the LIFO or capacity constraints.

In our formulation, we also use a kind of path inequalities but, since the LIFO policy is already enforced by inequalities (3.16), we use them just to eliminate tours for which the capacity of some stack is exceeded. Denote $\bar{W} = \{w \in W : y^k_{iw} = 1\}$ as the set of items in W loaded or unloaded on stack k, and $\hat{W} = \{w \in W : y^k_{iw} = 0\}$ as the items loaded or unloaded on some other stack $l \neq k$. Let $y^k(\bar{W})$ be the sum of variables y^k_{ij} such that (i, j) is an arc in the path W such that $j \in \bar{W}$, and $x(\hat{W})$ be the sum of variables $x_{i,j}$, $(i, j) \in W$ and $j \in \hat{W}$. Our capacity infeasible path inequalities are as follow:

$$y^k(\bar{W}) + x(\hat{W}) \leq |W| - 1,\; k \in M,\; W \in \Psi \tag{3.21}$$

where Ψ is the set of paths for which $q(\bar{W}) > Q$ that is, if the vehicle follows this path loading items in $\bar{W}$ on stack k then the capacity of this stack will be exceeded at the end of the path. An example of an infeasible path inequality is illustrated by figure 3.5. Consider a vehicle with stack capacity $Q = 10$, and that items i_1 and i_3 have length 6 and item i_2 has length 3. In the figure, solid arcs represent operations performed on stack k, and the dashed ones represent operations done on a different stack. Suppose that the vehicle arrives at location j_1 with stack k empty. Items i_1, i_2 and i_3 are loaded on stack k. After loading item i_2, the total used space is 9. When the vehicle arrives at location i_3 the capacity of stack k is exceeded, since the lengths of items i_1 and i_3 sum up to 12.

Following the described notation, $\bar{W} = \{i_1, i_2, n + i_2, i_3\}$, $\hat{W} = \{j_1, n + j_1, n + j_2\}$ and

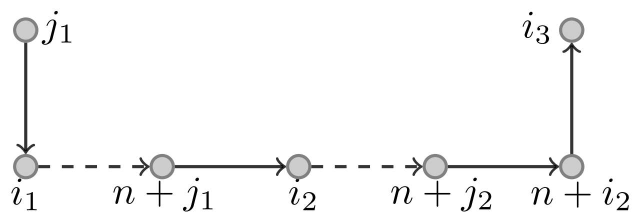


Figure 3.5: Infeasible path inequalities. Dashed arcs correspond to operations performed on a stack different of k. The total item length load on stack k for this path is $q_{i_1} + q_{i_2} - q_{i_2} + q_{i_3} > Q$.

the inequality for this path becomes:

$$y^k_{j_1,i_1} + x_{i_1,n+j_1} + y^k_{n+j_1,i_2} + x_{i_2,n+j_2} + y^k_{n+j_2,n+i_2} + y^k_{n+i_2,i_3} \leq 6 - 1 = 5$$

Observe that for the operations performed on stack k, variables y^k_{ij} are used. If the load/unload operation is performed on another stack, variables x_{ij} are used, meaning that it does no matter which stack is used, the path is still infeasible.

In Chapter 4 we show that separating path inequalities is easy when the solution is integral. Note that we do not limit W to be a tour passing through all clients and, in this way, we may cut infeasible capacity paths of any size that may be part of many infeasible tours.

3.3 Summary

This chapter presented ILP formulations for the PDTSPMS based on a classical model for the PDTSP, but adopting rather different approaches for modelling the LIFO policy. In the formulation proposed by Côté et al. [2012a], the LIFO policy is enforced through inequalities that prohibit paths having $K+1$ or more items crossing each other since, in this case, no packing of the items is possible. When a feasible PDTSP route is found, a packing subproblem is solved to ensure a feasible solution to the PDTSPMS, as the inequalities concerning the capacity of each stack are not sufficient to cut all infeasible solutions.

In our approach, we add more variables to the formulation to tackle the load/unload operations on each arc. In this way, rather than considering a path with $K+1$ items crossing each other, the LIFO inequalities in our formulation state that two items crossing each other need to be loaded on different stacks.

On one hand, the packing subproblem solved in Côté et al. [2012a] is **NP**-Hard, on the other hand, our formulation uses more variables and, thus, solving the model may be harder. Additional variables are often used to achieve compact formulations (with a polynomial set of constraints and variables) for the asymmetric TSP [Gouveia and Pires, 2001]. We refer to Gouveia and Pesneau [2006] for an example in which the additional variables are used in the context of extended formulations to allow a better description of the problem. The authors used an additional (polynomial) set of precedence variables to better model the problem and to derive several exponential sets of inequalities used within a B&C algorithm.

We close this chapter by noting that all the proposed valid inequalities within our formulation were derived from the particular structure of the PDTSPMS. A more in-depth analysis of the polyhedral structure of the problem could be interesting for deriving strong inequalities for the problem using techniques from Integer Programming theory.

Chapter 4

An Exact Algorithm for PDTSPMS

In what follows, we describe the branch-and-cut algorithm used to obtain an exact solution using our proposed formulation for the Pickup and Delivery Travelling Salesman Problem with Multiple Stacks. Instead of solving a packing subproblem for every feasible Pickup and Delivery Travelling Salesman Problem tour found during the algorithm and, in case the packing is infeasible, removing this tour from the solution space, in our approach we obtain paths leading to infeasible tours concerning the capacity of each stack from the current integer solution. Hence, besides eliminating the need of solving a potentially hard packing subproblem, we eliminate from the solution space all tours containing an infeasible subpath.

Valid inequalities concerning the capacity of the stacks and valid inequalities commonly used for strengthening the PDTSP polytope are presented in Section 4.2. The separation procedures for each class of inequalities in our formulation are presented in Section 4.3. Decision choices taken during the implementation of the B&C algorithm are described in Section 4.4, followed by some general remarks in Section 4.5.

4.1 Overview

Polyhedral approaches such as B&C algorithms are among the most successful methods for solving the asymmetric TSP to optimality [Fischetti and Toth, 1997]. Moreover, considering the PDTSP, the B&C algorithm proposed by Dumitrescu et al. [2010] is capable of solving instances not solved by previous approaches. For the PDTSPMS, the first exact algorithm proposed by Côté et al. [2012a] is also a B&C algorithm.

Considering a minimization problem, *cutting planes* algorithms start with an approximation, say $\mathcal{Q}'$, for the problem polytope $\mathcal{Q}$ and add 'cuts', or inequalities that intersected with $\mathcal{Q}'$ form a better approximation for $\mathcal{Q}$. Within a *branch-and-bound* scheme, a series of increasing lower bounds are obtained by solving the relaxation of $\mathcal{Q}'$, and a series of decreasing upper bounds can be obtained through primal heuristics and the exploration of the search tree. If the lower and upper bounds coincide, then the optimal feasible solution for the polytope $\mathcal{Q}'$ is proved to be optimal for $\mathcal{Q}$. These bounds give a certificate of quality (or gap) of the current solution concerning the optimal solution. For more detailed content regarding cutting planes and general decomposition techniques, the reader is referred to Fügenschuh and Martin [2005] and Jünger et al. [1995].

One of the most important aspects in the success of a B&C algorithm is its ability to generate strong cuts, or inequalities that are among the most effective in providing a better approximation for the problem polytope and better lower and upper bounds. Describing valid inequalities that are *facet*-defining in the polytope can be a hard task. Polyhedral studies concerning the structure of the asymmetric TSP and PDTSP polytopes provide results for obtaining such inequalities. In the case of the DTSPMS, Borne et al. [2012] provided the dimension of the associated polytope and proved that every facet of the asymmetric TSP polytope also defines a facet in the former. To the best of our knowledge, no study provides such analysis of the polyhedral structure of the PDTSPMS or PDTSPL polytopes, and such endeavor is out of the reach of this dissertation.

4.2 Valid inequalities

4.2.1 Inequalities for the PDTSPMS

Valid inequalities within our formulation exploiting the particular structure of the problem were presented in Chapter 3. In particular, with the successor inequalities (3.17) we restrict the successor of a given pickup i concerning the LIFO policy, and with (3.18), (3.19) and the infeasible path inequalities (3.21) we eliminate from the solution space those tours containing subpaths leading to some infeasibility due to the capacity of the stacks.

We also include in our model an adaptation of a classical inequality used in the context of the capacited VRP. The capacity constraint of each vehicle in this problem can be imposed as follows:

$$x(S,\bar{S}) + x(\bar{S},S) \geq 2r(S) \qquad \forall S \subset P \cup D \tag{4.1}$$

Inequalities (4.1) impose that the minimum number of vehicles required to serve the customers in S, $r(S)$, enter and leave the set (S does not include any of the depots). The value of $r(S)$ can be computed by solving a *bin-packing problem* for the item set S, each of capacity q_i, $i \in S$ and bins of capacity Q. Since this problem is **NP**-Hard, a lower bound $\left\lceil \frac{|q(S)|}{Q} \right\rceil$ for $r(S)$ can be used and the inequality remains valid [Cordeau et al., 2007].

Côté et al. [2012a] provided an adaptation of those inequalities for dealing with the PDTSPMS. Rather than considering each stack individually, the authors ignore the presence of multiple stacks and take into account the whole vehicle capacity $K \times Q$, and the rounded capacity inequalities for the PDTSPMS become:

$$x(S,\bar{S}) + x(\bar{S},S) \geq 2 \left\lceil \frac{|q(S)|}{KQ} \right\rceil \qquad \forall S \subset P \cup D \tag{4.2}$$

These inequalities are not sufficient to cut all infeasible capacity tours since it is possible to have a set S for which (4.2) is not violated, but packing the items of the customers in S inside the vehicle accordingly to the LIFO policy for a given tour necessarily violates the capacity of some stack. Note that, within our formulation, inequalities (4.2) only reference variables x_{ij} that is, those variables used to model the tour ignoring the LIFO policy.

We also use another class of valid inequalities proposed by Côté et al. [2012a]. The *conflict capacity inequalities* are defined as:

$$x(i,S) + x(S) + x(S,n+i) \leq |S| \qquad \forall S \subset P \cup D,\ i, n+i \notin S,\ z(S) > Q(K-1) \tag{4.3}$$

where $z(S) = max\{q(\pi(S)\backslash S), -q(\sigma(S)\backslash S)\}$, and $\pi(S)\backslash S$ are the set of pickup locations not in S but for which the corresponding delivery is in S, and $\sigma(S)\backslash S)$ are the delivery locations not in S for which the pickup is in S (note the minus sign, as a delivery location $n+i$ has item length $-q_i$). Recall that items crossing an item i cannot be loaded on the same stack as i, thus they must fit in the remaining $(K-1)Q$ available space. Hence, if $z(S) = q(\pi(S)\backslash S) > (K-1)Q$ then it is not possible for this vehicle to load item i, then visit the set S and immediately visit $n+i$ after visiting the set S, since items in $\pi(S)\backslash S$ cross item i and, thus, these items should fill at most $(K-1)Q$ space. Observe that an item for which the pickup is in S but the delivery is outside S also crosses item i (given that i is visited immediately before S and $n+i$ immediately after S). Thus, if $z(S) = -q(\sigma(S)\backslash S) > (K-1)Q$ then those items crossing item i inside S also should fill at most $(K-1)Q$ space.

