ABORDAGENS EXATAS PARA PROBLEMAS DE TOPOLOGIA DE REDE E ROTEAMENTO

ARMANDO HONORIO PEREIRA

ABORDAGENS EXATAS PARA PROBLEMAS DE TOPOLOGIA DE REDE E ROTEAMENTO

Tese 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 Doutor em Ciência da Computação.

Orientador: Geraldo Robson Mateus
Coorientador: Sebastián Alberto Urrutia

Belo Horizonte

Maio de 2021

ARMANDO HONORIO PEREIRA

EXACT APPROACHES FOR NETWORK TOPOLOGY AND ROUTING PROBLEMS

Thesis presented to the Graduate Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Doctor in Computer Science.

Advisor: Geraldo Robson Mateus
Co-Advisor: Sebastián Alberto Urrutia

Belo Horizonte
May 2021

Pereira, Armando Honório

P436e Exact approaches for network topology and routing problems [manuscrito] / Armando Honório Pereira. — Belo Horizonte, 2021.
xiv, 81 f. : il. ; 29cm

Orientador: Geraldo Robson Mateus.
Coorientador: Sebastián Alberto Urrutia.

Tese (doutorado) - Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação.
Referências: f. 74-81

1. Computação – Teses. 2. Otimização combinatória - Teses. 3. Programação inteira - Teses. 4. Redes de sensores sem fio – Teses. 5. Problema do caixeiro viajante – Teses. I. Mateus, Geraldo Robson. II. Urrutia, Sebastián Alberto. III. Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação. IV. Título.

CDU 519.6*61(043)

Ficha catalográfica elaborada pela bibliotecária Irénquer Vismeg Lucas Cruz - CRB 6ª Região nº 819.

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

FOLHA DE APROVAÇÃO

EXACT APPROACHES FOR NETWORK TOPOLOGY AND ROUTING PROBLEMS

ARMANDO HONÓRIO PEREIRA

Tese defendida e aprovada pela banca examinadora constituída pelos Senhores:

Belo Horizonte, 18 de maio de 2021.

Documento assinado eletronicamente por **Geraldo Robson Mateus**, **Professor Magistério Superior - Voluntário**, em 21/05/2021, às 10:31, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Sebastian Alberto Urrutia**, **Professor do Magistério Superior**, em 25/05/2021, às 18:06, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Gabriel de Morais Coutinho**, **Professor do Magistério Superior**, em 14/06/2021, às 09:09, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Rafael Augusto de Melo**, **Usuário Externo**, em 14/06/2021, às 13:16, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Haroldo Gambini Santos**, **Usuário Externo**, em 18/06/2021, às 14:24, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Rosiane de Freitas Rodrigues**, **Usuário Externo**, em 22/06/2021, às 01:17, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site https://sei.ufmg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **0740995** e o código CRC **9CA075B3**.

Referência: Processo nº 23072.226417/2021-95 SEI nº 0740995

Acknowledgments

I thank my advisor, prof. Geraldo Robson Mateus, and my co-advisor, prof. Sebastián Alberto Urrutia, for believing in my ability and having helped me during my doctorate.

I thank the PPGCC secretaries for their attention and efficiency.

I thank all PPGCC teachers who contributed to my educational background.

I thank UFMG for the infrastructure.

I thank the examining board for dedicating part of their time to evaluate my work.

I thank all the members of LAPO for their pleasant coffee breaks and discussions.

I am thankful to the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) and the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) for providing financial support during my doctorate course.

I thank my parents and my brother for having always supported me.

I thank my girlfriend for being by my side.

"It does not matter that only a few in each generation will grasp and achieve the full reality of man's proper stature–and that the rest will betray it. It is those few that move the world and give life its meaning–and it is those few that I have always sought to address. The rest are no concern of mine; it is not me or The Fountainhead that they will betray: it is their own souls."

(Ayn Rand, *The Fountaihead*)

Resumo

Nesta tese, investigamos abordagens de solução exata para o *p-arborescence star problem* (*p*-ASP) e o *pickup and delivery traveling salesman problem with multiple stacks* (PDTSPMS). Esses dois problemas de otimização combinatória possuem aplicações no projeto de redes de sensores sem fio e no planejamento de rotas veiculares sob restrições de carregamento, respectivamente. Para o *p*-ASP, nós desenvolvemos algoritmos *branch-and-cut* baseados em cada uma das nossas duas formulações propostas e melhoramos um algoritmo *branch-and-cut* anterior para o problema com um método de separação exata. Além disso, provamos que encontrar uma solução viável para uma instância arbitrária do *p*-ASP é NP-difícil e introduzimos procedimentos de pré-processamento. Para instâncias do *benchmark* de tamanho pequeno e médio, nossos algoritmos propostos obtém os melhores resultados. Para as instâncias maiores, nosso melhor algoritmo mostra-se competitivo com a versão melhorada da abordagem existente. Ambos os algoritmos têm desempenho similar para essas instâncias difíceis e resolvem uma instância que o outro não é capaz. Com relação ao PDTSPMS, propomos uma nova formulação junto com novas desigualdades válidas. As novas desigualdades válidas são versões fortalecidas de desigualdades usadas com sucesso em algoritmos para o problema e suas variantes. Em seguida, implementamos um algoritmo *branch-and-cut* baseado na formulação proposta que incorpora as desigualdades válidas. Além disso, também apresentamos várias estratégias de aceleração para nosso método. O algoritmo é comparado com todas as abordagens exatas para o PDTSPMS encontradas na literatura. Nosso algoritmo implementado supera todos os algoritmos para as instâncias de *benchmark*. Além de reduzir drasticamente o tempo necessário para resolver a maioria das instâncias, o algoritmo proposto resolve mais instâncias no total do que os outros métodos.

Palavras-chave: programação inteira; *branch-and-cut*; arborescência; redes de sensores sem fio; caixeiro viajante; restrições de carregamento; coleta e entrega.

Abstract

In this thesis, we investigate exact solution approaches for the p-arborescence star problem (p-ASP) and the pickup and delivery traveling salesman problem with multiple stacks (PDTSPMS). These two combinatorial optimization problems have applications in the design of wireless networks and the planning of vehicle routes under loading constraints, respectively. For the p-ASP, we develop branch-and-cut algorithms based on each one of our two proposed formulations and improve an earlier branch-and-cut algorithm for the problem with an exact separation method. Also, we prove that finding a feasible solution for an arbitrary p-ASP instance is NP-hard and introduce preprocessing procedures. For small and medium-sized benchmark instances, our proposed algorithms provide the best results. For large instances, our best algorithm shows to be competitive with the improved version of the existing approach. Both algorithms perform similarly for these hard instances and solve one instance that the other does not. Regarding the PDTSPMS, we propose a new formulation along with new valid inequalities. The new valid inequalities are lifted versions of inequalities used successfully in exact algorithms for the problem and its variants. Then, we implement a branch-and-cut algorithm based on the proposed formulation which incorporates the valid inequalities. Moreover, we also present several acceleration strategies for our method. The algorithm is compared with all exact approaches for the PDTSPMS found in the literature. Our implemented algorithm outperforms all other algorithms for the benchmark instances. Besides drastically reducing the time needed to solve most of the instances, the proposed algorithm solved more instances in total than the other methods.

Keywords: integer programming; branch-and-cut; arborescence; wireless sensor networks; traveling salesman; loading constraints; pickup and delivery.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

In this first chapter, we outline and motivate the two problems studied in this thesis. First, we motivate and give a general description of the investigated problems. Next, we detail the contributions resulted from the work of this doctorate. Finally, in the last section, we summarize the structure and organization of the present text.

1.1 Motivation

Multiple applications found in the design of wireless sensor networks and planning of vehicle fleets involve discrete choices, for instance, the topology of the wireless network, where to install transmission towers, the number of vehicles to use, which customers they should visit, and etcetera. Integer linear programming approaches provide the means to model such choices. The essence of these models involves deciding which elements will be part of a solution or not. Thus, the use of combinatorial optimization techniques allows us to model and solve a huge number of practical situations. Problems involving network planning and vehicle routing are among the most studied topics in combinatorial optimization [Magnanti and Wong, 1984].

So far, there is no efficient algorithm of polynomial complexity to solve general problems of integer linear programming. Despite this difficulty, several successful approaches have been proposed to address these types of problems. The idea of a branch-and-bound enumeration method [Land and Doig, 1960] combined with a cutting planes algorithm [Gomory, 1958] to evaluate the relaxed linear problems resulted in branch-and-cut methods. This method has been successfully used for several combinatorial optimization problems in recent years [Morrison et al., 2016]. For further reading and study about integer programming, we refer the reader to the works of Wolsey [1998] and Cook et al. [2011].

Networks can be used to represent communication systems where there is an exchange of information between elements. Likewise, routing problems are used to model problems where one or more vehicles must visit a set of locations. These classes of problems are usually encountered with several additional constraints. Regarding network design problems, we can find in the literature problems involving networks with a fixed number of nodes [Golden et al., 2012], selection of clusters with coverage constraints [Calik et al., 2017], topologies with multiple levels [Labbé et al., 2004], among others. Similarly, we can find vehicle routing problems involving plenty of features like time windows [Azi et al., 2007], distance constraints [Kek et al., 2008], pickup and delivery restrictions for multiple vehicles [Ropke and Cordeau, 2009], and several others. We indicate the surveys of Akyildiz et al. [2002] and Pollaris et al. [2015] for an overview of problems found in the area of design of wireless sensor networks and vehicle routing, respectively.

We study two combinatorial optimization problems in this thesis, namely, the p-arborescence star problem (p-ASP), and the pickup and delivery traveling salesman problem with multiple stacks (PDTSPMS). The p-ASP consists of determining p cluster-heads such that they define a reverse arborescence topology covering the remaining nodes of the network. The problem finds applications in the context of wireless sensor networks design [Morais et al., 2019]. In such settings, a clustering protocol is commonly used to limit the number of active nodes, aiming at extending the network lifetime. The PDTSPMS is a routing problem where a single vehicle must pick up and deliver items to complete customer requests. The items are stored in stacks of limited capacity which follow the last-in-first-out policy. The aim of the problem is to find a vehicle route of minimum cost fulfilling all requests. Practical applications of the problem include situations where the vehicle can only be operated through the rear or where relocating items inside the vehicle is expensive [Pereira and Urrutia, 2018].

Although the p-ASP and the PDTSPMS have different characteristics and definitions, we can still see them in a general form and use similar approaches. Both problems can be defined in directed graphs subject to a set of side constraints. By modeling such constraints in an integer programming formulation, we can relax them and use a branch-and-cut approach to solve them. A common aspect between the problems is the connectivity constraint which can be modeled through cutsets or subtour elimination constraints. Consequently, strategies to separate and manage such inequalities can be applied to both problems. Other approaches such as preprocessing and arc elimination can also be applied to the p-ASP and PDTSPMS.