Côté et al. [2012a] also extend inequalities (4.3) to deal with a set $\mathcal{L} = \{i_1, ..., i_k\}$ of k items crossing each other, $2 \leq k \leq K-1$. The proposed inequalities have the same form as (3.8), but K is replaced with k, the number of items. Since the calculation of $z(S_{i_2}, ..., S_{i_k})$ involves computing the intersection of these sets, the authors only use values $k = 2$ and $k = 3$ in order to decrease computational time during the separation procedure. Again, even if $k = K-1$, those inequalities are not sufficient to cut all infeasible capacity tours.

We only use inequalities (4.3), that is for $k = 1$. Again, within our formulation these inequalities only refer to variables x_{ij}.

4.2.2 Inequalities for the PDTSP

Being a pickup and delivery problem, PDTSPMS may benefit from valid inequalities for PDTSP. Balas et al. [1995] introduced a set of inequalities, lifting the sub-tour elimination constraints, considering the more general problem where each vertex can have multiple successors or predecessors (not just one successor, for a pickup, or just one predecessor, for a delivery, as in the PDTSP).

$$x(S) + x(S, \bar{S} \cap \pi(S)) + x(S \cap \pi(S), \bar{S} \backslash \pi(S)) \leq |S| - 1 \qquad S \subset P \cup D \tag{4.4}$$

$$x(S) + x(\bar{S} \cap \sigma(S), S) + x(\bar{S} \backslash \sigma(S), S \cap \sigma(S)) \leq |S| - 1 \qquad S \subset P \cup D \tag{4.5}$$

where $B = \{(i, j) : i \prec j\}$ is the set of precedences that must be satisfied and $\pi(S) = \{i \in V \backslash \{0\} : (i, j) \in B$ for some $j \in S\}$ and $\sigma(S) = \{j \in V \backslash \{0\} : (i, j) \in B$ for some $i \in S\}$. In particular, if $i \prec j$ is a precedence that must be satisfied, inequalities (4.6) are also valid:

$$x(j, S) + x(S) + x(S, i) \leq |S| \qquad \forall S \subseteq V \backslash \{0, 2n+1, i, j\} \tag{4.6}$$

(4.4) and (4.5) are referred as *predecessor and successor inequalities*, respectively. Note that, for the PDTSP (and for the PDTSPMS), the set of precedences that must be satisfied is $B = \{(i, n+i) : i \in P\}$, so in (4.6) we replace j by $n + i$, and the definitions of π and σ are those described in Chapter 3. In Balas et al. [1995] it is shown that for a set $Q \subset V \backslash \{0, 2n+1, i, j\}$, if $S = Q \cup \{j\}$ then (4.4) strictly dominates (4.6). Similary, if $S = Q \cup \{i\}$ then (4.5) strictly dominates (4.6).

We also consider another set of valid inequalities introduced by Balas et al. [1995]. Let $S_1, ... S_m \subset P \cup D$ be mutually disjoint subsets such that $\sigma(S_i) \cap S_{i+1} \neq \emptyset$, $S_{m+1} = S_1$, then the *precedence cycle breaking inequalities* are valid:

$$\sum_{i=1}^{m} x(S_i) \leq \sum_{i=1}^{m} |S_i| - m - 1 \tag{4.7}$$

For instance, for $m = 2$ and $S_1 = \{i, n+j\}$ and $S_2 = \{j, n+i\}$ in a violated inequality (4.7) either a cycle occurs inside S_1 or S_2, or one of the precedences $i \prec n+i$, $j \prec n+j$ is violated.

4.3 Separation procedures

We now describe the separation procedures used to find violated constraints for each exponentially-sized class of inequalities within our formulation. For some classes, *exact* approaches are implemented, that is, if a given vector $(\mathbf{x}^*, \mathbf{y}^*)$ violates any inequality in a class, the procedure is capable of finding it. In particular, for inequalities (3.4), (3.5), (3.16) and (3.21) exact procedures are used and we can state whether an integral vector $(\mathbf{x}^*, \mathbf{y}^*)$ is a solution to our model. Valid inequalities presented in Section 4.2 are separated through heuristic procedures, that is, the procedure may terminate without finding any violation even when the solution does not satisfy an inequality in one of these classes.

Given a vector $(\mathbf{x}^*, \mathbf{y}^*) \in \mathbb{R}^{|A|} \times \mathbb{R}^{K|A|}$ with non-negative entries, the associated *support graph* G^* is the directed graph with vertices $V^* = P \cup D \cup \{0, 2n+1\}$ and arcs $A^* = \{(i,j) | x^*_{ij} > 0\}$. The exact separation algorithms that we use are those described by Cordeau et al. [2010]. They basically consist of solving *maximum flow* problems in G^* for given source and sink vertices, where the weight of an arc $(i,j) \in A^*$ is $w(i,j) = x^*_{ij}$. If the solution of the flow problem falls bellow a given value, a set $S \in V$ violating some inequality is computed using the *mincut-maxflow* theorem.

We highlight that only the values of $\mathbf{x}$ are used in the construction of the support graph for the separation of the subtour elimination constraints and the PDTSP inequalities, that is, those classes that do not consider the loading aspect of the solution. For the separation of our proposed inequalities for modelling the LIFO policy, we modify the support graph with the information provided by the vector $\mathbf{y}$.

When the vector $(\mathbf{x}^*, \mathbf{y}^*)$ is integral, rather than solving flow problems to find cuts, in this work we use a different approach and apply specialized algorithms that take advantage of this special graph structure to solve the separation problem with less effort.

4.3.1 Subtour elimination constraints

Given a non-empty subset S of $P \cup D$, the subtour inequality (3.4) for S requires that at most $|S| - 1$ arcs joining vertices in S are used, otherwise a cycle would be formed. Thus, this inequality can be written as $x(S, \bar{S}) + x(\bar{S}, S) \geq 2$, that is, the variables corresponding to arcs joining vertices in S to vertices outside S and the variables joining vertices outside S to vertices in S must sum to at least 2. On the other hand, if $0 \in S$ and $i \notin S$ for some $i \in P$, since no arc enters 0 the sum of the variables joining vertices inside S to vertices outside S (in particular i) must sum to at least 1, that is $x(S, \bar{S}) \geq 1$ (an integer path from 0 to i is illustrative; note that exactly one arc leaves S). Also, if $2n+1 \notin S$ and $n+i \in S$ for some $n+i \in D$, since no arc leaves $2n+1$ we have $x(S, \bar{S}) \geq 1$ (in an integer path from $n+i$ to

$2n+1$, exactly one arc leaves S). Now, observe that if a cycle exist for a subset $T \subset P \cup D$, then $x(S,\bar{S}) < 1$ for some S such that $0 \in S$ $i \in P, i \in T$ and $i \notin S$, or for some S such that $n+i \in T$, $n+i \in S$, $2n+1 \notin S$.

The exact separation of (3.4) consist of solving a max-flow from 0 to each pickup $i \in P$, and from each delivery $n+i \in D$ to $2n+1$. In both cases, a violated inequality is found if the flow value is smaller than 1. In the first case, the set S computed by the mincut-maxflow theorem will contain both 0 and $2n+1$ since, because of the degree equations (3.2) and (3.3), the support graph G^* will be such that every flow that leaves vertex 0 must reach vertex $2n+1$. Accordingly, instead of using S to define the cut, we use $\bar{S}$. To avoid generating the same cut for a given vector $(\mathbf{x}^*, \mathbf{y}^*)$, we do not not run the max-flow problem when the sink (resp. source) pickup (resp. delivery) location is already included in previous sets S violating a (3.4) inequality.

In the case of an integral input vector, the separation of (3.4) consist of identifying the *connected components* of G^*. In particular, observe that if G^* violates any inequality (3.4), then G^* is induced by a path from 0 to $2n+1$ and by $C_1, ..., C_l$ cycles, where each C_i, $1 \le i \le l$, defines a violated SEC. We proceed by identifying the path from 0 to $2n+1$. Then, we choose an arbitrary vertex i not included in this path and follow the path starting at i until this vertex is visited again, computing C_i and adding an inequality (3.4) for this set. We continue until all cycles are identified.

4.3.2 PDTSP inequalities

For the exact separation of inequalities (3.5) we proceed as follows. First, for a given set $S \in \mathcal{S}$, an inequality (3.5) for S requires that at most $|S|-2$ arcs joining vertices in S are used, otherwise a path from 0 to a delivery, say, $n+i$ without visiting the correspondent picku-up i would be formed, thus violating the precedence. Note that this inequality is equivalent to $x(S,\bar{S}) \ge 2$ since if a path starts at 0 and visits all vertices in S (in particular $n+i$), but visits $i \notin S$ before $n+i$, then it must leaves the set at least two times (also note that $2n+1 \notin S$). For each pickup $i \in P$ the support graph need to be modified, adding arcs $(0, n+i)$ and $(i, 2n+1)$ both with capacity 2. The max-flow from 0 to $2n+1$ is computed, if this value is smaller than 2 then a set $S \in \mathcal{S}$ that violates (3.5) is found (note that i, $2n+1 \notin S$ and $0, n+1 \in S$ by the mincut-maxflow theorem).

For integral vectors, observe that within a cycle it is not possible to define a visiting order on the vertices, hence a precedence inequality can not be defined. On the other hand, we can find violated cuts on the path from 0 to $2n+1$ since we start the tour at 0. Starting at the initial depot 0, we visit the vertices in the path until we reach a delivery $n+i$ for which the corresponding pickup i was not visited yet. The set $S = \{0, ..., n+i\}$ (that is the path from 0 to $n+i$), defines a violated inequality (3.5).

Instead of using the heuristic procedures described by Cordeau et al. [2010] for separating inequalities (4.4) and (4.5), we proceed as follows. An exact algorithm was proposed by the same authors for separating inequalities (4.6) (recall that in the PDTSP the successor of i is $n+i$). A dummy vertex $2n+2$ is inserted in G^* and connected to every other vertex $v \in P \cup D$ with arcs of capacity $w(2n+2, v) = x^*_{n+i,v} + x^*_{v,i}$. The capacity of arcs $(2n+1, i)$, $(0, i)$ and $(n+i, i)$ are set to 2. If the max-flow from $2n+2$ to i is less than 2, then a violated inequality (4.6) is found (note that we discard $2n+2$ from S and $0, i, n+i, 2n+1 \notin S$, according to the definition). Instead of adding this cut to the model, we extend the set S with $\{n+i\}$ and $\{i\}$ for defining violated inequalities (4.4) and (4.5), respectively, since both inequalities dominates (4.6).

For inequalities (4.7), Cordeau et al. [2010] derived a heuristic approach to separate them. But the lower bounds at the root node using the obtained inequalities were not good and they were not included in the final model. We include all inequalities (4.7) for $m = 2$ and sets $S_1 = \{i, n+j\}$ $S_2 = \{j, n+i\}$, for each pair $i, j \in P$ in the initial model. In our experiments, this approach proved to give overall better results than not including any of these inequalities.

4.3.3 LIFO inequalities

To separate our proposed inequalities (3.16), we devised the following heuristic algorithm. For a given pair $i, j \in P$ and a stack k, we need to check if there exist a set S such that $i \in S$, $j, n+j, n+i \notin S$, for which the vehicle loads item i on stack k, visits the customers in S and immediately delivery item j from stack k, thus violating the LIFO policy.