Since they are challenging problems with multiple applications, the main motivation of this thesis is to improve the state-of-the-art for these two problems using well-established optimization techniques for exact approaches. Although the first problem follows a line of tree-based formulations while the second fits the group of problems based on path generation, the motivation for this thesis is to use the same exact algorithm structure for both problems, more specifically exploring classic branch-and-cut methods. Traditional strategies for improving such algorithms are the study of valid inequalities and separation methods. Valid inequalities and separation methods can improve the dual bounds obtained from the linear relaxations of the formulations. Thus, small reductions in the optimality gaps can drastically decrease the size of the branch-and-bound tree. Moreover, we also study acceleration strategies for the algorithms based on specific aspects of the problems. Even though the focus of this thesis is on specific topology and routing problems, the new algorithms and strategies introduced for these problems can also be extended to other similar variants.

1.2 Contributions

Our main contributions to the p-ASP concern exact algorithms and theoretical results. We propose new models for the problem together with theoretical results of the relationship between the existing formulations. We develop an exact separation procedure for an inequality that was previously separated using a heuristic method. The new separation procedure improves the existing branch-and-cut algorithm. Besides, we also introduce two new exact algorithms for the problem. Given an arbitrary p-ASP instance, we show that finding a feasible solution is an NP-hard problem. Although it is NP-hard to find a feasible solution for the problem, we present preprocessing algorithms that allow us to determine if a set of nodes contains at least one cluster-head. Considering the same computational environment, our proposed branch-and-cut algorithms obtain better results than the previous existing approach for the problem.

The major contribution for the PDTSPMS is a new branch-and-cut algorithm. The algorithm is based on a new formulation that incorporates new valid inequalities. The model uses the same set of constraints as previous approaches to model connectivity and precedence. However, we use new constraints for the LIFO policy and capacity. We introduce new valid inequalities and use specific characteristics of the problem to lift existing valid inequalities from the PDTSPMS literature. Furthermore, we present new separation procedures and acceleration strategies that are incorporated in our branch-and-cut algorithm. One of the strategies solves a bin packing problem to check if the corresponding capacity inequality can be lifted. All algorithms from the PDTSPMS literature are executed and compared in the same computational environment. Our proposed branch-and-cut algorithm outperforms all other algorithms considering the benchmark instances.

1.3 Text organization

The primary objective of our research is to devise exact algorithms for the p-ASP and PDTSPMS. In this thesis, we present and discuss the findings and results of our research during the doctoral program. The remainder of this text is divided into three chapters. Chapter 2 is dedicated to the p-ASP and is based on the article Pereira et al. [2020] with a few additions. In particular, we added Theorems 1 and 2 which were removed during the peer review phase. We chose to add these proofs since they might help the reader have a better understanding in his first contact with the work. The reader might be a student in his contact with the work and might not be familiar with the results of Magnanti and Wolsey [1995] for network optimization problems on trees. Chapter 3 is about a new branch-and-cut algorithm for the PDTSPMS. Initially, we describe our formulation and propose new valid inequalities. Next, we describe the details of the branch-and-cut algorithm implemented based upon the formulation and valid inequalities. Finally, we analyze the results of the algorithm for the benchmark instances and compare them with the results of other approaches from the literature. This chapter was compiled into an article and submitted to a top journal in the field of operational research. We conclude with Chapter 4, in which we discuss the work done and the future work directions we plan to follow.

Although the chapters share similar terms, we emphasize that the notations used in Chapters 2 and 3 are distinct. We do not unify the notations between them because we chose to use the standard notation in the literature for each one of the problems. The p-ASP has a notation aligned with the context of networks, while the PDTSPMS has a notation directed to the context of vehicle routing.

Chapter 2

The p-arborescence star problem

Given a connected digraph, a vertex designated as the root, and an integer p, the p-arborescence star problem is to choose p vertices beside the root and define a reverse arborescence spanning them. Each vertex outside the arborescence must be assigned to one vertex inside it. The objective of the problem is to minimize arborescence and assignment costs. We propose two formulations for the problem and prove theoretical results about their strength. Moreover, we develop branch-and-cut algorithms based on each one of the formulations and improve an earlier branch-and-cut algorithm for the problem with an exact separation method. Additionally, we show that finding a feasible solution for an arbitrary instance is NP-hard and introduce preprocessing procedures for the problem. The proposed algorithms are evaluated with a set of benchmark instances from the literature. For small and medium-sized instances, our proposed algorithms provide the best results when compared to the existing algorithm from the literature. For large instances, our best algorithm obtains similar results to the improved version of the existing approach. Each algorithm solves an instance that the other is unable to. Therefore, for the hard instances in the benchmark, one algorithm does not dominate the other.

2.1 Introduction

Let $G = (V, A)$ be a symmetric connected digraph where V is the set of vertices and A the set of arcs. We observe that the set of arcs contains no self-loop and might be incomplete. A reverse arborescence is a directed rooted tree where there is precisely one elementary path from each vertex to the root. Associated with the use of each arc $(i, j) \in A$ there is a nonnegative arborescence or assignment cost c_{ij}. Given an integer p and a root $r \in V$, the p-arborescence star problem (p-ASP) consists of finding a subset $H \subseteq V \setminus \{r\}$ of vertices such that:

- The cardinality of H is equal to p.
- The subgraph induced by $H_r = H \cup \{r\}$ defines a reverse arborescence $T = (H_r, A_T)$ rooted at r, $A_T \subseteq A$.
- The subset H is a dominating set of G: for each vertex $v \in V \setminus H_r$, there is a vertex $i \in H$ such that $(v, i) \in A$.
- The cost $\sum\limits_{(u,v)\,\in\,A_T} c_{uv} + \sum\limits_{u\,\in\,V\setminus H_r} \arg\min\limits_{v \in H} c_{uv}$ of making all the connections is minimized.

In summary, we wish to find a reverse arborescence of size $p + 1$ rooted at r such that vertices outside the arborescence are covered by exactly one vertex inside it, and the cost of the connections is minimized. We call the reverse arborescence T the backbone. A vertex $i \in H$ is called cluster-head and the vertices not in H directly connected to it form its cluster. Given an arc $(i, j) \in A$, if $i \in H$ and $j \in H$ we say it is an inter-cluster arc, else, if $j \in H$ and $i \in V \setminus H_r$ we say it is an intra-cluster arc. The name star is given from the intra-cluster star topology derivated from the arcs pointing towards a cluster-head.

Figure 2.1 displays a feasible p-ASP solution. The graph has 41 vertices and the value p is set to 9. In the figure, cluster-heads are represented as gray vertices, and the root is indicated by a black vertex. The dashed arcs describe intra-cluster arcs. The backbone rooted at the root $r = 0$ is drawn with solid lines.

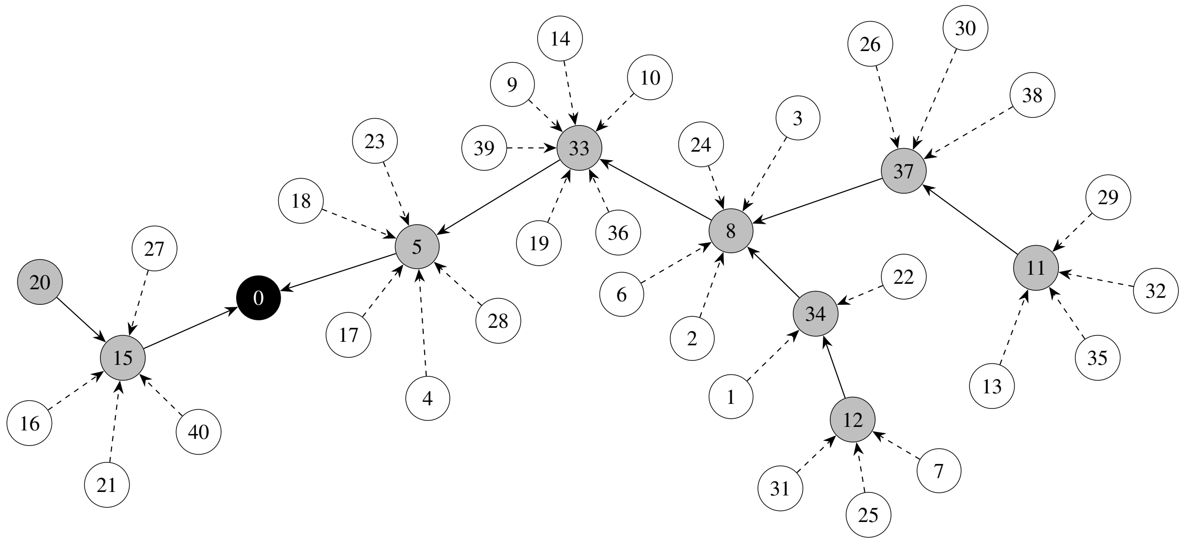


Figure 2.1: Graph showing a feasible p-ASP solution.

The p-ASP finds applications in the context of wireless sensor networks (WSNs). A WSN consists of a large number of sensors operating autonomously in unattended environments. These sensor nodes are responsible for monitoring physical or environmental variables, such as movement or temperature, and transmitting data to a sink node [Akyildiz et al., 2002]. These variables can then be used in meteorological forecasts or disaster management situations such as earthquakes. One of the advantages of a WSN is its ability to work in isolated or harsh environments, in which human supervision is costly or risky [Abbasi and Younis, 2007].

Sensors usually have limited memory, sensing devices, communication hardware, and a limited battery. Given the lifespan of the battery, the energy efficiency of the communication widely affects the network lifetime. Therefore, the efficient use of energy among the sensors is fundamental to prolong the lifetime of the network [Akyildiz et al., 2002]. An approach to reduce the energy consumption among sensors is a clustering protocol. The nodes of the network are divided into groups, each possessing a leader called cluster-head, in charge of communication with the sink [Younis and Akkaya, 2008]. In a typical network, the traffic is gathered from many sensors, combined, and finally forwarded to the sink.

Taking into consideration the previous discussion, the p-ASP consists of selecting p vertices to be cluster-heads, connecting them through a reverse arborescence topology, and assigning the regular sensors to the chosen cluster-heads. The aim is to maximize the network lifetime while minimizing transmission costs. The cluster-heads are expected to spend more energy than the regular nodes in the network. Therefore, the cardinality constraint restricts the percentage of active sensors in the network, reducing energy consumption and the overall cost if all the nodes were responsible for the transmissions. Clustering protocols such as LEACH [Heinzelman et al., 2000] and LEACH-C [Heinzelman et al., 2002] keep a fixed number of active nodes in the network, changing the cluster-heads of the network at regular intervals to increase the network lifetime. Practical applications of this type of problem might contain additional constraints. For instance, a cluster-head should have a limit on the number of incoming connections [Duarte-Melo and Liu, 2003] or have an energy capacity that might deplete along time [Matos et al., 2012].

The problem has polynomial cases depending on the value of p. In the case where $p = 1$, the chosen cluster-head $i \in V \setminus \{r\}$ must be connected through one arc to all $j \in V \setminus \{i\}$. If no such vertex i exists, then the problem is infeasible. Otherwise, choose the vertex $i \in V \setminus \{r\}$ which minimizes the cost $\sum_{(u,i)\in A} c_{ui} + c_{ir}$ as the only cluster-head. If $p = n$, the problem reduces to the minimum weight spanning arborescence problem [Chu and Liu, 1965; Edmonds, 1967]. The p-ASP is NP-hard in the general case, since the p-median problem [Hakimi, 1965] can be reduced to it [Morais et al., 2019].

The contributions of this work are two new integer linear programming formulations and branch-and-cut algorithms for the p-ASP. Additionally, we introduce several preprocessings for the problem, and improve an earlier branch-and-cut algorithm from the p-ASP literature with an exact separation method. We compare the branch-and-cut algorithms experimentally using the set of benchmark instances. The proposed algorithms are able to converge faster on the solved benchmark instances. As theoretical contributions, we study the relationship of the formulations and show that finding a feasible solution or proving the infeasibility of an arbitrary p-ASP instance is NP-hard.

The chapter is organized as follows. We provide a literature review for the p-ASP in Section 2.2. In Section 2.3, we present the proposed formulations for the problem along with theoretical results about their strength. Based on our proposed formulations, we describe branch-and-cut algorithms in Section 2.4. In this section, we also prove that determining the feasibility of an arbitrary p-ASP instance is NP-hard and describe our suggested preprocessing procedures. Section 2.5 compares the branch-and-cut algorithms using the set of benchmark instances. Moreover, we conduct a study about the impact of parameter p on the execution time. The last section summarizes the chapter and discusses ideas for future works.

2.2 Literature Review

The p-ASP was formally defined and introduced in the work of Morais et al. [2019]. They investigated exact solution approaches for the problem based on two formulations. The first is a compact formulation based on multicommodity directed flows. The second one uses an exponential number of cutset constraints to impose the connected backbone. Both formulations use two sets of arc variables to differentiate between intra and inter-cluster arcs. They also introduced a heuristic algorithm based on Benders decomposition to provide initial upper bounds for the proposed exact algorithms. The results showed that the cutset based formulation is computationally superior to the multicommodity flow formulation. The branch-and-cut algorithm based on the cutset formulation was able to solve more instances and converge to an optimal solution in less time. Morais and Mateus [2019] proposed a set partitioning p-ASP formulation based on the concept of configurations. The authors define a configuration as p cluster-heads and their corresponding stars. They devise a column-and-row generation algorithm to compute the lower bound given by the linear relaxation at the root node of the branch-and-bound tree. The connected backbone constraint is imposed on the master problem. Columns corresponding to configurations are added dynamically. Despite improving the lower bounds obtained by Morais et al. [2019], no exact algorithm based on the procedure to compute the correspondent lower bounds is provided.

Matos et al. [2012] studied a clustering problem to manage energy consumption in wireless sensor networks efficiently. The cluster of the problem has a fixed cardinality, but no topology is imposed on it. Despite describing the problem, no formal definition or formulation is proposed. They also implemented a Greedy Randomized Adaptive Search heuristic coupled with an intensification phase based on Path Relinking. The performance of their heuristic is compared with other clustering protocols found in the literature.

A related problem that Simonetti et al. [2011] considered is the minimum connected dominating set problem (MCDSP). They presented an integer programming formulation and valid inequalities that are incorporated in a branch-and-cut algorithm. In the MCDSP, one wishes to find a connected dominating set of minimum cardinality. Despite the need to exist at least one edge between a vertex outside the dominating set with a vertex inside it, no cost is associated with such edges in the objective function. Applications for the MCDSP arise in the design of ad-hoc wireless sensor networks where the network topology is dynamic. Gendron et al. [2014] studied several exact algorithms for solving the MCDSP. The authors investigated three approaches to deal with the problem, a probing strategy, a Benders decomposition algorithm, and a branch-and-cut method. Besides, they proposed hybrid algorithms based on the combination of the previous three strategies. Gendron et al. [2014] also explored several valid inequalities and integrated them into their algorithms according to their strengths. It is worth mentioning that the branch-and-cut algorithm of Gendron et al. [2014] is an improved version of the algorithm initially proposed by Simonetti et al. [2011]. In the MCDSP, unlike the p-ASP, the dominating set has no fixed cardinality associated with it, and there are no costs associated with connections.

The MCDSP is closely related to the maximum leaf spanning tree problem (MLSTP) where you have to find a spanning tree with the maximum number of leaves. Fujie [2003] proposed the first exact branch-and-cut algorithm for the problem. Later, Fujie [2004] conducted polyhedral investigations for the MLSTP and showed several facet defining inequalities. Lucena et al. [2010] proposed two formulations for the MLSTP and implemented branch-and-cut algorithms for these two formulations. The first model is a directed graph reformulation of the formulation used by Fujie [2004]. The second formulation recasts the MLSTP as a Steiner arborescence problem. Lucena et al. [2010] also enhanced the algorithm introduced by Fujie [2003] with a dynamic greedy heuristic. Recently, Gouveia and Simonetti [2017] presented a new extended model applicable to the MLSTP. The new formulation improved previous known dual bounds for the problem, and the algorithm based on it proved competitive with the existing approaches.

Another relevant problem is the tree star problem (T-SP) [Lucena et al., 2016]. In the T-SP, an edge has an assigned cost according to its role in the spanning tree that contains it. Considering different costs for edges connecting leaves and internal vertices, the T-SP is to find a minimum cost spanning tree. The problem has applications in telecommunications where links are priced according to their capacities. Lucena et al. [2016] introduced a mixed integer programming formulation based on generalized subtour elimination constraints together with a branch-and-cut algorithm.

Leitner et al. [2017] studied a family of problems with tree-star topologies, namely, the connected facility location problem (ConFL) [Gollowitzer and Ljubić, 2011], rent-or-buy problem [Nuggehalli et al., 2003; Swamy and Kumar, 2004], and the (generalized) Steiner tree-star [Khuller and Zhu, 2002]. Their approach consisted of showing that all the previous tree-star problems can be modeled as special cases of the asymmetric version of the ConFL. The ConFL consists of choosing which facilities to open, such that the core network induced from the chosen facilities is connected, and all the customers are assigned to an open facility. The objective is to minimize the cost of all connections and the cost associated with opening facilities. The authors incorporated three strategies to improve their branch-and-cut algorithm: (i) use of cuts based on a dual ascent procedure, (ii) reduction procedures in the input graph based on dual costs, and (iii) incorporation of primal heuristics to obtain tight upper bounds. Different from the p-ASP, there are subsets of vertices for each role in the network, Steiner vertices can be used to connect facilities, and no cardinality is imposed in the core network.

The p-cable trench problem (p-CTP) has a fixed cardinality constraint on its topology, like the p-ASP. In the problem, p facilities have to be located in order to define a forest such that every customer is connected to precisely one facility. Unlike the p-ASP, the opened facilities need not be connected, and there is no star topology. The customer and the facility do not need to be connected through only one arc. There may be other customers on the path between a facility and a specific customer. Marianov et al. [2012] defined the p-CTP as an extension of the original cable trench problem introduced by Vasko et al. [2002] to model telecommunication network problems. Marianov et al. [2012] proposed a multicommodity flow formulation and two Lagrangean relaxation heuristics for the p-CTP. Calik et al. [2017] generalized the previous variants of the problem and introduced an algorithm framework based on Benders decomposition to deal with the various cable trench problems found in the literature. Their proposed general version includes capacity and covering constraints.

Telecommunication networks are commonly structured as multilayer hierarchical networks. In this context, the p-ASP can be viewed as a two-level network design problem. The minimum ring-star problem (MRSP), where a cycle (ring) has to be defined, and vertices outside the cycle must be covered, also fits this classification. Each edge has two costs according to whether both endpoints are internal vertices or a leaf and an internal vertex. Labbé et al. [2004] proposed several valid inequalities and implemented a branch-and-cut algorithm for the problem. Simonetti et al. [2011] formulated the MRSP as a Steiner arborescence problem on a layered graph and developed a branch-and-cut algorithm based on this formulation. This formulation dominates the formulation proposed by Labbé et al. [2004]. Simonetti et al. [2011] also investigated a related problem that has not received too much attention, the minimum spanning caterpillar problem (MSCP). In the MSCP, one is looking for the minimum path such that all vertices not in the path are leaves. Integer programming formulations for the MSCP were first studied in Simonetti et al. [2009]. Baldacci et al. [2007] addressed a general version of the MRSP where multiple rings are allowed. Besides multiple rings, each ring has a maximum length. The authors suggested two formulations and assessed their correspondent branch-and-cut algorithms. Baldacci et al. [2007] also provide a nice discussion about the implementation issues of their approaches.

In this work, we focus on exact algorithms for the p-ASP. Initially, we show that due to the equal costs between intra and inter-cluster arcs in the problem, we can define the network topology using only one set of arc variables. Next, we propose two approaches to enforce backbone connectivity. The first, following an earlier approach for the problem, is based on cutset constraints. The second is based on the classic subtour elimination constraints. We show that the linear relaxations of both formulations provide the same dual limits. Furthermore, we prove that the formulation of Morais et al. [2019] is strictly contained within the proposed formulations. Although the dual bounds are stronger, our computational experiments show that the proposed formulations solve the linear programming relaxations faster. Finally, we present preprocessing procedures together with our branch-and-cut algorithms and improve the branch-and-cut algorithm of Morais et al. [2019]. For the small and medium-sized instances, our proposed algorithms outperform the previous exact algorithm in the literature. In the hard instances, there was no clear dominance between our best algorithm and the improved algorithm of Morais et al. [2019].