Recall inequalities (3.16):

$$y^k(\bar{S}, i) + x(S) + y^k(S, n+j) \le |S| \qquad j, n+j, n+i \notin S,\ i \in S,\ k \in M$$

and note that they are equivalent to:

$$x(S, \bar{S}) + \sum_{l \neq k} y^l(\bar{S}, i) + y^k(\bar{S}, n+j) + \sum_{l \neq k} y^l(S, n+j) \ge 2 \quad j, n+j, n+i \notin S,\ i \in S,\ k \in M \tag{4.8}$$

that is, if (3.16) is satisfied then the vehicle leaves set S at least two times (term $x(S, \bar{S})$), or it loads item i on a stack $l \neq k$ (term $y^l(\bar{S}, i)$), or it does not unload item j from stack k immediately after visiting the customers in S (term $y^k(\bar{S}, n+j)$) or it unloads item j immediately after visiting S but does so from a stack $l \neq k$ (term $y^l(S, n+j)$). Given a fractional solution $(\mathbf{x}^*, \mathbf{y}^*)$, we create the support graph G^* and modify it in the following way:

1. increase the capacity of arcs (i,u) by the value $y^k_{u,n+j}$, $u \in (P \cup D) \backslash \{i, n+j\}$;
2. increase the capacity of arc (i,j) by the value $\sum_{l \neq k} y^l(V,i)$;
3. set the capacity of the arcs $(j,n+j)$, $(n+j,j)$, $(n+j,n+i)$ and $(n+i,n+j)$ to 2;
4. solve the max-flow from i to j. If the flow value is smaller than 2 then, by the mincut-maxflow theorem, we can obtain a set S such that $i \in S$, $j, n+j, n+i \notin S$. Now, we need to check if for this set S the last part of the expression (4.8), $\sum_{l \neq k} y^l(S, n+j)$, does not render the inequality satisfied. If not, S defines a violated inequality (3.16).

Figure 4.1 illustrates the idea behind the modification on the support graph to obtain a violated inequality (3.16). Basically, we aggregate the values $\sum_{l \neq k} y^l(V,i)$ on x^*_{ij}, and $y^k_{u,n+j}$ on $x^*_{i,u}$, $u = 1...2n$. Then, when we run the max-flow problem, what we are computing is exactly $x(S, \bar{S})$ and this value takes into account the sum $\sum_{l \neq k} y^l(\bar{S}, i) + y^k(\bar{S}, n+j)$. Thus, if $x(S, \bar{S}) < 2$, the set S is a candidate to define a violated (4.8). But we need to check the last part of the expression, not accounted in the modification of the graph. Observe that if the set S found does not define a violated inequality, it is still possible that another set, S^*, defining a cut with capacity greater or equal $x(S, \bar{S})$ render the inequality violated, since we may have $\sum_{l \neq k} y^l(S^*, n+j) < \sum_{l \neq k} y^l(S, n+j)$.

For an integral vector solution, the separation can be performed with a simpler approach, using the values of y^k_{ij} for each arc in the solution. As in the case of separating the precedence inequalities, if the solution contains cycles, we do not try to find violated LIFO inequalities within the cycle, since we can not define a visiting order for the vertices in the cycle. Thus, we just consider the path from 0 to $2n+1$. We start at the first visited pickup, say i, loaded at some stack k for which $n+i$ is in the path and $i \prec n+i$ (note that all this information can be obtained with simply traversing the path in $O(V)$). Then, we obtain a path $i, S, n+i$, and look inside S for a pickup j loaded at stack k (that is $y^k_{lj} = 1$ for some $l \in S \cup \{i\}$) and for which $n+j \notin S$, or for a delivery $n+j \in S$ for which $j \notin S$ such that $y^k_{l,n+j} = 1$ for some $l \in S \cup \{i\}$ (that is, item j was loaded on stack k). In the first case, a violated inequality (3.16) is obtained for the subpath $S' \subseteq S$ from j to $n+i$ and, in the second case, for the subpath $S' \subseteq S$ from i to $n+j$. Recall figure 3.3 for a better view of these procedures. We proceed looking for all pickups satisfying the conditions of the above $i \in P$ in the path from 0 to $2n+1$.

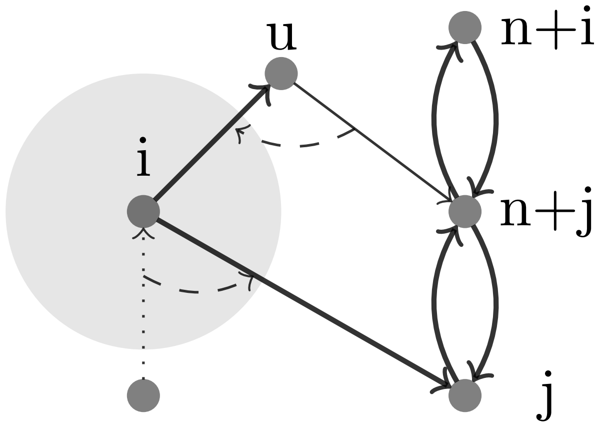


Figure 4.1: Separation procedure for inequalities (3.16). The solid thick lines represent arcs in the support graph. For $u = 1...2n$, we increase the weight of arc (i,u) by the value $y^k_{u,n+j}$ and the weight of arc (i,j) by $\sum_{l\neq k} y^l(V,i)$.

4.3.4 Capacity inequalities

Concerning the capacity inequalities, we opt not to use the heuristic procedure used by Côté et al. [2012a] for the separation of (4.2). The authors note it is relatively easy to find a set S violating such an inequality. The heuristic is based on some random parameters, and they run the procedure only five times. In our algorithm, we use the sets already computed by the other separation procedures and check if they define a violated inequality. In particular, after solving the separation procedures for the subtour elimination constraints and the inequalities (4.4) and (4.5), if the sets that have been determine do not define violated cuts, we check if they define a violated inequality (4.2). To avoid including the same cut more than once, we just add at most one inequality (4.2) for each solution $(\mathbf{x}^*, \mathbf{y}^*)$.

Inequalities (4.3), are only separated for integral solutions. While separating LIFO inequalites, if the set S in the path $i, S, n+i$ does not define any violated inequality (3.16), we check if $z(S) > Q(K-1)$. If this is the case, then the set S defines a violated inequality (4.3).

We search for violated inequalities (3.18) and (3.19) for the same sets as for inequalities (4.2). First we check if $q(\pi(S)\backslash S) > Q$ and if this is the case, we look for the vertex v maximizing $x(v,S)$ and check for each $k \in M$ whether a violated inequality (3.18) is obtained for v and S. If not, in the case of $-q(\sigma(S)\backslash S) > Q$, we perform the same verification for (3.19).

Finally, the separation of the infeasible path inequalities (3.21), when the solution

$(\mathbf{x}^*, \mathbf{y}^*)$ is integral, is done in the following way. If the path from 0 to $2n+1$ does not contain any invalid precedence $(n+i \prec i)$, we follow the path, keeping track of the residual capacity of each stack. When the vehicle loads an item on some stack k and the total length exceed Q, we add a path inequality from the last position in which stack k was empty to this last pickup performed. Observe that the LIFO policy for load and unload the vehicle are not a concern here, just the total capacity of each stack. That is, the vehicle unloads item i from stack k even when this item is not on the top of stack k.

4.4 Branch-and-Cut algorithm

We are now in a position to describe the implementation of our B&C algorithm to solve the proposed formulation for the PDTSPMS.

4.4.1 Preprocessing

Before starting the algorithm, we eliminate some variables from our model. Graph G can be reduced eliminating some of its arcs that do not appear in any feasible solution. Depot $2n+1$ cannot be the successor of a pickup location nor depot 0 can be the predecessor of a delivery location. Thus, arcs of the form $(0, j),\ j \in D$ and $(i, 2n+1),\ i \in P$ are removed from the graph, that is, we do not consider variables $x_{0,j}$ nor $x_{i,2n+1}$. Also, as location $n+i \in D$ cannot be the direct predecessor of its corresponding pickup $i \in P$, arcs $(n+i, i)\ \forall i \in P$ can also be excluded.

4.4.2 Symmetry breaking

Variables $\mathbf{y}$ introduced some degree of symmetry on our formulation. In fact, any feasible solution has a set of equivalent solutions consisting in the same tour interchanging the stack assignment of items from two or more stacks.

Note that the vehicle arrives at the final depot $2n+1$ with all stacks empty and that the predecessor of this depot is necessarily a location $d \in D$. Also, observe that the first location visited by the vehicle immediately after leaving depot 0 is necessarily a location $p \in P$ and that this load operation can be done using any stack. In order to break part of the symmetry in our model, both operations are fixed to stack 0. Constraints (4.9) and (4.10) below are added to our formulation:

$$\sum_{j\in P} y^0_{0,j} = 1 \tag{4.9}$$

$$\sum_{i\in D} y^0_{i,2n+1} = 1 \tag{4.10}$$

4.4.3 Initial Model and Cut Pool

The initial model consists in the objective function (3.1), the degree constraints (3.2) and (3.3), the symmetry constraints (4.9) and (4.10), the coupling constraints (3.13) between variables **x** and **y**, and the constraints (3.14) to ensure that an item is loaded and unloaded from the same stack. We also include all subtour elimination constraints (3.4) with $|S| = 2$ and the precedence cycle breaking inequalities (4.7) for $m = 2$ and $S_1 = \{i, n + j\}$, $S_2 = \{j, n + i\}$. Finally, we include a successor inequality (3.17) for each $i \in P$, $k \in M$.

4.4.4 Separation strategy

Initially, we drop the integrality constraints on **x** and **y**, and try to separate violated cuts by solving the max-flow problems described in previous section using the hierarchical order illustrated by Algorithm (1). In order to reduce computational time, the algorithm only tries to find violated LIFO cuts (3.16) for pairs of pickups $i, j \in P$ and stack $k \in M$ for which $y^k(V,i) > 0.5$ and $y^k(V, j) > 0.5$ that is, for pairs of pickups having more than 50% of the associated items loaded on the same stack k. Also, note that when a violated inequality (3.16) is found for $i, j \in P$ $k \in M$ it is valid for all $k' \in K\setminus\{k\}$, even if they are not violated. Accordingly, if a violated cut is found for a given stack, an inequality is added $\forall k \in M$.

When no violated cut is found, we proceed to the branch-and-cut phase, where the root node is given by the model returned by Algorithm 1, with solution $(\mathbf{x}^*, \mathbf{y}^*)$. At each other node of the branch-and-bound tree, if the associated solution is fractional, the separation is performed as in Algorithm 1, but we do not separate the LIFO inequalities (3.16). In our tests, separating these inequalities at fractional nodes resulted in many violated cuts and a large model to solve. This approach performed better from a computational point of view, giving better running times. Also, the cuts are added globally that is, all inequalities are valid for every node in the tree.