2.3 Mathematical formulations

Before we present our formulations, we will introduce the notation used throughout the chapter. Given a subset $S \subseteq V$, we define $\delta^+(S) = \{(i,j) \in A : i \in S \wedge j \notin S\}$ as the set of arcs leaving S, and $\delta^-(S) = \{(i,j) \in A : i \notin S \wedge j \in S\}$ as the set of arcs entering subset S. When $S = \{i\}$, we write $\delta^+(\{i\}) = \delta^+(i)$ and $\delta^-(\{i\}) = \delta^-(i)$. Also, let $A(S) = \{(i,j) \in A : i \in S \wedge j \in S\}$ be the set of arcs with both endpoints inside S. Finally, we define $\overline{S} = V \setminus S$ as the complement of subset $S \subseteq V$.

First, we describe the formulation proposed by Morais et al. [2019]. We use the model to demonstrate theoretical results about the strength of our proposed formulations. Moreover, the reproduction of the formulation here allows a more convenient reading and easy comparison. To formulate the p-ASP, we make use of the following sets of variables:

- $\{h_i \in \{0,1\} : i \in V\}$ indicates whether the vertex $i \in V$ is a cluster-head ($h_i = 1$) or not ($h_i = 0$).
- $\{y_{ij} \in \{0,1\} : (i,j) \in A\}$ expresses if an arc $(i,j) \in A$ is used as an intra-cluster arc ($y_{ij} = 1$) or not ($y_{ij} = 0$).
- $\{z_{ij} \in \{0,1\} : (i,j) \in A\}$ states whether an arc $(i,j) \in A$ is used in the backbone ($z_{ij} = 1$) or not ($z_{ij} = 0$).

Given these sets of variables, Morais et al. [2019] formulated the p-ASP as:

$$
\begin{aligned}
(\mathcal{F}_1) \qquad \mathit{min} \quad & \sum_{(i,j)\in A} c_{ij}\,(z_{ij} + y_{ij}), & & (2.1)\\
\mathit{s.t.} \quad & \sum_{i \in V\setminus\{r\}} h_i = p, & & (2.2)\\
& h_r = 1, & & (2.3)\\
& \sum_{(i,j)\in A} y_{ij} = 1 - h_i, & i \in V \setminus \{r\}, & \quad (2.4)\\
& \sum_{(i,j)\in A} z_{ij} = h_i, & i \in V \setminus \{r\}, & \quad (2.5)\\
& z_{ij} + z_{ji} + y_{ij} \le h_j, & (i,j) \in A, & \quad (2.6)\\
& \sum_{(i,j)\in \delta^-(r)} (z_{ji} + y_{ij} + y_{ji}) = 0, & & (2.7)\\
& \sum_{(i,j)\in \delta^-(r)} z_{ij} \ge 1, & & (2.8)
\end{aligned}
$$

$$\sum_{(i,j)\in\delta^+(S)} z_{ij} - \sum_{(v,j)\in A(S)} y_{vj} \geq h_v, \qquad S \subset V \setminus \{r\}, 2 \leq |S|, v \in S, \tag{2.9}$$

$$h_i \in \{0,1\}, \qquad i \in V, \tag{2.10}$$

$$y_{ij} \in \{0,1\}, \qquad (i,j) \in A, \tag{2.11}$$

$$z_{ij} \in \{0,1\}, \qquad (i,j) \in A. \tag{2.12}$$

The objective function (2.1) minimizes the total cost of making all the connections. The fixed cardinality of the backbone is imposed through (2.2), stating that p cluster-heads must be chosen. Equality (2.3) fixes the root as a cluster-head. Constraints (2.4) establish the star located around each cluster-head. If the vertex i is not defined as a cluster-head, then there is an intra-cluster arc pointing out from it. Otherwise, the vertex i has no intra-cluster outgoing arc. Each vertex defined as cluster-head has an outgoing backbone arc through constraints (2.5). Constraints (2.6) are responsible for the consistency between the h, y, and z variables. If an arc $(i,j) \in A$ is designated as an inter-cluster arc, it cannot be an intra-cluster arc at the same time. Also, if an arc $(i,j) \in A$ is selected to be in the solution, then vertex j must necessarily be a cluster-head. We use constraints (2.7) to enforce that the root r has no outgoing arc and no intra-cluster arc points to it. Moreover, constraint (2.8) requires at least one arc to reach the root. The domain of variables h, y, and z, is established through constraints (2.10)-(2.12).

The connectivity of the backbone is imposed with constraints (2.9). Morais et al. [2019] proposed two forms of enforcing backbone connectivity. The first one is a classical way of enforcing connectivity on tree problems. They used a set of constraints based on multicommodity flows that originated from each vertex directed to the root [Magnanti and Wolsey, 1995]. The second manner used to impose the connectivity was the application of exponentially many cutset constraints. Initially, they defined a standard cutset (without the term $\sum_{(v,j)\in A(S)} y_{vj}$) for the backbone. However, the two sets of arc variables allow the use of the ideas found in the work of Gendron et al. [2014] to lift these standard cutset inequalities, which resulted in constraints (2.9). We highlight that Morais et al. [2019] separated constraints (2.9) heuristically in their branch-and-cut algorithm. In Section 2.4, we describe how to separate constraints (2.9) exactly, which improves the overall performance of the algorithm.

2.3.1 Formulation with one set of arc variables

We can take advantage of the fact that there is no difference between intra and inter-cluster arc costs in the problem to formulate the p-ASP using only one set of arc variables. This fact allows us to use half of the number of arc variables from the previous formulation, which helps to tackle one of the bottlenecks in the problem, which we will discuss in Section 2.5. Our proposed formulation makes use of the variables $\{x_{ij} \in \{0,1\} : (i,j) \in A\}$, $x_{ij} = 1$ if and only if arc $(i,j) \in A$ is used in a solution, otherwise, $x_{ij} = 0$. The x variables indicate both the use of intra and inter-cluster arcs.

We still use the binary variables h defined previously to indicate if a vertex is chosen as a cluster-head or not. Given the defined variables, the p-ASP is formulated as:

$$\begin{aligned}
\min \quad & \sum_{(i,j)\in A} c_{ij}x_{ij}, && && (2.13)\\
s.t. \quad & \sum_{i\in V\setminus\{r\}} h_i = p, && && (2.14)\\
& h_r = 1, && && (2.15)\\
& \sum_{(i,j)\in A} x_{ij} = 1, && i \in V \setminus \{r\}, && (2.16)\\
& x_{ij} \le h_j, && (i,j) \in A, && (2.17)\\
& x_{ir} \le h_i, && (i,r) \in A, && (2.18)\\
& \sum_{(i,j)\in \delta^-(r)} x_{ij} \ge 1, && && (2.19)\\
& h_i \in \{0,1\}, && i \in V, && (2.20)\\
& x_{ij} \in \{0,1\}, && (i,j) \in A. && (2.21)
\end{aligned}$$

Since the x variables indicate the use of intra and inter-cluster arcs, the objective function (2.13) minimizes backbone and assignment costs. Equality (2.14) states that p cluster-heads must be chosen, and equality (2.15) defines the root as a cluster-head. Constraints (2.16) ensure that at least one arc leaves each vertex. Since the topology is a spanning reverse arborescence, the outdegree of each vertex is exactly one. Note that according to the problem definition, only vertices defined as cluster-heads can have an indegree greater than or equal to one. Therefore, as specified by constraints (2.17), if a vertex has an arc incident to it, it must be a cluster-head. Also, as constraints (2.18) state, if a vertex has an arc directed to the root, then, necessarily, this vertex is a cluster-head. At least one arc must reach the root r according to inequality (2.19). Note that equality (2.15) is implied by inequalities (2.17) and (2.19). Constraints (2.20) and (2.21) state the binary nature of variables h and x.

Formulation (2.13)-(2.21) imposes the degree constraints and the overall structure of the problem. However, it does not enforce backbone connectivity. In Figure 2.2, we show an infeasible solution for the p-ASP, but valid for formulation (2.13)-(2.21). A vertex $i \in V \setminus \{r\}$ with gray color indicates that $h_i = 1$, and white that $h_i = 0$. The root $r = 0$ is colored in black. The arcs in the figure represent the values of the x variables. We depict intra-cluster arcs with dashed lines and inter-cluster arcs with solid lines. The solution satisfies constraints (2.14)-(2.21). Despite that, it violates the restriction that the backbone obtained from the cluster-heads must establish a reverse arborescence.

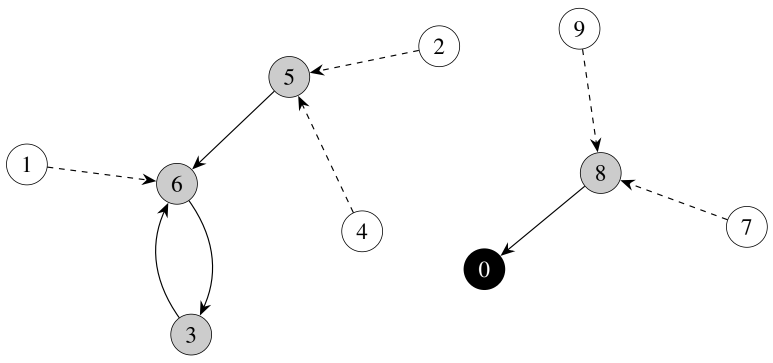


Figure 2.2: Integer feasible solution satisfying formulation (2.13)-(2.21).

In the remainder of the section, we show our approaches to enforce backbone connectivity. In addition, we prove theoretical results about the relationship of the formulations.

2.3.1.1 Cutset backbone

The first approach to impose backbone connectivity is through cutset constraints. Recall that an arborescence is a directed subgraph in which there is exactly one path between the root and any other vertex. In a reverse arborescence, we reverse all the arcs, pointing them towards the root. Therefore, the root must be reachable from any vertex in the graph. This definition naturally leads to a cutset formulation:

$$\sum_{(i,j)\in\delta^+(S)} x_{ij} \geq 1, \qquad S \subset V \setminus \{r\}, |S| \geq 2. \tag{2.22}$$

Inequalities (2.22) state that given any subset S of vertices not containing the root, there must be an arc going from S to $\overline{S}$. In other words, given any subset of vertices not including the root, at least one arc must leave the subset in order to reach the root. We refer to the formulation given by (2.13)-(2.22) as $\mathcal{F}_2$.

2.3.1.2 Subtour elimination constraint backbone

Our alternative approach for enforcing connectivity with the root is the idea that if a subset $S \subset V \setminus \{r\}$ does not reach the root r of the reverse arborescence, then there is a cycle inside S. Consequently, backbone connectivity can also be established through subtour elimination constraints:

$$\sum_{(i,j)\in A(S)} x_{ij} \leq |S| - 1, \qquad S \subset V \setminus \{r\}, |S| \geq 2. \tag{2.23}$$

While inequalities (2.22) follow from the definition of reverse arborescence, the same does not happen for constraints (2.23). Next, we prove the validity of the inequalities.

Theorem 1. *Inequalities* (2.23) *are valid for the p-ASP.*

Proof. Let $S \subset V \setminus \{r\}$ be a subset satisfying $\sum_{(i,j)\in A(S)} x_{ij} > |S| - 1$. We will show that S contains a cycle. Summing up (2.16) for every vertex $i \in S$, we obtain $\sum_{(i,j)\in A(S)} x_{ij} + \sum_{(i,j)\in \delta^+(S)} x_{ij} = |S|$. Since we supposed $\sum_{(i,j)\in A(S)} x_{ij} > |S| - 1$, we have $\sum_{(i,j)\in \delta^+(S)} x_{ij} = 0$. Thus, there are $|S|$ arcs inside S. Note that we want to show that (2.23) does not cutoff any feasible solution for the p-ASP. Therefore, we are dealing only with integral solutions.

While S has a vertex v with indegree zero, we remove it. Then, since we subtract one from each side, we have that $\sum_{(i,j)\in A(S\setminus\{v\})} x_{ij} > |S \setminus \{v\}| - 1$, and the inequality is still violated. Otherwise, all vertices inside S have indegree at least one, and according to constraints (2.17), are cluster-heads. Choose a cluster-head $v \in S$, since $\sum_{(i,j)\in \delta^+(S)} x_{ij} = 0$, because of (2.16) we know there is an $u \in S$ such that $x_{vu} = 1$. Apply this same argument iteratively. Since S is finite, at some point we will return to a vertex already considered. Therefore, S has a cycle. ■

Using Figure 2.2 as an example, after the vertices with indegree zero are removed from the component in the left, the remaining vertices 3 and 6 define a cycle. We denote by $\mathcal{F}_3$ the formulation given by (2.13)-(2.21) and (2.23).

2.3.2 Comparison of formulations

In this section, we compare the formulations for the p-ASP presented previously. Initially, we discuss that the linear relaxations of the formulations $\mathcal{F}_2$ and $\mathcal{F}_3$ provide the same dual limits. Next, we show that for any feasible fractional solution of formulation $\mathcal{F}_1$, it is possible to obtain an equivalent solution for formulation $\mathcal{F}_2$.

The result indicates that formulation $\mathcal{F}_1$ is at least as strong as formulation $\mathcal{F}_2$. To conclude, we provide a feasible fractional point for formulation $\mathcal{F}_2$ that is infeasible for formulation $\mathcal{F}_1$. This concludes the proof and demonstrates that formulation $\mathcal{F}_1$ is stronger than formulation $\mathcal{F}_2$. Since formulations $\mathcal{F}_2$ and $\mathcal{F}_3$ are equivalent, the previous results are identical for formulation $\mathcal{F}_3$.

In the minimum spanning tree problem, it is well known that models based on directed graphs produce stronger linear programming bounds than models defined in undirected graphs. For models defined by cutset and subtour elimination constraints, if both models are defined in an undirected graph, the subtour elimination constraint model provides tighter bounds [Magnanti and Wolsey, 1995]. However, in directed graphs, both models obtain the same linear programming bounds. The previous result is valid for the p-ASP. That is, formulations $\mathcal{F}_2$ and $\mathcal{F}_3$ are equivalent.

Theorem 2. *Formulations $\mathcal{F}_2$ and $\mathcal{F}_3$ are equivalent. The linear relaxation of both formulations represent the same set of fractional points.*

Proof. Given that constraints (2.13)-(2.21) are the same in both formulations $\mathcal{F}_1$ and $\mathcal{F}_2$, we have to show that, for each fractional solution of formulation $\mathcal{F}_2$ such that (2.22) is violated, then (2.23) of $\mathcal{F}_3$ is also violated, and vice-versa.

Let $S \subset V \setminus \{r\}$ be a subset of vertices used in the definition of (2.22). Summing up (2.16) for each $i \in S$ we obtain:

$$\sum_{i\in S}\sum_{(i,j)\in A} x_{ij} = |S|, \qquad \text{(summing (2.16) for each } i \in S) \tag{2.24}$$

$$\sum_{(i,j)\in A(S)} x_{ij} + \sum_{(i,j)\in\delta^+(S)} x_{ij} = |S|, \qquad \text{(using our defined notation)} \tag{2.25}$$

$$\sum_{(i,j)\in\delta^+(S)} x_{ij} = |S| - \sum_{(i,j)\in A(S)} x_{ij}, \qquad \text{(rearranging the terms)} \tag{2.26}$$

$$1 > |S| - \sum_{(i,j)\in A(S)} x_{ij}, \qquad \text{(assuming } \sum_{(i,j)\in\delta^+(S)} x_{ij} < 1\text{, i.e., (2.22) is violated)} \tag{2.27}$$

$$\sum_{(i,j)\in A(S)} x_{ij} > |S| - 1, \qquad \text{(rearranging the terms, (2.22) is violated if and only if (2.23) is violated).} \tag{2.28}$$

■

While both formulations $\mathcal{F}_2$ and $\mathcal{F}_3$ are equivalent, as we will see in Section 2.5, the overall performances of the branch-and-cut algorithms based on both formulations are different. Next, we prove that given any feasible fractional solution for formulation $\mathcal{F}_1$, it is possible to obtain a feasible solution for formulation $\mathcal{F}_2$. Since formulations

$\mathcal{F}_2$ and $\mathcal{F}_3$ represent the same set of solutions, the result is also valid for formulation $\mathcal{F}_3$.

The y and z variables are used to indicate the use of an arc according to whether its origin is a cluster-head or not. Therefore, the x variables are not used in formulation $\mathcal{F}_1$. However, for the purpose of proving the results about the formulations, we add the x variables in (2.1)-(2.12) with the following constraints:

$$x_{ij} = z_{ij} + y_{ij}, \qquad (i,j) \in A, \tag{2.29}$$

$$x_{ij} \in \{0,1\}, \qquad (i,j) \in A. \tag{2.30}$$

Also, we substitute the y and z variables in the objective function (2.1) for the x variables using the new equalities (2.29). We obtain:

$$\min \sum_{(i,j)\in A} c_{ij}\left(z_{ij} + y_{ij}\right) = \min \sum_{(i,j)\in A} c_{ij} x_{ij}. \tag{2.31}$$

Note that the value of the new objective function stays the same as the costs are equal between intra and inter-cluster arcs. We denote by $\mathcal{F}_1'$ the new formulation given by the objective function (2.31) and constraints (2.2)-(2.12) and (2.29)-(2.30).

Theorem 3. *The projection of $\mathcal{F}_1'$ in the h and x variables space is contained in $\mathcal{F}_2$.*

Proof. Let $(\overline{h}, \overline{x}, \overline{y}, \overline{z})$ be a feasible fractional solution for the linear programming relaxation of formulation $\mathcal{F}_1'$, obtained by eliminating the restriction that the decision variables h, x, y, and z, in the model (2.2)-(2.12) and (2.29)-(2.31), need to be integer. We will show that $(\overline{h}, \overline{x})$ is a feasible solution for the linear programming relaxation of formulation $\mathcal{F}_2$.

One can observe that (2.2) and (2.3) are equivalent to (2.14) and (2.15), respectively. Therefore, the solution values $\overline{h}$ also satisfy constraints (2.14)-(2.15) of the linear relaxation of $\mathcal{F}_2$. Next, for the x variables, we show that $\overline{x}$ together with $\overline{h}$ satisfy (2.16)-(2.19) and (2.22).

For constraints (2.16), if we fix a vertex $i \in V \setminus \{r\}$ and sum constraints (2.4) with constraints (2.5), we obtain $\sum_{(i,j)\in A} \overline{y}_{ij} + \sum_{(i,j)\in A} \overline{z}_{ij} = 1 - \overline{h}_i + \overline{h}_i = 1$, which means, using the linking constraint (2.29) to substitute the sum of the y and z variables, $\sum_{(i,j)\in A} \overline{x}_{ij} = 1$.

Inequalities (2.6) can be written as $x_{ij} + z_{ji} \leq h_j$. Accordingly, without modifying the inequality, we have $\overline{x}_{ij} \leq \overline{h}_j$, and constraints (2.17) is satisfied.

For inequalities (2.18), we want to show $x_{ir} \leq h_i$ for each $(i,r) \in A$. From (2.7) we have that for each $(i,r) \in A$, $\overline{y}_{ir} = 0$, and $\overline{z}_{ri} = 0$. Breaking the sum of equalities

(2.5), we have $\sum_{j \in V \setminus \{r\}} z_{ij} + z_{ir} = h_i$, then, $z_{ir} \leq h_i$. Therefore, we obtain $\overline{z}_{ir} \leq \overline{h}_i$, and since $\overline{x}_{ir} = \overline{z}_{ir} + \overline{y}_{ir}$ and $\overline{y}_{ir} = 0$, we have that $\overline{x}_{ir} \leq \overline{h}_r$.

Starting from inequality (2.8), we have that $\sum_{(i,j) \in \delta^-(r)} \overline{y}_{ij} + \sum_{(i,j) \in \delta^-(r)} \overline{z}_{ij} \geq 1$. Because adding $\sum_{(i,j) \in \delta^-(r)} \overline{y}_{ij}$ does not change the inequality. Therefore, after substituting equality (2.29), we obtain, $\sum_{(i,j) \in \delta^-(r)} \overline{x}_{ij} \geq 1$, satisfying inequality (2.19). Also, note that we have $\sum_{(i,j) \in \delta^-(r)} \overline{y}_{ij} = 0$ from equalities (2.6). Now, only inequalities (2.22) remains to be satisfied.