For a node with integral solution, the separation routines are implemented taking advantage of the special structure of the support graph. In particular, if no violated inequality (3.4), (3.5) and (3.16) are found, we only need to check if the capacity of each stack is not exceeded to have a feasible solution for the PDTSPMS. In the strategy pre-

Algorithm 1 Computes a LP-solution ($\mathbf{x}^*$,$\mathbf{y}^*$)

.

```
1: Construct the support graph $G^*$
2: $T \leftarrow \emptyset$
3: for all $i \in P \setminus T$ do
4:     Run the max-flow from 0 to $i$ and obtain the mincut set $S$
5:     if $flow < 1$ then
6:         A violated cut (3.4) for $\bar{S}$ has been found, add it to the model.
7:         $T \leftarrow T \cup S$
8:     else {if no inequality (4.2), (3.18), (3.19) were added for $\mathbf{x}^*$,$\mathbf{y}^*$.}
9:         Check if $S$ defines violated inequalities (4.2), (3.18), (3.19) and add then to the model.
10:    end if
11: end for
12: $T \leftarrow \emptyset$
13: for all $i \in D \setminus T$ do
14:    Run the max-flow from $i$ to $2n+1$ and obtain the mincut set $S$.
15:    if $flow < 1$ then
16:        A violated cut (3.4) for $S$ has been found, add it to the model.
17:        $T \leftarrow T \cup S$
18:    else {if no inequality (4.2), (3.18), (3.19) were added for $\mathbf{x}^*$,$\mathbf{y}^*$.}
19:        Check if $S$ defines violated inequalities (4.2), (3.18), (3.19) and add then to the model.
20:    end if
21: end for

22: for all $i \in P$ do
23:    Modify $G^*$ accordingly to subsection 4.3.2 and run the maxflow from 0 to $2n+1$. Obtain the mincut set $S$.
24:    if $flow < 2$ then
25:        A violated inequality (3.5) has been found, add it to the model.
26:    end if
27:    Modify $G^*$ accordingly to subsection 4.3.2 and run the maxflow from $2n+2$ to $i$. Obtain the mincut set $S$.
28:    if $flow < 2$ then
29:        A violated inequality (4.6) has been found.
30:        Violated (4.4) and (4.5) are defined for $S = S \cup \{n+i\}$ and $S = S \cup \{i\}$, respectively. Add then to the model.
31:    else {if no inequality (4.2) were added so far.}
32:        Check if $S$ defines a violated inequality (4.2), and add it to the model.
33:    end if
34: end for

35: for all $i \in P$ do
36:    for all $j \in P \setminus \{i\}$ do
37:        for all $k \in M$ do
38:            if $y^k(V,i) > 0.5$ and $y^k(V,j) > 0.5$ then
39:                Modify $G^*$ accordingly to subsection 4.3.3 and run the maxflow from $i$ to $j$. Obtain the mincut set $S$.
40:                if $flow + \sum_{l \neq k} y^l(S, n+j) < 2$ then
41:                    A violated inequality (3.16) has been found.
42:                    Add an inequality (3.16) $\forall k \in M$ for the given $S, i, j$.
43:                end if
44:            end if
45:        end for
46:    end for
47: end for
48: if No violated cuts were found then
49:    Terminate the LP-solution.
50: else
51:    Solve the model again with the new inequalities found and obtain ($\mathbf{x}^*$,$\mathbf{y}^*$).
52:    Go back to 1.
53: end if
```

sented by Côté et al. [2012a], this is done by solving a packing problem for the tour (not violating any (3.4) and (3.5)). In the case of a feasible packing solution, the tour is a feasible solution for the PDTSPMS. If not, this tour is eliminated from the solution space with a path inequality (3.20). In our formulation, we can simulate the load and unload operations performed by the vehicle using the variables y_{ij}^k, and check whether the tour violates the capacity of some stack. For each violation found, a path inequality (3.21) is separated, thus eliminating all tours containing this path.

The separation strategy for an integral solution is depicted in Algorithm 2. Recall that the support graph is induced by a path $W = 0, ..., 2n+1$, and by cycles $C_1, ..., C_l$, one for each violated SEC (3.4).

4.4.5 Implementation details

Our algorithm uses CPLEX 12.5 as a B&C framework through the Concert API Library. All preprocessing, heuristics and automatic cut generation of the solver are turned off. The callback mechanism is used for the separation of user cuts and lazy constraints. We give preference to variables x_{ij} over variables y_{ij}^k during branching. In doing this, our aim is to first obtain a tour and then separate violated LIFO and capacity inequalities. The node selection, variable and direction for branching are chosen using CPLEX's default values. As we consider instances with integer-valued costs, we set the absolute MIP gap tolerance and the absolute objective difference cut-off parameters of CPLEX to 0.9999.

For the cut separation solving maximum flows problems, we used an implementation of Dinic's algorithm [Dinitz, 2006] with worst-case complexity of $O(|V^*|^2|A^*|)$. We also tested an implementation of the highest-label preflow-push algorithm of Goldberg and Tarjan [1986] available in the open-source Library for Efficient Modelling and Optimization in Networks (LEMON) [Dezs et al., 2011] with worst-case running time $O(|V^*|^2\sqrt{|A^*|})$. An advantage of push-relabel algorithms is that in applications where just the flow value and the minimum cut set is required (as is the case in the separation procedures), the implementation can be adjusted to run slightly faster, running just the first phase of the algorithm.

We performed a set of tests for assessing the empirical performance of both implementations. Six batches of 100 support graphs $G^*(V^*, A^*)$, with $|V^*|$ ranging from 28 to 44 and $|A^*|$ from 27 to 103, were submitted to each algorithm and the total time spent for solving each batch (obtaining the flow and the minimum cut set for each problem) recorded. Empirically, our implementation of Dinic's algorithm performed better for those instances. The data structures used in LEMON are quite more elaborated than the graph data structures we use in our algorithm. The running times can be explained by

Algorithm 2 Separation strategy for an integral solution $(\mathbf{x}^*, \mathbf{y}^*)$

```
•
 1: Construct the support graph $G^*$
 2: Find the path $W = 0, ..., 2n+1$ and the connected components $C_1, ..., C_l \subset P \cup D$. $\{W = w_0 = 0, w_1, ..., w_r = 2n+1\}$

 3: for all $C_i$, $1 \leq i \leq l$ do
 4:     Include a SEC (3.4) for $C_i$.
 5: end for

 6: $S \leftarrow \{0\}$, $i \leftarrow 1$.
 7: while $w_i \neq 2n+1$ do
 8:     $S \leftarrow S \cup \{w_i\}$.
 9:     if $w_i \in D$ and $\pi(\{w_i\}) \notin W$ then
10:         A violated inequality (3.5) has been found for $S$.
11:     end if
12:     $i \leftarrow i+1$.
13: end while

14: for all $i \in P : i,\ n+i \in W$ do
15:     Find de subpath $i, S, n+i$ in $W$.
16:     if $y^k_{lj} = 1,\ j \in P, n+j \notin W,\ l \in S \cup \{i\}$ or $y^k_{l,n+j} = 1, n+j \in D, j \notin W,\ l \in S \cup \{i\}$ then
17:         A violated inequality (3.16) has been found.(with the roles of $i$ and $j$ interchanged accordingly with the case).
18:     else {no $j \in P$ or $n+j \in D$ could be found.}
19:         if $q(S) > (K-1)Q$ then
20:             A violated inequality (4.3) has been found for $S$ and $i, n+i$.
21:         end if
22:     end if
23: end for

24: if No violated inequality (3.5) was found then
25:     $load[0], ..., load[K-1] \leftarrow 0$.
26:     $start[0], ..., start[K-1] \leftarrow 0$.
27:     $i \leftarrow 1$.
28:     while $w_i \neq 2n+1$ do
29:         Let $k \in M$ such that $y^k_{w_{i-1} w_i} = 1$.
30:         if $w_i \in P$and $load[k] = 0$ then
31:             $start[k] \leftarrow i-1$.
32:         end if
33:         $load[k] \leftarrow load[k] + q_i$.
34:         if $load[k] > Q$ then
35:             An infeasible inequality for the path starting at $w_{start[k]}$ ending at $w_i$ has been found. Add (3.21) for each $k \in M$.
36:         end if
37:         $i \leftarrow i+1$.
38:     end while
39: end if

40: if No violated cut was found then
41:     print $(\mathbf{x}, \mathbf{y})$ is a feasible solution for PDTSPMS.
42: end if
```

the burden incurred in creating such structures. For an overview of the computational aspects of various maximum flow algorithms, the reader is referred to Ahuja et al. [1997].

4.5 Final remarks

In this chapter, valid inequalities to strengthen the PDTSP polytope and valid inequalities for PDTSPMS concerning the capacity of the vehicle were presented. The B&C algorithm relies on separation procedures for the exponentially-sized classes of inequalities based on max-flow problems, when the solution is fractional, and on specialized algorithms for integral solutions. In particular, for the proposed LIFO inequalities, we devised a heuristic procedure when the solution is fractional and an exact procedure for integral solutions.

Some implementation decisions were made in order to improve the running time of the B&C algorithm. For example, in the case that a separation procedure is able to find more than one violated inequality, a wide range of strategies to decide which ones to include in the model can be adopted (the first cut found, all violated inequalities, the mostly violated among all, *orthogonal* cuts). In our experiments, the trade-off between the time for computing all violated cuts for a given solution, and then deciding which ones to include and the impact that the selected cuts had on the overall performance of the algorithm did not endorse such approaches. Hence, we use a quite straightforward strategy as illustrated by Algorithms 1 and 2.

Another implementation issue concerns the size of the support graph G^* during the separation procedures. Shrink algorithms [Padberg and Rinaldi, 1990] could be used to decrease computational time on separation procedures using the max-flow algorithm. Given the relative small size of the instances considered in this work (from 24 to 44 nodes), we opt to not use such an approach.

Chapter 5

Computational Results

This chapter presents an experimental evaluation of the proposed B&C algorithm for solving the ILP formulation for the Pickup and Delivery Travelling Salesman Problem with Multiple Stacks. Section 5.1 describes the computational scenario for running the experiments and the data sets for benchmark instances. We investigate the impact of the proposed valid inequalities on the performance of the algorithm in Section 5.2. Next, we present a comparison with the results available in the literature for the B&C algorithm by Côté et al. [2012a]. Finally, in Section 5.4, we analyse the results and assess the weaknesses and advantages of the proposed algorithm.

5.1 Overview

All algorithms described in the previous chapter were implemented in C++ and compiled using version 4.7.3 of the g++ compiler with -O3 flag activated. The tests were run on an 2.2GHz Intel Xeon E5520 processor with 16GB of RAM running Linux.

The benchmark data set used in our experiments is the set of instances introduced by Côté et al. [2012a]. The authors generated the instances based on Pickup and Delivery Travelling Salesman Problem with LIFO Loading benchmark instances in Cordeau et al. [2010] and Carrabs et al. [2007a,b]. The clients distance matrix of each instance is taken from one of the following TSP instances from the TSPLIB [Reinelt, 1991]: *a280, att532, brd14051, d15112, d18512, fnl4461, nrw1379, pr1002, ts225.* In each file, the location of the first city is defined as the initial (location 0) and final depots (location $2n+1$). The smallest instance consist of the first 23 cities and form the base for constructing the larger instances. For the assignment of the vertices as pick-up or delivery locations, the 22 subsequent cities are paired randomly, obtaining the $n = 11$ requests. The larger instances ($n = 13, 15, 17, 19, 21$) are built sequentially by performing a random pairing between the

	C1 class		C2 class	
Instance	K	Q	K	Q
a280	2	2	2	12
att532	3	3	2	15
brd14051	2	2	3	11
d15112	3	2	3	10
d18512	2	3	2	14
fnl4461	4	1	3	10
nrw1379	3	2	4	10
pr1002	2	2	3	13
ts225	2	2	2	12

Table 5.1: Benchmark instances for both classes.

new locations added in each step to form the extra pickup and delivery assignments. This procedure is outlined in Carrabs et al. [2007a] in order to ensure that the solution cost of a larger instance is always at least as large as the cost of a smaller instance from the same file.

In that way, 54 instances were generated and divided in nine PDTSP base instances each with six values for the number of requests, namely, $n = 11, 13, 15, 17, 19, 21$ (resp. $24, 28, 32, 36, 40, 44$ locations). Now, two classes of instances for the PDTSPMS are generated. In the first class, C1, the length of each pickup, q_i, is 1, the number of stacks, K, is a random number between 2 and 4, and the capacity, Q, of each stack is a random number between 1 and 3. In the second class, C2, the length of items is a random number between 1 and 10, the number of stacks is a random number between 2 and 4, and the capacity is a random number between 10 and 15. Those values are justified by Côté et al. [2012a] in order to obtain difficult instances for which the optimal solution of the PDTSP for an instance is not an optimal solution to the PDTSPMS. Table 5.1 report the number of stacks (K) and the capacity (Q) used to generate each instance within each of the nine TSPLIB files.

After reading the location of each city in the TSPLIB file, the cost associated with each arc, c_{ij}, is rounded to the nearest integer. We use an initial upper bound (UB) on the optimal value of each instance provided by the Large Neighbourhood Search heuristic in Côté et al. [2012b].

Model	Solved	Root	Gap	Time	Cuts	Path
Plain	31	5.40%	0.66%	491.2	967.3	610.7
(3.17)	32	5.19%	0.24%	436.8	782.3	539.4
(3.18,3.19)	32	5.14%	0.20%	435.2	895.9	547.6
(3.17,3.18,3.19)	32	5.19%	0.18%	401.0	821.3	546.3

Table 5.2: Model comparison for C1 instances.

Model	Solved	Root	Gap	Time	Cuts	Path
Plain	24	5.31%	0.00%	151.9	570.7	696
(3.17)	24	5.27%	0.00%	183.6	548.3	729
(3.18,3.19)	23	5.31%	0.10%	273.7	626.6	688.8
(3.17,3.18,3.19)	24	5.28%	0.00%	238.8	543.4	556.6

Table 5.3: Model comparison for C2 instances.

5.2 Effectiveness of valid inequalities

In Tables 5.2 and 5.3, we evaluate the contributions of the proposed successor inequalities (3.17), and the capacity subtour elimination constraints (3.18, 3.19) on tackling the set of of 34 (resp. 24) solved instances on class C1 (resp. C2) by the B&C algorithm in Côté et al. [2012a]. For these experiments, we also set the maximum computation time to one hour. In the row Plain, we report the results obtained with our B&C algorithm considering the initial model described in Section 4.4 without including inequalities (3.17) and not checking for violated inequalities (3.18, 3.19) during the execution of Algorithm 1. In the subsequent rows, we add one of these two families of inequalities to the Plain formulation and, in the last row, we add the two families. For each model, we give the number of solved instances (Solved), the average root node lower bound (Root) computed as 100*(UB-LB)/UB, the average gap (Gap) between the best lower bound and UB (for unsolved instances), the average CPU time (Time), the average number of cuts of type (3.16) (LIFO cuts), (4.2), (4.3), (3.18) and (3.19) (capacity cuts) added through the execution of the algorithm. Finally, in column Path, we report the number of path inequalities (3.21) generated.

We observe that some classes of inequalities (the precedence and successor inequalities (4.4, 4.5), the rounded capacity inequalities (4.2), the LIFO inequalities (3.16) and the CSEC (3.18, 3.19)) are not separated exactly during the execution of Algorithm 1, used to obtain the root lower bound. Thus, although we include all inequalities (3.17), it is possible that different violated cuts are found for each model. This explains the small deterioration on the root gap values on the last row of each table, that is, the root gap is slightly greater using the two classes than when using just one of the classes.

For class C1, two instances could not be solved (*d18512* for $n = 15$ and *fnl4461* for $n = 17$) by any of models and the Plain model could neither solve the instance (*ts225* for $n = 17$). The root gap and the gap on unsolved instances could be slightly improved with the addition of the inequalities. Also, the number of added cuts to impose the LIFO policy and assignments that do not violate the capacity of each stack (Cuts) is reduced with the inclusion of the proposed inequalities.

In the context of C2 instances, only one instance (*ts225* for $n = 13$) could not be solved by the model including the CSEC inequalities (3.18, 3.19). The root gap could also be improved with the addition of the inequalities, but the contributions were smaller than for C1 instances. Nevertheless, the number of added Cuts and the path inequalities could also be reduced. On the other hand, the CPU time was increased with the inclusion of the inequalities, mostly likely due to the procedure of identifying cuts of type (3.18, 3.19).

The small contribution of the proposed inequalities on the root node lower bound can be explained by the fact that these inequalities may only contribute to the packing of the items, without necessarily changing the tour. For example, on C1 instances *att532* for $n = 11, 13, 15, 17$, *nrw1379* for $n = 11$ and *ts225* for $n = 11, 13$, the optimal PDTSP tour is feasible concerning the LIFO policy, that is, there is a feasible packing for this tour satisfying both the loading/unloading operations and the capacity of each stack. Thus, the impact of the proposed inequalities is more significant towards the assignment of the items in the stacks and not on the modification of the tour.

In the following reports, the model with our proposed inequalities (3.17) and (3.18, 3.19) will be considered.

5.3 Comparison with results from literature

Next, we report a comparative analysis of the results obtained in this work in comparison with those obtained by the B&C by Côté et al. [2012a] on the PDTSPMS instances. We note that the authors performed their tests using a 2.2GHz AMD Opteron 275 processor. Also, the maximum computation time was set to one hour. Accordingly, in running our experiments, we also set this time limit. The initial upper bound (UB) on both algorithms are those provided by the LNS heuristic in Côté et al. [2012b].

In Tables[1] 5.4 and 5.6 we summarize the overall results obtained by both approaches for each class of instances, C1 and C2, respectively. In each table, we report the number

[1]The reported values for the average final gap on unsolved instances (Gap) in a class were computed based on the detailed data available in Côté et al. [2012a]. The Gap value for each class presented in this latter work did not match the detailed results reported for each instance in a class.

Method	Solved	Root	Gap	Time	Path
Côté et al. [2012a] B&C	34	6.1%	6.00%	381.9-1573.8	8.3
Our B&C	33	9.18%	11.2%	212.5-1529.9	506.7

Table 5.4: Overall results on C1 instances.

	Côté et al. B&C				**Our B&C**			
Instance	Solved	Root	Gap	Time	Solved	Root	Gap	Time
a280	5	6.20	10.35	499.5	5	11.91	6.19	239.4
att532	6	2.00	–	342.4	6	2.16	–	36.4
brd14051	2	6.15	4.29	19.5	2	16,79	15,76	12,6
d15112	4	7.73	7.54	834.6	4	9,16	8,85	220.5
d18512	3	5.71	6.12	2230.2	2	6.65	4.47	1.05
fnl4461	4	6.22	3.29	30.4	3	11.28	15.57	1004.3
nrw1379	1	10.92	8.46	0.9	1	11.51	9.20	0.5
pr1002	5	3.98	2.31	459.6	6	5.19	–	187.8
ts225	4	5.76	3.92	13.1	4	7.98	3.35	137.2

Table 5.5: Overall results for each instance in class C1.

Method	Solved	Root	Gap	Time	Path
Côté et al. [2012a] B&C	24	7.8%	6.6%	312.2-2138.8	186.5
Our B&C	25	9.8%	10.2%	224.2-1988.6	460.7

Table 5.6: Overall results on C2 instances.

of instances solved (Solved) over the total of 54, the average root node gap (Root), that is, the relative value between the objective value achieved before starting the branching phase of the algorithm (lower bound , LB) and the optimal value (the final UB, in the case of unsolved instances) computed as 100*(UB-LB)/UB. Next, we report the average final dual gap (Gap) on unsolved instances. Finally, we show the average CPU time in seconds (Time), considering just the solved instances and considering all instances, and the average number of path inequalities (3.20) generated (Path), one for each infeasible packing subproblem solved by the algorithm of Côté et al. [2012a] and, in the case of our approach, the average number of violated inequalities (3.21) found for integer solutions.

Tables 5.5 and 5.7 report the overall results obtained on each of the nines base TSPLIB instances considered, for class C1 and C2, respectively. (The column Time report the average time on the solved instances). In Appendix A, we show the detailed results considering each particular instance in both classes.

	Côté et al. B&C				**Our B&C**			
Instance	Solved	Root	Gap	Time	Solved	Root	Gap	Time
a280	4	8.32	9.81	400.9	5	13.11	4.63	195.0
att532	2	7.93	7.00	2.5	2	8,55	6,16	0.5
brd14051	2	6.64	4.60	482.2	2	14,85	17,69	85.0
d15112	3	8.23	5.06	620.0	3	8.49	3.96	324.3
d18512	1	7.23	5.56	70.6	1	12.59	9.74	27.6
fnl4461	3	5.87	6.74	17.3	3	4,21	4,67	32.3
nrw1379	1	9.78	7.05	1.3	1	10.98	8.59	0.5
pr1002	6	3.17	–	324.9	6	2.97	–	60.3
ts225	2	12.77	8.64	493.1	2	17,6	6,74	1425.8

Table 5.7: Overall results for each instance in class C2.

5.3.1 C1 instances

For class C1, our algorithm could solve 33 out of 54 instances, while 34 are solved by the B&C algorithm by Côté et al. [2012a]. Our algorithm was not capable of solving two instances solved by the latter (*d18512* for $n = 15$ and *fnl4461* for $n = 17$), but on the other hand, we could provide a new certificate of optimality for an instance not solved before (*pr1002* for $n = 21$). Except for these cases, the set of solved instances for class C1 are the same. Although on average the root gap values obtained with the proposed algorithm are higher, for those sets with the same number of solved instances we could achieve better execution times and/or smaller values for the final gap on unsolved instances. In particular, for the benchmark instance *att532*, our algorithm outperforms the previous results especially in the larger-sized instance ($n = 21$). The root node gap value is slightly smaller (3.53% against 3.60%) but the time to solve this instance is reduced by one order of magnitude. We note that, for *att532* in C1, the optimal PDTSP tour is also an optimal PDTSPMS tour for $n = 11, 13, 15, 17, 19$, and after the addition of few LIFO and capacity inequalities, a feasible assignment of the items in each stack is obtained. On the other hand, for $n = 21$, the optimal PDTSP tour needs to be slightly modified to allow the correct sequence of loads and unloads of the items. Our algorithm achieves this modification after the addition of less cuts (769 against 3110), while the algorithm by Côté et al. [2012a] needs to solve at least 32 packing subproblems to cut feasible PDTSP tours rendering an infeasible packing of the items from the solution space.

We highlight the inability of our proposed algorithm in solving instances *brd14051* and *fnl4461*. For the former, while the time to tackle the two solved instances ($n = 11, 13$) was smaller than the time for solving these same instances with the B&C algorithm by Côté et al. [2012a], the final gap on unsolved instances were larger, especially for the instances with more requests ($n = 19,21$). In the set *fnl4461*, we could not solve three instances ($n = 17,19,21$) and the time to tackle the solved ones were also higher. For instances in this set, the vehicle contains $K = 4$ stacks and capacity, $Q = 1$. Our algorithm generates a larger number of LIFO and capacity inequalities for instances in this set.