Let $S \subset V \setminus \{r\}$, such that $|S| \geq 2$, be a subset of vertices used in the definition of (2.9). Given a vertex $v \in S$, after rearranging the terms from equalities (2.4), we obtain $h_v = 1 - \sum_{(v,j) \in A} y_{vj}$. Hence, substituting it into the right-hand side of inequalities (2.9) for a given vertex $v \in S$, yields:

$$\sum_{(i,j) \in \delta^+(S)} z_{ij} - \sum_{(v,j) \in A(S)} y_{vj} \geq 1 - \sum_{(v,j) \in A} y_{vj}, \tag{2.32}$$

$$\sum_{(i,j) \in \delta^+(S)} z_{ij} + \sum_{(v,j) \in A} y_{vj} - \sum_{(v,j) \in A(S)} y_{vj} \geq 1, \quad \text{(rearranging the terms)} \tag{2.33}$$

$$\sum_{(i,j) \in \delta^+(S)} z_{ij} + \sum_{(v,j) \in \delta^+(S)} y_{vj} \geq 1, \quad \text{(after subtraction only arcs outgoing } S \text{ remain)} \tag{2.34}$$

$$\sum_{(i,j) \in \delta^+(S)} z_{ij} + \sum_{(i,j) \in \delta^+(S)} y_{ij} \geq 1, \quad \text{(Since } v \in S \text{, after the sum the inequality is still valid)} \tag{2.35}$$

$$\sum_{(i,j) \in \delta^+(S)} x_{ij} \geq 1, \quad \text{(using equalities (2.29))}. \tag{2.36}$$

Therefore, $\sum_{(i,j) \in \delta^+(S)} \overline{x}_{ij} \geq 1$, and inequalities (2.22) are also respected. ■

We proved that for any feasible fractional solution of the linear relaxation of formulation $\mathcal{F}_1'$ we can build a corresponding feasible solution for the linear relaxation of formulation $\mathcal{F}_2$. Since formulations $\mathcal{F}_2$ and $\mathcal{F}_3$ represent the same space of fractional solutions, the solutions built by the previous procedure are also valid for $\mathcal{F}_3$. Note that the term $\sum_{(v,j) \in A(S)} y_{vj}$ in inequalities (2.9) can be viewed as a slackness. Which might mean that formulation $\mathcal{F}_2$ contains fractional points not contained in $\mathcal{F}_1'$. We prove this result in the next theorem.

Theorem 4. *The projection of $\mathcal{F}_1'$ in the h and x variables space is strictly contained in $\mathcal{F}_2$.*

Proof. Let $(\overline{h}, \overline{x}, \overline{y}, \overline{z})$ be a fractional solution for formulation $\mathcal{F}_1'$ satisfying (2.2)-(2.8) and (2.29), that violates inequalities (2.9), such that, the subset $S \subset V \setminus \{r\}$ and the vertex $v \in S$ causing the violation satisfy: the subset S has at least three vertices

$\{t, u, v\}$ in a manner that $\overline{h}_v = 1$ and $\overline{h}_i = 0.5$, for $i \in S$ and $i \neq v$. Also, $\sum_{(t,j)\in\delta^+(S)} \overline{z}_{tj} = 0.5$, $\sum_{(t,j)\in\delta^+(S)} \overline{y}_{tj} = 0.5$, and $\sum_{(i,j)\in\delta^+(S\setminus\{t\})} \overline{z}_{ij} + \overline{y}_{ij} = 0.0$. The outflow from $S \setminus \{t\}$ must be zero. Only vertex $t \in S$ has an outflow from S greater than zero, which is divided equally between the y and z variables. Therefore, we need that $\sum_{(i,j)\in A(S\setminus\{t\})} \overline{z}_{ij} + \overline{y}_{ij} = |S \setminus \{t\}|$. Observe that without vertex u in S, we could have $\sum_{(i,j)\in\delta^+(S\setminus\{t\})} \overline{z}_{ij} > 0.0$, and the correspondent inequality (2.9) could be satisfied. Given the previous discussion, the inequality (2.9) defined is:

$$\overbrace{\sum_{(i,j)\in\delta^+(S)} z_{ij}}^{=0.5} - \overbrace{\sum_{(v,j)\in A(S)} y_{vj}}^{=0.0} \geq h_v \tag{2.37}$$

$$0.5 - 0.0 \ngeq 1.0 \tag{2.38}$$

And the inequality is violated. Using the proof of Theorem 3 to construct a solution $(\overline{h}, \overline{x})$ for $\mathcal{F}_2$, we obtain $\sum_{(t,j)\in\delta^+(S)} \overline{z}_{tj} + \sum_{(t,j)\in\delta^+(S)} \overline{y}_{tj} = \sum_{(t,j)\in\delta^+(S)} \overline{x}_{tj} = 1.0$. Then, for the inequality (2.22) defined by S, we have $\sum_{(i,j)\in\delta^+(S)} \overline{x}_{ij} \geq 1.0$. The outflow from S using the x variables for vertex t is 1. Thus, the inequality is satisfied. As we showed in the previous proof, the solution also respects (2.14)-(2.19) and (2.22). Therefore, the solution $(\overline{h}, \overline{x})$ constructed is feasible for $\mathcal{F}_2$. ■

In Figure 2.3, we show an infeasible fractional solution for formulation $\mathcal{F}_1'$ that violates inequality (2.9). In the figure, $S = \{1, 4, 5\}$, $t = 1$, $u = 5$, and $v = 4$. The solid arcs represent the z variables, and the dashed arcs represent the y variables. The value of the variable corresponding to each arc is in its label. We also indicate in the figure the value of the h variable for each vertex. If we construct a solution for formulation $\mathcal{F}_2$ using the previously described procedure, the resulting solution is feasible. Since the solution displayed in the figure is feasible for $\mathcal{F}_2$, but infeasible for $\mathcal{F}_1'$, we proved that formulation $\mathcal{F}_1'$ is stronger than formulation $\mathcal{F}_2$ and $\mathcal{F}_3$. Notice that with the $\overline{x}$ and $\overline{h}$ values fixed, there is no assignment of values for $\overline{y}$ and $\overline{z}$ such that the solution is feasible for $\mathcal{F}_1'$.

We proved previously that formulation $\mathcal{F}_1$ is stronger than formulations $\mathcal{F}_2$ and $\mathcal{F}_3$. However, although stronger, it has twice the number of arc variables. Thus, the solution of the linear programming relaxation of formulation $\mathcal{F}_1$ is expected to be more expensive in terms of computational effort than the solutions of formulations $\mathcal{F}_2$ and $\mathcal{F}_3$. As we will see in Section 2.5, the computational experiments with benchmark instances show that our proposed models solve in much shorter times the linear programming relaxations of the formulations. The results show that our tradeoff between lower

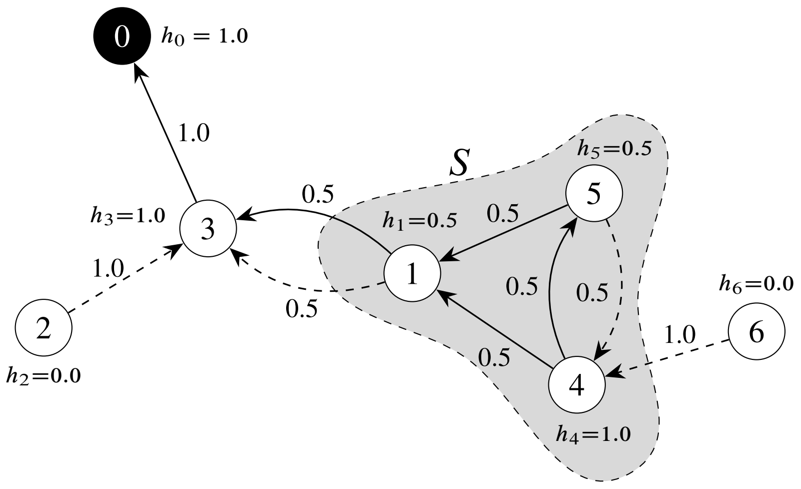


Figure 2.3: Example of a fractional solution satisfying the conditions of Theorem 4.

bound quality and speed to obtain it is advantageous, the quality of the lower bound deteriorates, but the time to calculate it improves.

2.4 Algorithms

In this section, we provide the implementation details of our proposed branch-and-cut algorithms. We use the branch-and-cut framework available through the Concert library in CPLEX 12.6 to implement the algorithms. We provide an upper bound (UB) at the beginning of the algorithms. The UB is obtained by the combinatorial Benders heuristic introduced by Morais et al. [2019].

Morais et al. [2019] implemented a heuristic based on Benders decomposition to find feasible solutions for the p-ASP. However, in the worst case, the heuristic can enumerate all feasibility cuts for the model, which is an exponential amount. Thus, a question arises if finding a feasible solution for a general p-ASP instance is a polynomial problem. Next, we prove that the decision version of the feasibility problem is NP-hard:

FEASIBILITY p-ASP
Input: A digraph $G = (V, A)$, a positive integer p, and a root $r \in V$.
Question: Is there a feasible solution for the instance?

We use a reduction from the decision version of the dominating set problem [Garey and Johnson, 1990]:

DOMINATING SET

Input: An undirected graph $G = (V, E)$ and a positive integer $k \leq |V|$.

Question: Is there a dominating set of size k or less?

Theorem 5. FEASIBILITY p-ASP *is NP-hard.*

Proof. To reduce an instance of the DOMINANT SET in the FEASIBILITY p-ASP, we build a digraph $G = (V + \{r\}, A)$ and set $p = k$. For each edge $\{i, j\} \in E$, we create two arcs, $(i, j) \in A$ and $(j, i) \in A$. Also, for every $u \in V$ we add arcs $(u, r) \in A$ and $(r, u) \in A$. Now, we have to show that a DOMINANT SET instance has a dominating set of size at most k if, and only if, the corresponding FEASIBILITY p-ASP instance built is feasible.

Suppose that the DOMINANT SET instance has a dominating set of size at most k. Therefore, there is a subset $S \subseteq V \setminus \{r\}$ of cluster-heads with cardinality p such that, for any vertex $v \notin S \cup \{r\}$, there is an arc $(v, u) \in A$ such that $u \in S$. Thus, v is covered by u. Moreover, we know that S define a reverse arborescence by construction because of the arcs (u, r) for each $u \in V$. Thus, the correspondent FEASIBILITY p-ASP instance is feasible. Furthermore, if the cardinality of subset S is less than p, we can increase it by adding leaves until $|S| = p$.

Assume, by contraposition, that the DOMINANT SET instance has no dominant set of size k or less. Observe that any subset $S \subseteq V \setminus \{r\}$ of vertices with cardinality p induces a reverse arboresce with the root r by construction. Although the reverse arborescence constraint is satisfied, there is no subset $S \subseteq V \setminus \{r\}$ with cardinality p such that all the vertices in $V \setminus S$ are covered, since it would contradict our initial assumption that there is no dominating set of size k or less. Therefore, the instance is infeasible and the answer of FEASIBILITY p-ASP is negative. ■

Note that finding a feasible solution for a p-ASP instance cannot be easier than determining whether it exists or not. Given a candidate solution for the p-ASP, we can check if the set H is a dominant set and induces a reverse arborescence in polynomial time. Therefore, FEASIBILITY p-ASP is NP-complete.

2.4.1 Preprocessing procedures

In the proposed algorithms, arcs of the form (r, i), such that $i \in V \setminus \{r\}$, do not appear in any feasible solution. Therefore, we remove all outgoing arcs from the root r. We do not remove any other types of arcs. In the minimum weight spanning arborescence problem, we can eliminate leaves from the graph and the problem remains the same. The optimal solution will not be affected as long as we add the cost of their connections.