Few inequalities (3.20) are generated by the packing subproblem described by Côté et al. [2012a] for instances in C1, meaning that when a feasible PDTSP tour is found by their algorithm the packing subproblem is unlikely to be infeasible. Observe that the average time for tackling all instances is essentially the same for the two algorithms ($1573.8 - 1529.9$).

5.3.2 C2 instances

In the context of instances in class C2, our algorithm was capable of solving instance *a280* for $n = 19$ requests, not previously solved. All other instances solved before (24) were also solved. Values for the average root gap were higher in our algorithm, but the difference is smaller than for C1 instances, and for the sets *fnl4461* and *pr1002* these values were smaller. Moreover, for the sets with the same number of solved instances, the execution time and/or the final gap on unsolved instances were improved with our algorithm.

As for class C1, our algorithm performs comparatively worst on the *brd14051* and *fnl4461* sets. Although the time to solve the two instances in *brd14051* ($n = 11,13$) is much smaller (85-482.2secs), the final gap for the larger unsolved instances was too high. For the set *fnl4461* better root lower bound and final gaps on unsolved instances were obtained using our algorithm, but the time to solve the instances were almost doubled. Again, this set contains the instances where the vehicle contains the tightest stack capacity ($Q = 10$).

Class C2 contains the instances in which more path inequalities (3.20) are generated with the packing procedure by Côté et al. [2012a]. This may explain the difference on the total execution time of both algorithms. We highlight the difference for solving instance *brd14051* with $n = 13$ requests. While the B&C by Côté et al. [2012a] took 960.4 seconds, solving at least 1820 packing subproblems and generating 5276 LIFO and capacity cuts, our algorithm took 168.2 seconds, generating only 308 cuts and 297 path inequalities (3.21).

5.3.3 Analysis

It is worth to mention the impact that the number of stacks, the capacity and the size of each item have on the tractability of solving a particular instance. For example, while all instances in set *att532* are solved for class C1, for class C2 only the cases $n = 11,13$ are solved to optimality. On the other hand, both algorithms take more time to solve instances in the set *pr1002* for class C1 than C2. Also, the B&C by Côté et al. [2012a] could not solve the instance *pr1002* with $n = 21$ requests in class C1, while all instances in class C2 are solved. Finally, observe that the cost of optimal tours for *pr1002* in class C2 are smaller than in C1 (our algorithm proved the optimal value of 20150 for $n = 21$), meaning that packing the not all unitary items inside this vehicle for feasible PDTSP tours is easier. In that sense, problem difficulty is largely dependent on the characteristics of the vehicle (K and Q) and on the size of the items to be transported.

Nevertheless, as noted by Côté et al. [2012a], it seems that if solving the PDTSP is hard, the same applies to find an optimal PDTSPMS tour. Only the smaller instances are solved for the sets *brd14051* ($n = 11, 13$) and *nrw1379* ($n = 11$) for both classes. For *nrw1379*, while it is easy to solve instances with $n = 11$ requests, for the larger instances none of the algorithms could not reduce the final gap below 3% .

Concerning the performance of both algorithms on the set of benchmark instances, our proposed B&C generates a search tree with more nodes. This is due to the extra variables $\mathbf{y} \in \mathbb{B}^{|A|K}$. Despite this fact, we could achieve better execution times on several instances for both classes. We emphasize that with the addition of the extra set of variables $\mathbf{y} \in \mathbb{B}^{|A|K}$, our B&C algorithm can find violations of the LIFO policy and on the capacity of each stack without the need of solving a hard packing subproblem. Moreover, our algorithm generates less LIFO and capacity inequalities in solving the majority of instances.

5.4 Conclusions

This chapter presented a computational experience with B&C algorithms for solving the PDTSPMS exactly. Our algorithm relies on a new formulation for the problem, adding a new set o variables to better describe the LIFO and the capacity inequalities.

Besides the classical arc inclusion variables x_{ij}, we consider an extra set of binary variables y_{ij}^k that model the loading and unloading operations on each stack. While the approach in the literature only considers the arc variables and postpone the assignment of each item to a stack when a feasible PDTSP tour is found, in our approach we use the new set of variables to simultaneously impose constraints on the tour and on the packing of the items inside the vehicle. We reported computational results that support this approach.

We could obtain two new optimality certificates. Also, we could improve the dual gap for several unsolved instances. The gap values obtained at the root node were worst for our algorithm on larger instances for which the PDTSP problem is hard to solve. This can be explained by the fact that we do not separate violated cuts for one class of valid inequalities for the PDTSP separated in the B&C algorithm by Côté et al. [2012a]. In future work, the impact of adding such inequalities should be investigated.

Chapter 6

Conclusion and Future Work

This dissertation addressed the exact solution of the Pickup and Delivery Travelling Salesman Problem with Multiple Stacks. This is a difficult combinatorial problem that arises from the combination of routing and loading aspects on the Vehicle Routing Problem.

For concluding the work, this final chapter assesses the contributions made by this dissertation. Next, we discuss future work and research possibilities which could further contribute to the developments of exact algorithms to solve the PDTSPMS.

6.1 Contributions

The main contributions of this dissertation were the introduction of a new ILP formulation for the PDTSPMS and a B&C algorithm for solving this formulation. The algorithm uses some valid inequalities from previous work, but new valid inequalities within the new formulation were also proposed. We have applied this algorithm on a set of benchmark instances from the literature. Our algorithm was able to solve to optimality two instances not solved before. Also, the final dual gap for some unsolved instances could be improved.

The first exact algorithm proposed by Côté et al. [2012a] relies on solving a packing subproblem for the assignment of the items to a stack for a given tour. As an attempt to avoid the need of solving such a difficult problem, we introduce a new set of variables to better model the loading and unloading operations performed by the vehicle. In doing so, our aim was to couple the routing and loading aspects and derive procedures that could facilitate the search for violations on the loading and unloading policy of the stacks.

Our computational experiments supported the new approach. Not only could we obtain new optimality certificates, but we could also solve several instances with less computational effort. The results also point out that more attention should be paid to the separation of valid inequalities for the Pickup and Delivery Travelling Salesman Problem, as for some cases the root lower bound obtained with our algorithm is still very low, especially for the larger instances for which the associated PDTSP is difficult to solve.

The work is also of interest since it provided a new approach for solving the PDTSPMS, confirming the results of the only one exact algorithm for the problem [Côté et al., 2012a].

6.2 Further work

One source of weakness in this work is that in looking for cuts in some classes, instead of investigating the structure of the current solution and try to construct a set $S \subset P \cup V$ defining a violated inequality, we use the sets computed on separation procedures of other classes. Despite the fact that we could find several violated cuts using this approach, it would be interesting to investigate the use of specialized heuristic separation procedures for these classes.

Another possible investigation concerns the use of our proposed B&C algorithm to solve the Double Travelling Salesman Problem with Multiple Stacks instances as Côté et al. [2012a] have done. We plan to adapt our algorithm and evaluate its performance on this problem.

Finally, polyhedral approaches like B&C algorithms would benefit from a better understanding of the polyhedral structure of the problem. Findings towards this direction could provide new insights on the development of stronger formulations and valid inequalities that may leverage the efficiency of such approaches.

Appendix A

Detailed results

This appendix reports the detailed results for each benchmark instance proposed by Côté et al. [2012a]. Recall that the base for each instance consist of the TSPLIB file given in column Instance, where the assignment of the pickup and deliver locations is performed as described in Section 5.1. Thus, we have 54 base PDTSP instances divided in nine sets and six values for the number of requisitions n. Using these instances, two classes of PDTSPMS instances are generated: in class C1 the items have unitary size and, in class C2, items have sizes randomly chosen from the set $\{1, 2, ..., 10\}$.

In the following tables, the column Instance report the details of each instance. UB is the initial upper bound provided by the heuristic of Côté et al. [2012b]. Opt is the optimal value. The columns Root, Gap are the same as described in Section 5.1, and Time is the CPU time in seconds. Column Cuts reports the number of LIFO cuts (3.16), and capacity cuts (3.18), (3.19), (4.2) and (4.3) added by our proposed B&C algorithm, and, in the case of the algorithm of Côté et al. [2012a], the number of LIFO cuts (3.8) and capacity cuts (4.2) and (4.3). Column Nodes reports the number of nodes in the search tree.

For each one of the six instances within a TSPLIB file, the best solution is displayed in bold. The quality of a solution is measured accordingly with the execution time to solve the instance, or the final gap for unsolved instances. Tables A.1, A.2 and A.3 refer to instances in class C1. Tables A.4, A.5 and A.6 refer to instances in class C2.

Instance				Proposed B&C						Côté B&C					
name	n	UB	Opt	Root	Gap	Time	Cuts	Path	Nodes	Root	Gap	Time	Cuts	Path	Nodes
a280	11	449	449	10,58		5,9	483	159	1573	**3,79**		**2,0**	**192**	**0**	**42**
	13	488	477	10,47		19,4	722	229	1611	**3,46**		**6,9**	**168**	**0**	**41**
	15	613	542	13,28		607,6	1665	845	11226	**4,59**		**105,1**	**679**	**0**	**320**
	17	633	624	10,9		**93,3**	**694**	**285**	**3813**	5,54		181,3	1139	0	631
	19	709	669	10,76		**470,9**	**947**	**447**	**9813**	5,81		2202,1	3434	0	1982
	21	773	n.a	**15,48**	**6,19**	**3600**	**2324**	**1811**	**29926**	13,98	10,35	3600	3702	0	1923
att532	11	4177	4177	**1,02**		**0,1**	**17**	**0**	**3**	0,93		0,2	10	0	2
	13	4937	4937	**1,18**		**0,2**	**18**	**0**	**3**	1,31		0,7	7	0	6
	15	5151	5151	**2,86**		**0,7**	**35**	**3**	**24**	2,30		2,9	24	0	17
	17	5294	5294	**1,91**		**0,8**	**77**	**30**	**19**	1,51		2,8	19	0	10
	19	5587	5587	**2,48**		**2,6**	**101**	**54**	**102**	2,35		13,8	30	0	25
	21	9266	9266	**3,53**		**214,0**	**769**	**1725**	**13552**	3,60		2034,2	3110	32	4020
brd14051	11	4396	4396	**3,74**		**2,5**	**133**	**41**	**207**	3,69		8,7	314	0	84
	13	4439	4439	**4,48**		**22,7**	**185**	**26**	**1672**	4,46		30,4	642	3	164
	15	4809	n.a.	10,4	4,79	3600	443	15	30639	**8,38**	**3,41**	**3600**	**31596**	**0**	**4894**
	17	4945	n.a.	12,46	9,57	3600	500	17	22004	**6,87**	**4,65**	**3600**	**20525**	**0**	**3267**
	19	6704	n.a.	33,88	32,56	3600	799	42	14738	**8,87**	**4,72**	**3600**	**9645**	**0**	**1905**
	21	6923	n.a.	35,81	16,15	3600	908	20	8289	**4,60**	**4,40**	**3600**	**7024**	**0**	**1305**

Table A.1: Detailed results for C1 instances.

Instance				Proposed B&C						Côté et al. B&C					
name	n	UB	Opt	Root	Gap	Time	Cuts	Path	Nodes	Root	Gap	Time	Cuts	Path	Nodes
d15112	11	74603	74603	**4,79**		**0,9**	**155**	**24**	**163**	6,32		2,9	98	2	66
	13	80690	80690	**7,85**		**14,8**	**998**	**308**	**2063**	6,89		37,0	454	0	265
	15	89754	89754	**5,96**		**13,7**	**499**	**223**	**2119**	4,82		20,6	200	8	122
	17	96804	96804	**9,51**		**852,6**	**2379**	**616**	**23777**	6,51		3278	8569	66	5402
	19	103609	n.a.	11,42	5,67	3600	3074	494	22365	**8,86**	**5,39**	**3600**	**5110**	**1**	**2630**
	21	109048	n.a.	15,43	12,03	3600	1244	142	9228	**12,97**	**9,68**	**3600**	**3875**	**0**	**1390**
d18512	11	4280	4280	**1,02**		**0,4**	**61**	**30**	**56**	1,03		1,4	46	0	8
	13	4301	4301	**1,2**		**1,7**	**324**	**57**	**262**	1,30		5,7	103	0	32
	15	4638	4638	6,99	1,94	3600	5370	1537	69588	**5,84**		**2574,1**	**15090**	**331**	**7901**
	17	4741	n.a.	**8,34**	**2,32**	**3600**	**3924**	**455**	**42460**	6,68	3,56	3600	9299	0	4278
	19	4917	n.a.	**9,91**	**4,48**	**3600**	**3146**	**268**	**25268**	8,22	5,78	3600	5298	0	3037
	21	5100	n.a.	12,46	9,16	3600	1283	81	14335	**11,19**	**9,02**	**3600**	**3321**	**0**	**1923**
fnl4461	11	1889	1889	**0,48**		**0,2**	**151**	**126**	**35**	0,50		0,6	54	0	10
	13	2088	2088	2,39		7,2	471	397	1614	**0,50**		**1,6**	**81**	**0**	**8**
	15	2356	2356	6,69		3005,5	2658	2165	137856	**1,78**		**16,8**	**238**	**0**	**73**
	17	2517	2517	9,69	4,29	3600	3467	1867	45487	**3,80**		**102,6**	**560**	**0**	**168**
	19	2933	n.a.	18,57	15,27	3600	3195	847	25541	**11,02**	**3,02**	3600	8883	0	2639
	21	3561	n.a.	29,87	27,14	3600	5994	1288	14874	**19,71**	**3,55**	3600	7174	0	1408

Table A.2: Detailed results for C1 instances (continued).

Instance				Proposed B&C						Côté et al. B&C					
name	n	UB	Opt	Root	Gap	Time	Cuts	Path	Nodes	Root	Gap	Time	Cuts	Path	Nodes
nrw1379	11	2690	2690	**2,12**		**0,5**	**43**	**4**	**12**	2,16		0,9	39	0	6
	13	3061	n.a.	11,39	5.00	3600	2299	452	32750	**10,74**	**3,68**	**3600**	**21519**	**1**	**10552**
	15	3117	n.a.	10,67	5,95	3600	790	31	39037	**10,67**	**5,63**	**3600**	**18099**	**0**	**7165**
	17	3197	n.a.	**10,24**	**5,72**	**3600**	**1032**	**31**	**24576**	10,07	6,21	3600	11748	0	4310
	19	3476	n.a.	15,84	13,2	3600	799	1	18407	**14,97**	**12,11**	**3600**	**7324**	**0**	**2253**
	21	3799	n.a.	18,83	16,17	3600	970	0	9820	**16,89**	**14,69**	**3600**	**5533**	**0**	**1466**
pr1002	11	13718	13718	**2,73**		**0,2**	**54**	**42**	**39**	1,05		0,4	45	0	7
	13	15436	15436	**5,30**		**1,4**	**78**	**117**	**355**	3,34		5,1	121	0	37
	15	16268	16268	**5,80**		**38,6**	**326**	**895**	**6995**	5,01		146,6	1686	0	813
	17	17601	17601	**6,01**		**164,5**	**377**	**1092**	**14322**	4,30		384,2	3143	0	1484
	19	18673	18673	**5,62**		**343,7**	**425**	**1619**	**18917**	4,33		1761,7	9266	0	4414
	21	20199	**20150**	**5,66**		**578,7**	**496**	**1332**	**18772**	5,88	2,31	3600	4388	0	2456
ts225	11	22000	22000	**0.00**		**0,3**	**50**	**95**	**52**	0,00		1,5	66	3	20
	13	29395	29395	**0.00**		**0,1**	**52**	**17**	**8**	0,26		2,4	51	0	15
	15	32541	32541	4,48		286,3	2636	1866	28041	**2,66**		**4,0**	**63**	**0**	**10**
	17	36405	36405	10,67		262,2	1504	1636	16409	**7,78**		**44,4**	**241**	**1**	**73**
	19	40395	n.a.	**15,61**	**0,63**	**3600**	**1565**	**1335**	**63491**	10,55	2,18	3600	5309	0	3156
	21	43056	n.a.	17,14	6,07	3600	894	122	34298	**13,31**	**5,65**	**3600**	**4980**	**0**	**2299**

Table A.3: Detailed results for C1 instances (continued).

Instance				**Proposed B&C**						**Côté et al. B&C**					
name	n	UB	Opt	Root	Gap	Time	Cuts	Path	Nodes	Root	Gap	Time	Cuts	Path	Nodes
a280	11	455	455	**11,76**		**4,0**	**320**	**97**	**760**	4,89		8,6	252	5	82
	13	479	479	**12,53**		**5,6**	**363**	**128**	**815**	4,62		15,3	450	3	139
	15	592	553	**15,01**		**255,5**	**967**	**395**	**7369**	5,95		497,5	3105	19	1131
	17	655	635	**12,44**		**234,3**	**842**	**511**	**6625**	6,93		1082,3	2509	4	1106
	19	717	**677**	**11,82**		**475,7**	**1039**	**589**	**9006**	11,57	6,94	3600	7622	11	3266
	21	793	n.a.	**15,15**	**4,63**	**3600**	**1722**	**1228**	**28314**	15,98	12,68	3600	4199	0	1753
att532	11	4190	4190	**1,32**		**0,1**	**27**	**6**	**6**	0,50		0,3	17	0	3
	13	5033	5033	**3,07**		**0,9**	**75**	**40**	**120**	3,31		4,7	97	0	44
	15	5665	n.a.	**11,68**	**3,72**	**3600**	**930**	**274**	**54869**	11,08	5,23	3600	13263	0	7842
	17	5920	n.a.	**12,13**	**6,40**	**3600**	**855**	**108**	**44437**	11,60	6,98	3600	10772	0	5551
	19	6184	n.a.	**12,3**	**6,22**	**3600**	**996**	**101**	**26789**	11,26	7,68	3600	5083	0	2560
	21	10025	n.a.	10,80	8,31	3600	620	29	8736	**9,85**	**8,12**	**3600**	**4311**	**0**	**1566**
brd14051	11	4386	4386	**3,52**		**1,9**	**71**	**3**	**100**	3,45		4,0	66	0	31
	13	4459	4458	**4,88**		**168,3**	**400**	**297**	**14513**	4,87		960,4	5276	1820	6540
	15	4795	n.a.	10,09	6,13	3600	663	0	39324	**10,07**	**3,99**	**3600**	**29334**	**0**	**5802**
	17	4891	n.a.	11,5	8,43	3600	796	48	22212	**11,25**	**8,21**	**3600**	**20218**	**0**	**3569**
	19	6276	n.a.	29,38	27,72	3600	2328	174	12907	**5,25**	**2,81**	**3600**	**10015**	**0**	**2180**
	21	6322	n.a.	29,74	28,48	3600	1153	38	6166	**4,95**	**3,37**	**3600**	**6494**	**0**	**1846**

Table A.4: Detailed results for C2 instances.

Instance				**Proposed B&C**						**Côté et al. B&C**					
name	n	UB	Opt	Root	Gap	Time	Cuts	Path	Nodes	Root	Gap	Time	Cuts	Path	Nodes
d15112	11	73872	73872	**3,85**		**0,8**	**229**	**86**	**119**	5,40		5,1	98	12	78
	13	81657	81657	**8,94**		**224,1**	**2288**	**1040**	**22060**	8,52		452,7	2935	996	3734
	15	91799	91799	**8,06**		**748,0**	**1472**	**1516**	**49969**	7,34		1402,2	4781	1673	7027
	17	97040	n.a.	**9,73**	**2,56**	**3600**	**4991**	**2332**	50850	9,21	3,22	3600	8631	93	5481
	19	99729	n.a.	**7,98**	**1,85**	**3600**	**4093**	**738**	**40675**	7,88	3,48	3600	5160	8	3784
	21	105242	n.a.	**12,37**	**7,48**	**3600**	**2538**	**303**	**19036**	11,03	8,50	3600	3048	0	1706
d18512	11	4341	4341	**2,41**		**27,6**	**646**	**176**	**6605**	2,40		70,6	1467	186	1085
	13	4572	n.a.	7,05	3,47	3600	3370	775	72309	**5,97**	**2,35**	**3600**	**28051**	**3162**	**17907**
	15	4893	n.a.	11,84	4,33	3600	2779	334	30533	**6,91**	**2,82**	**3600**	**13879**	**0**	**4795**
	17	5099	n.a.	14,28	8,00	3600	1841	127	14487	**5,84**	**4,15**	**3600**	**14801**	**0**	**3672**
	19	5359	n.a.	17,34	12,77	3600	1410	66	19221	**9,56**	**6,48**	**3600**	**8993**	**0**	**2535**
	21	5768	n.a.	22,6	20,13	3600	947	25	13624	**12,69**	**12,00**	**3600**	**5936**	**0**	**1520**
fnl4461	11	1883	1889	**0,16**		**0,1**	**65**	**37**	**33**	0,66		0,5	30	1	6
	13	2088	2088	2,39		23,5	855	316	6083	**2,19**		**14,7**	**332**	**33**	**189**
	15	2262	2262	2,81		73,4	865	437	10797	**2,99**		**36,8**	**502**	**26**	**224**
	17	2428	n.a.	**6,38**	**2,75**	**3600**	**3647**	**757**	**74835**	7,31	3,09	3600	12539	2	6083
	19	2634	n.a.	**9,32**	**6,60**	**3600**	**3939**	**683**	**39123**	11,11	8,35	3600	7019	7	3603
	21	2773	n.a.	**9,94**	**7,46**	**3600**	**2785**	**366**	**13986**	10,94	8,79	3600	2522	0	1919

Table A.5: Detailed results for C2 instances (continued).