However, the same does not apply to the p-ASP since the leaves may be necessary to reach the value p. Additionally, due to Theorem 5, the feasibility of the new instance is not guaranteed. Consequently, removing vertices in the problem is not a simple task.

Despite the difficulty of eliminating vertices in the problem, there are cases where we can say that a subset of vertices contains at least one cluster-head. Let $S \subseteq V \setminus \{r\}$ be a subset of vertices not containing the root, in the case where S must have at least one cluster-head, we define the following general valid inequalities:

$$\sum_{i \in S} h_i \geq 1, \qquad S \subseteq V \setminus \{r\}. \tag{2.39}$$

These inequalities state that at least one vertex of S is a cluster-head. If a vertex $v \in V \setminus \{r\}$ is an articulation point, then $S = \{v\}$ defines a valid inequality (2.39). One can detect articulation points using a depth-first search starting from the root r. A vertex $v \in V \setminus \{r\}$ is an articulation point if and only if v has a child u such that there is no back arc from u or any child of u to an ancestor of v [Cormen et al., 2009]. In the following, we provide several conditions where a subset S defines a valid inequality (2.39).

2.4.1.1 Vertex separator

Given a digraph $G = (V, A)$ and two vertices u and v, a subset $S \subseteq V \setminus \{u, v\}$ is called a vertex separator if the removal of S from the graph disconnects u and v. In the p-ASP, if a subset S is a vertex separator between a vertex $v \in V \setminus \{r\}$ and the root r, then S contains at least one cluster-head. Every path connecting v and the root r visits at least one vertex of S. Thus, if S does not have a cluster-head, it is impossible to connect v with the root r.

Considering the previous discussion, if S is a vertex separator between a vertex $v \in V \setminus \{r\}$ and the root r, then S defines a valid inequality (2.39). We compute the vertex separator between each vertex $v \in V \setminus \{r\}$ and the root r using the algorithm found in Even [2011]. The algorithm solves a max-flow problem on a modified graph to compute the minimum vertex separator between the pair of vertices.

2.4.1.2 Minimum weight spanning arborescence with upper bound

Let $S \subseteq V \setminus \{r\}$ be a subset of vertices that will be connected as leaves in a reverse arborescence. Initially, we remove all vertices of S from the graph and solve the minimum weight spanning reverse arborescence in the resulting graph. Then, we connect the vertices of S with their cheapest neighbor in the computed reverse arborescence. If the total cost of the arborescence is greater than a valid UB, then at least one vertex in S is cluster-head, and the subset defines a valid inequality (2.39). Note that the solution is worse than the available UB if we connect all vertices of S as leaves. Therefore, at least one of the vertices of S has to be an internal vertex to have a solution equal to or better than the one the UB provides.

To find subsets that satisfy the previous description, we randomly create 3 types of subsets with $0.25\,|V|$ vertices each. For the first type of subset, we randomly choose vertices until the desired size. In the second type, we sort the set of vertices non-increasingly by the degree. Then, we choose a random number t in the range $[0, 1]$. The element in the position $\lceil |V| \cdot t^d \rceil$ is added to S. We proceed iteratively until S is filled. The third approach performs the previous procedure, but this time sorting non-decreasingly. After creating the subsets, we check whether S or $\overline{S} \setminus \{r\}$ define a valid inequality (2.39). We run the previous procedure for `MAXIT` iterations. Based on preliminary experiments with a subset of instances from the benchmark, we set d to 5 and `MAXIT` to 250. Although in small instances a small subset size is sufficient, for hard instances it is necessary to have a size of at least $0.2\,|V|$ in order to find valid subsets for inequalities (2.39).

The use of parameter d to choose a vertex in position $\lceil |V| \cdot t^d \rceil$ from the sorted list is motivated by the work of Shaw [1998]. In the work, he uses the parameter to control which requests are removed based on a distance metric. The removal of a vertex is biased towards those that are at the beginning of the list. As Shaw [1998] discusses, the parameter d controls the degree of the bias. A high value for parameter d favors vertices at the beginning of the sorted list.

2.4.1.3 Arborescence based subset

Given a subset of vertices $S \subseteq V \setminus \{r\}$ without the root, if $S \cup \{r\}$ does not define a reverse arborescence, then $\overline{S} \setminus \{r\}$ has at least one cluster-head. Otherwise, it is impossible to visit the root from all vertices in S.

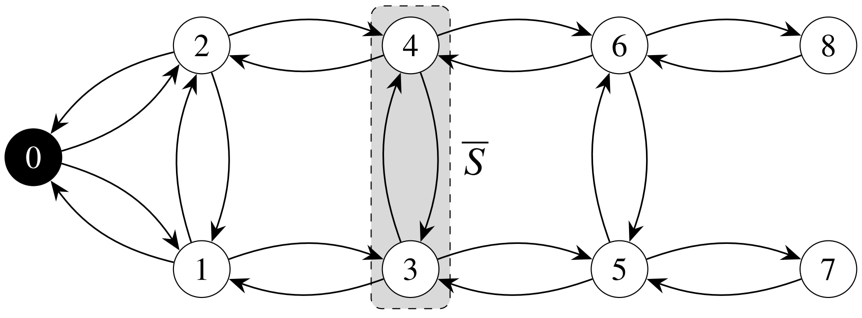


Figure 2.4: Example of a valid subset for inequalities (2.39).

Observe that $\overline{S}$ is a vertex separator, but it might be different from those computed in Section 2.4.1.1. We display an example in Figure 2.4. The subset $S = \{1, 2, 5, 6, 7, 8\}$ does not define a reverse arborescence with the root $r = 0$. Therefore, $\overline{S} \setminus \{r\} = \{3, 4\}$ is vertex separator and admits a valid inequality (2.39). Note that $\overline{S} \setminus \{r\}$ is different from the subsets $\{5\}$, $\{6\}$, $\{3, 6\}$, $\{4, 5\}$, $\{3, 2\}$, and $\{1, 4\}$, computed with the algorithm in Section 2.4.1.1.

If the procedure described in Section 2.4.1.2 does not find valid subsets S and $\overline{S} \setminus \{r\}$, we check whether the subgraphs induced by the subsets define a reverse arborescence with the root r. We perform a depth-first search and if the vertex is in the subset, we do not iterate over it recursively. At the end of the procedure, the subgraph induced by the search must define a reverse arborescence. We also check whether $\overline{S}$ defines a reverse arborescence for each vertex separator $S \setminus \{r\}$ computed in Section 2.4.1.1.

2.4.2 Initial model

The initial models for formulations $\mathcal{F}_2$ and $\mathcal{F}_3$ are both composed of the objective function (2.13) and constraints (2.14)-(2.19). The integrality restrictions (2.20) and (2.21) of the variables are removed during the solution of the root node and the enumeration of the branch-and-bound tree. We relax constraints (2.22) and (2.23) in their correspondent algorithms and dynamically add them through our separation procedures. We add in the models all valid inequalities (2.39) found during the preprocessing.

Despite relaxing constraints (2.22) and (2.23), we enumerate all inequalities for subsets of size two and three and add them in a cut pool in their respective initial models [Cordeau, 2006; Baldacci et al., 2007]. In our preliminary experiments, the version that enumerated all the previous subsets performed better than the version that did not enumerate them. Since the amount of inequalities (2.22) and (2.23) for $|S| = 2$ and $|S| = 3$ is small compared to the size of the models, we can enumerate and add them as lazy constraints ($\mathcal{O}(V^2)$ and $\mathcal{O}(V^3)$, respectively, which is smaller than the $\mathcal{O}(V^2E)$ complexity of the Dinic algorithm used to solve their separation problems). We observed that several cuts added in integral solutions were due to cycles of size two and three. Therefore, to reduce the computation time used to solve the separation problems of these simple cases and the reoptimization of such nodes, we used the cut pool mechanism.

2.4.3 Branching

As discussed by Gollowitzer and Ljubić [2011], to avoid creating imbalances in the branch-and-bound tree, it is advantageous to set branching priorities that take into account the topology of the problem. We set the highest branching priority to variables h in our branch-and-cut algorithms. Deciding which vertices are chosen as cluster-heads is more important than deciding which arcs to use in a solution. Since the backbone arcs are between cluster-heads, and the arcs inside a cluster depend on the vertex defined as cluster-head. In Section 2.5, we show the results of the algorithm without branching priorities.

We tested strong branching in our algorithms. While the small to medium-sized instances converged faster, it impacted negatively on the larger ones. We also evaluated limiting the number of simplex iterations performed on each variable in the candidate list for strong branching. Although the time taken by the algorithms decreased in this case, it was not enough to outperform the basic strategy without it. As we discuss in our results section, it is very important to solve the linear programs obtained at each node fast. Therefore, it was not beneficial to spend the extra time with strong branching.

2.4.4 Separation procedures

Given a fractional solution $(\overline{h}, \overline{x})$ for formulations $\mathcal{F}_2$ and $\mathcal{F}_3$, we construct the associated support graph $\overline{G} = (V, \overline{A})$, with vertices V and arcs $\overline{A} = A$. Each arc $(i,j) \in \overline{A}$ has a capacity equal to $\overline{x}_{ij}$ in $\overline{G}$. The exact separation procedure for inequalities (2.22) and (2.23) is the same [Padberg and Wolsey, 1983]. Note that the solution is feasible if and only if the support graph $\overline{G}$ has a one unit flow from every vertex to the root r. Therefore, the separation is carried out by solving a max-flow problem for each vertex in the support graph. For each vertex $v \in V \setminus \{r\}$, we compute the max-flow from v to r. If the value of the maximum flow is less than 1, then, by the max-flow min-cut theorem, the set S defining the minimum cut has a capacity less than 1, which means, $\sum_{(i,j)\in\delta^+(S)} x_{ij} < 1$. Likewise, we have $\sum_{(i,j)\in A(S)} x_{ij} > |S| - 1$.

If the solution in the current node of the branch-and-bound tree is integral, we can use a cheaper procedure to separate inequalities (2.22) and (2.23). We can separate them using a graph search algorithm such as breadth-first search or depth-first search. In our algorithms, we implemented a breadth-first search to find the connected components and check if there is a cycle in the solution.

During the execution of the algorithm, we separate fractional solutions only at the root node of the branch-and-bound tree. We tested separating fractional solutions on all nodes of the branch-and-bound tree, but this version performed worse in our preliminary experiments. We implemented the algorithm of Dinic to solve our max-flow problems [Dinic, 1970].

In our algorithms, we add all the cuts found during the separation phase of a node, since, as we discuss in the next section, the number of cuts found is small. We implemented a version of the algorithm that added the most violated cut in each iteration. However, this approach brought no performance gain.