Instance				**Proposed B&C**						**Côté et al.** B&C					
name	n	UB	Opt	Root	Gap	Time	Cuts	Path	Nodes	Root	Gap	Time	Cuts	Path	Nodes
nrw1379	11	2690	2690	**2,04**		**0,5**	**73**	**34**	**25**	2,34		1,3	38	0	9
	13	3055	n.a.	10,17	3,19	3600	1034	247	51593	**10,48**	**2,82**	**3600**	**27110**	**5**	**13395**
	15	3116	n.a.	10,64	6,19	3600	1338	73	36084	**10,61**	**5,62**	**3600**	**13525**	**18**	**6614**
	17	3197	n.a.	10,28	6,15	3600	1143	14	23586	**10,14**	**6,01**	**3600**	**11452**	**0**	**4628**
	19	3422	n.a.	14,54	11,66	3600	1090	0	14217	**12,02**	**9,47**	**3600**	**4320**	**0**	**1482**
	21	3769	n.a.	18,19	15,76	3600	1566	0	10781	**13,08**	**11,33**	**3600**	**5390**	**0**	**1603**
pr1002	11	13527	13527	**1,92**		**0,4**	**103**	**100**	**161**	1,55		2,0	39	18	68
	13	15221	15221	4,21		17,8	194	907	6617	**3,92**		**14,2**	**170**	**60**	**271**
	15	15676	15676	**2,56**		**14,4**	**167**	**845**	**3768**	3,19		30,9	333	35	301
	17	17009	17009	**2,59**		**20,8**	**169**	**1121**	**2942**	2,79		75,7	574	52	427
	19	18136	18136	**3,28**		**172,5**	**265**	**2758**	**15031**	3,31		504,8	2085	66	1558
	21	19613	19613	**3,29**		**135,9**	**225**	**1019**	**10188**	4,24		1322,2	4018	95	2650
ts225	11	22000	22000	**0.00**		**0,2**	**81**	**51**	**28**	0,00		0,9	39	0	9
	13	34000	34000	13,54	1,45	3600	2309	1469	114861	**10,60**		**985,3**	**9191**	**468**	**5419**
	15	37703	n.a.	17,56	5,2	3600	2247	851	100105	**10,57**	**3,44**	**3600**	**15077**	**802**	**8884**
	17	41703	n.a.	21,62	7,09	3600	2684	672	64030	**15,23**	**5,63**	**3600**	**10796**	**392**	**6549**
	19	45703	n.a.	**25,55**	**11,09**	**3600**	**1162**	**294**	**15347**	20,03	11,29	3600	4652	0	2348
	21	49097	n.a.	**27,33**	**13,23**	**3600**	**1391**	**242**	**21826**	20,21	14,18	3600	3476	0	1759

Table A.6: Detailed results for C2 instances (continued).

Bibliography

Ahuja, R. K., Kodialam, M., Mishra, A. K., and Orlin, J. B. (1997). Computational investigation of maximum flow algorithms. *European Journal of Operational Research*, (3):509--542. ISSN 1572-5286.

Applegate, D. L., Bixby, R. E., Chvatal, V., and Cook, W. J. (2007). *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton University Press, Princeton, NJ, USA. ISBN 0691129932, 9780691129938.

Ascheuer, N., Jünger, M., and Reinelt, G. (2000). A branch & cut algorithm for the asymmetric traveling salesman problem with precedence constraints. *Computational Optimization and Applications*, 17(1):61--84.

Azi, N., Gendreau, M., and Potvin, J.-Y. (2010). An exact algorithm for a vehicle routing problem with time windows and multiple use of vehicles. *European Journal of Operational Research*, 202(3):756–763.

Balas, E., Fischetti, M., and Pulleyblank, W. (1995). The precedence-constrained asymmetric traveling salesman polytope. *Mathematical Programming*, 68(1-3):241--265.

Baldacci, R., Hadjiconstantinou, E., and Mingozzi, A. (2004). An exact algorithm for the capacitated vehicle routing problem based on a two-commodity network flow formulation. *Operations Research*, 52(5):723--738. ISSN 0030-364X.

Berbeglia, G., Cordeau, J.-F., Gribkovskaia, I., and Laporte, G. (2007). Static pickup and delivery problems: a classification scheme and survey. *TOP: An Official Journal of the Spanish Society of Statistics and Operations Research*, 15(1):1--31. ISSN 1134-5764.

Borne, S., Grappe, R., and Lacroix, M. (2012). The Uncapacitated Asymmetric Traveling Salesman Problem with Multiple Stacks. In Mahjoub, Markakis, V., Milis, I., and Paschos, V., editors, *Combinatorial Optimization*, volume 7422 of *Lecture Notes in Computer Science*, pages 105--116. Springer Berlin Heidelberg.

Carrabs, F., Cerulli, R., and Cordeau, J.-F. (2007a). An additive branch-and-bound algorithm for the pickup and delivery traveling salesman problem with lifo or fifo loading. *INFOR: Information Systems and Operational Research*, 45(4):223--238.

Carrabs, F., Cordeau, J. F., and Laporte, G. (2007b). Variable neighborhood search for the pickup and delivery traveling salesman problem with lifo loading. *INFORMS Journal on Computing*, 19(4):618--632. ISSN 1526-5528.

Casazza, M., Ceselli, A., and Nunkesser, M. (2012). Efficient algorithms for the double traveling salesman problem with multiple stacks. *Computers & Operations Research*, 39(5):1044--1053. ISSN 03050548.

Cassani, L. (2004). Algoritmi euristici per il TSP with rear-loading. Master's thesis, DTI - Università degli Studi di Milano.

Chvátal, V., Cook, W., Dantzig, G., Fulkerson, D., and Johnson, S. (2010). Solution of a Large-Scale Traveling-Salesman Problem. In Jünger, M., Liebling, T. M., Naddef, D., Nemhauser, G. L., Pulleyblank, W. R., Reinelt, G., Rinaldi, G., and Wolsey, L. A., editors, *50 Years of Integer Programming 1958-2008*, pages 7--28. Springer Berlin Heidelberg.

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

Cordeau, J.-F., Laporte, G., Savelsbergh, M. W., and Vigo, D. (2007). Vehicle routing. In Barnhart, C. and Laporte, G., editors, *Transportation*, volume 14 of *Handbooks in Operations Research and Management Science*, pages 367 – 428. Elsevier.

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

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

Dezs, B., Jüttner, A., and Kovács, P. (2011). LEMON - an Open Source C++ Graph Template Library. *Electronic Notes in Theoretical Computer Science*, 264(5):23--45. ISSN 1571-0661.

Dinitz, Y. (2006). Dinitz' Algorithm: The Original Version and Even's Version. In Goldreich, O., Rosenberg, A., and Selman, A., editors, *Theoretical Computer Science*, volume

3895 of *Lecture Notes in Computer Science*, chapter 10, pages 218--240. Springer Berlin Heidelberg, Berlin, Heidelberg.

Dumitrescu, I., Ropke, S., Cordeau, J.-F., and Laporte, G. (2010). The traveling salesman problem with pickup and delivery: polyhedral results and a branch-and-cut algorithm. *Mathematical Programming*, 121(2):269--305.

Felipe, A., Ortuño, T., and Tirado, G. (2009). The double traveling salesman problem with multiple stacks: A variable neighborhood search approach. *Computers & Operations Research*, 36(11):2983--2993. ISSN 0305-0548.

Fischetti, M. and Toth, P. (1997). A polyhedral approach to the asymmetric traveling salesman problem. *Management Science*, 43(11):1520--1536. ISSN 0025-1909.

Fügenschuh, A. and Martin, A. (2005). *Computational Integer Programming and Cutting Planes*, volume 12 of *Handbooks in Operations Research and Management Science*, pages 69--121. Elsevier.

Goldberg, A. V. and Tarjan, R. E. (1986). A new approach to the maximum flow problem. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, STOC '86, pages 136--146, New York, NY, USA. ACM.

Gouveia, L. and Pesneau, P. (2006). On extended formulations for the precedence constrained asymmetric traveling salesman problem. *Networks*, 48(2):77--89. ISSN 0028-3045.

Gouveia, L. and Pires, J. M. (2001). The asymmetric travelling salesman problem: on generalizations of disaggregated Miller-Tucker-Zemlin constraints. *Discrete Applied Mathematics*, 112(1-3):129–145. ISSN 0166-218X. Combinatorial Optimization Symposium, Selected Papers.

Hernández-Pérez, H. and Salazar-González, J.-J. (2004). A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery. *Discrete Applied Mathematics*, 145(1):126--139. ISSN 0166218X.

Iori, M., González, J. J. S., and Vigo, D. (2007). An Exact Approach for the Vehicle Routing Problem with Two-Dimensional Loading Constraints. *Transportation Science*, 41(2):253--264. ISSN 1526-5447.

Iori, M. and Martello, S. (2010). Routing problems with loading constraints. *TOP: An Official Journal of the Spanish Society of Statistics and Operations Research*, 18(1):4--27. ISSN 1134-5764.

Jünger, M., Reinelt, G., and Thienel, S. (1995). *Practical Problem Solving with Cutting Plane Algorithms in Combinatorial Optimization*, volume 20 of *DIMACS series in discrete mathematics and theoretical computer science*, pages 111--152. American Mathematical Society.

Kalantari, B., Hill, A. V., and Arora, S. R. (1985). An algorithm for the traveling salesman problem with pickup and delivery customers. *European Journal of Operational Research*, 22(3):377--386. ISSN 03772217.

Ladany, S. P. and Mehrez, A. (1984). Optimal routing of a single vehicle with loading and unloading constraints. *Transportation Planning and Technology*, 8(4):301--306.

Levitin, G. and Abezgaouz, R. (2003). Optimal routing of multiple-load AGV subject to LIFO loading constraints. *Computers & Operations Research*, 30(3):397--410. ISSN 03050548.

Li, Y., Lim, A., Oon, W.-C., Qin, H., and Tu, D. (2011). The tree representation for the pickup and delivery traveling salesman problem with LIFO loading. *European Journal of Operational Research*, 212(3):482--496. ISSN 03772217.

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

Miller, C. E., Tucker, A. W., and Zemlin, R. A. (1960). Integer Programming Formulation of Traveling Salesman Problems. *J. ACM*, 7(4):326--329. ISSN 0004-5411.

Pacheco, J. A. (1997). Heurístico para los problemas de rutas con carga y descarga en sistemas LIFO. *Quaderns d'estadística i investigació operativa*, 21(1-2).

Padberg, M. and Rinaldi, G. (1990). An efficient algorithm for the minimum capacity cut problem. *Mathematical Programming*, 47(1-3):19--36.

Padberg, M. and Sung, T.-Y. (1991). An analytical comparison of different formulations of the travelling salesman problem. *Mathematical Programming*, 52(1-3):315--357.

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

Petersen, H. L. and Madsen, O. B. G. (2009). The double travelling salesman problem with multiple stacks - Formulation and heuristic solution approaches. *European Journal of Operational Research*, 198(1):139--147. ISSN 03772217.

Reinelt, G. (1991). TSPLIB - A Traveling Salesman Problem Library. *ORSA Journal on Computing*, 3(4):376--384. ISSN 0899-1499.

Santos, F. A., Mateus, G. R., and da Cunha, A. S. (2013). The pickup and delivery problem with cross-docking. *Computers & Operations Research*, 40(4):1085--1093. ISSN 03050548.

Sarin, S. C., Sherali, H. D., and Bhootra, A. (2005). New tighter polynomial length formulations for the asymmetric traveling salesman problem with and without precedence constraints. *Operations Research Letters*, 33(1):62--70. ISSN 01676377.

Tarantilis, C. D., Zachariadis, E. E., and Kiranoudis, C. T. (2009). A hybrid metaheuristic algorithm for the integrated vehicle routing and three-dimensional container-loading problem. *Intelligent Transportation Systems, IEEE Transactions on*, 10(2):255--271. ISSN 1524-9050.

Toulouse, S. and Wolfler Calvo, R. (2009). On the complexity of the multiple stack TSP, kSTSP. In Chen, J. and Cooper, editors, *Theory and Applications of Models of Computation*, volume 5532 of *Lecture Notes in Computer Science*, pages 360--369. Springer Berlin Heidelberg.

Urrutia, S., Milanés, A., and Løkketangen, A. (2013). A dynamic programming based local search approach for the double traveling salesman problem with multiple stacks. *International Transactions in Operational Research*. In press.