As we highlighted in Section 2.3, Morais et al. [2019] used a heuristic procedure to separate inequalities (2.9) in their branch-and-cut algorithm. The separation algorithm creates the support graph taking into account only the value of the z variables. Then, a max-flow is solved from each vertex $v \in V\setminus\{r\}$ to the root r to find violated inequalities (2.9). The result is a heuristic algorithm to separate inequalities (2.9), given that the value of the term $\sum_{(v,j)\in A(S)} y_{vj}$ is not considered in the support graph when running the max-flow from v to r. Next, we show how to separate constraints (2.9) exactly.

2.4.4.1 Exact separation of the lifted cutset inequalities

In our preliminary experiments, we observed that, for some instances, our proposed formulations obtained tighter linear programming bounds than the formulation of Morais et al. [2019] at the root node of the branch-and-bound tree. This happens because they separate inequalities (2.9) heuristically in their branch-and-cut algorithm. Next, we show how to separate the inequalities exactly.

From the proof of Theorem 3, given a subset $S \subset V \setminus \{r\}$ and a vertex $v \in S$, after substituting $h_v = 1 - \sum_{(v,j)\in A} y_{vj}$ into the right-hand side of inequalities (2.9), we showed that constraints (2.9) are equivalent to:

$$\sum_{(i,j)\in\delta^+(S)} z_{ij} + \sum_{(v,j)\in\delta^+(S)} y_{vj} \geq 1, \qquad S \subset V \setminus \{r\}, 2 \leq |S|, v \in S. \tag{2.40}$$

Note that all terms in the sum are expressed as the weights of arcs leaving the subset S. Given a fractional solution $(\overline{h}, \overline{y}, \overline{z})$ for formulation $\mathcal{F}_1$, for each vertex $v \in V \setminus \{r\}$, we separate inequalities (2.40) in the following way:

1. Create the associated support graph $\overline{G} = (V, \overline{A})$, with vertices V and arcs $\overline{A}$ where each arc $(i,j) \in \overline{A}$ has capacity equal to $\overline{z}_{ij}$.

2. Increase the capacity of each arc $(v,u) \in A$ by the value $\overline{y}_{vu}$.

3. Run the max-flow from v to r. If the flow value is smaller than 1, then by the max-flow min-cut theorem, we can obtain a set S such that $v \in S$ and S violates inequality (2.40).

We modified the code of Morais et al. [2019], kindly provided by the authors, and implemented the exact separation procedure. In the next section, we compare the results of the previous version of the algorithm with the new version using our exact separation procedure.

2.5 Computational Results

In this section, we present the results from the computational experiments performed to evaluate our proposed algorithms. We refer to the branch-and-cut algorithm based on the cutset constraints formulation $\mathcal{F}_2$ as `CUTBC`, and to the algorithm based on the subtour elimination constraints formulation $\mathcal{F}_3$ as `SECBC`. The source codes from Morais et al. [2019] were kindly shared with us. Therefore, we implemented the exact separation procedure described in Section 2.4.4.1. This new branch-and-cut algorithm is denoted by `EXSEPBC`. First, we compare the algorithm of Morais et al. [2019] with our version `EXSEPBC` which includes the exact separation procedure. Then we compare the best of the two algorithms with our proposed `CUTBC` and `SECBC` algorithms.

The proposed algorithms were implemented using the C++ programming language and compiled with GCC 4.7.3 using the optimization flag -O3. We used the CPLEX 12.6 branch-and-cut framework to implement our algorithms and turned off the preprocessing, cuts, multithreading, and heuristics of the solver. We set Dual simplex as the default algorithm to use on node relaxations. The experiments were performed on an Intel Xeon E5645 hexacore machine with 2.4Ghz with a total of 32GB of RAM available running the Ubuntu 12.04 LTS operating system. All the algorithms were compiled and ran in the same computational environment. In our experiments, we set the maximum computational time to eight hours (28800 seconds).

We use the set of benchmark instances introduced by Morais et al. [2019]. The instances are generated from a complete graph $G = (V, A)$. The vertices are randomly located on a square grid with arc costs based on the euclidean distance. The cardinality p of the backbone is fixed with values from $\{4, 5, 8, 10, 15, 20, 30\}$ and the size $|V| = n$ of the vertex set with values ranging from 30 to 200. Let $S \subseteq V$ be a subset of vertices of G. We denote by $G[S] = (S, A(S))$ the subgraph of G induced by S, in which S is the vertex set and $A(S)$ is the set of arcs. In order to guarantee feasibility, a subset $W \subseteq V$ is randomly chosen with uniform probability such that $r \in W$,

Table 2.1: Computational evaluation of the impact of the exact separation procedure (results aggregated by the number of vertices).

	Morais et al. [2019]							EXSEPBC						
Vertices	Gap$_r$	Time$_r$	Gap$_f$	Time$_f$	Nodes	Cuts	Solved	Gap$_r$	Time$_r$	Gap$_f$	Time$_f$	Nodes	Cuts	Solved
30	2.18	0.26	0.00	0.81	4.14	10.71	7 / 7	0.28	0.30	0.00	0.33	0.00	10.29	7 / 7
51	2.49	0.98	0.00	5.17	50.5	7.25	8 / 8	2.10	1.72	0.00	5.74	25.25	4.00	8 / 8
76	2.50	8.38	0.00	26.35	31.17	10.17	6 / 6	2.07	12.25	0.00	32.75	17.50	4.67	6 / 6
99	4.91	20.49	0.00	124.43	216.17	33.67	6 / 6	4.76	30.04	0.00	162.68	244.00	53.67	6 / 6
101	4.99	20.40	0.00	113.07	487.14	17.71	7 / 7	4.31	46.06	0.00	166.25	395.14	10.29	7 / 7
127	1.77	31.72	0.00	140.75	133.33	20.33	12 / 12	1.59	27.57	0.00	117.56	99.42	18.17	12 / 12
152	7.17	428.43	0.11	10167.16	16722.86	181.00	6 / 7	6.55	84.76	0.00	8964.54	16282.14	610.00	7 / 7
180	13.65	412.99	0.70	17102.47	9834.20	92.40	3 / 5	11.33	838.51	0.47	17302.74	6496.20	68.60	4 / 5
200	14.26	1011.8	4.35	28800.00	16610.25	206.75	0 / 4	12.33	2455.73	4.23	28800.00	8522.25	112.75	0 / 4

$|W| = p + 1$, and the induced subgraph $G[W] = (W, A(W))$ is a connected subgraph. Then, additional arcs are randomly selected from $A \setminus A(W)$ until the desired arc density $d \in \{30\%, 40\%, 50\%, 60\%, 70\%, 100\%\}$ is achieved. The name of each instance indicates the parameters set in its definition, for example $n100p5d50$, we have $n = 100$, $p = 5$, and $d = 50\%$.

We summarize the results obtained by the algorithms of Morais et al. [2019] and EXSEPBC in Table 2.1. The values are the average of the results for all the instances in the benchmark set with the same number of vertices. The columns for each algorithm in the table represents: the average gap in the root node of the branch-and-bound tree (Gap$_r$), calculated as (BUB − LBR)/(BUB), where LBR is the lower bound obtained in such node and BUB is the best upper bound known; the average time, in seconds, taken to obtain the lower bound in the root node of the branch-and-bound tree (Time$_r$); the average final gap (Gap$_f$), computed as (BUB − BLB)/(BUB), where BLB is the best lower bound obtained by the algorithm during its execution; the average final execution time, in seconds, of the algorithm (Time$_f$); the average number of nodes investigated in the branch-and-bound tree (Nodes); the average amount of cuts added in nodes with integer solution of the branch-and-bound tree (Cuts); and the number of instances solved by each algorithm together with the total of instances in the benchmark set (Solved / Total).

Examining the results in Table 2.1, one can see that the average gap obtained in the root node of the branch-and-bound tree decreased for all instance classes. This reduction indicates that the algorithm of Morais et al. [2019] stops the optimization at the root node of the branch-and-bound tree prematurely because the heuristic separation for inequalities (2.9) finds no violated cut. Since the exact separation procedure continues beyond the point at which the heuristic separation stops, the time spent at the root node tends to increase. However, the total time spent by the EXSEPBC algorithm remained in the same order of magnitude as the algorithm of Morais et al. [2019]. Moreover, it is worth noting that the average number of nodes enumerated in

Table 2.2: Results for the variants of the branch-and-cut algorithm based on formulation $\mathcal{F}_2$.

Algorithm	Gap$_r$	Time$_r$	Gap$_f$	Time$_f$	Nodes	Cuts	Solved
`CUTBC`	5.98	61.72	0.85	3936.53	56670.89	49.76	57 / 62
`CUTBC` without preprocessing	6.00	61.14	1.08	4632.14	41491.89	46.56	55 / 62
`CUTBC` without preprocessing and branching priorities	6.00	61.84	2.20	7165.82	59057.05	40.84	47 / 62

the branch-and-bound tree decreased in the algorithm. The algorithm `EXSEPBC` solved two more instances than the algorithm of Morais et al. [2019], one with 152 vertices and another with 180 vertices. In view of the previous discussion, the algorithm `EXSEPBC` performed better. Thus, we use the results of the algorithm `EXSEPBC` in the rest of the section.

Table 2.2 displays the results of the variants of the algorithm `CUTBC`. The first column (Algorithm) indicates the algorithm considered. The following columns have the same meaning as in Table 2.1, this time considering all instances in the benchmark. From the table, we can see that preprocessing helped to solve two more instances and slightly improved the gap at the root. The average total time spent by the algorithm decreased. We suppose that inequalities (2.39) help the algorithm make better branching decisions since it has a huge influence on the performance. As we can see, the version of the algorithm where we do not set branching priorities has a worse behavior when compared to the other approaches.

We show in Tables 2.3 and 2.4 the computational results for each instance in the benchmark set to compare `EXSEPBC`, `CUTBC`, and `SECBC`. The tables have four blocks. The first one shows the name of the instance (Name) and the best known upper bound for it (BUB), if in any of the algorithms the instance has gap zero, then this value is the optimal solution. Otherwise, it is the minimum value of the best upper bounds found by the algorithms. The next three blocks contain the results for each of the algorithms. Each block has six consecutive columns, which represent the same information as in Table 2.1, but for each specific instance, instead of the average for all the instances with the same number of vertices. If the instance has 28800 in its final time column, it indicates that the time limit was reached and the optimization aborted. Consequently, optimality is not guaranteed.

Analyzing the results from Table 2.3, we observe that the algorithms `CUTBC` and `SECBC` obtained the overall best performance for these instances. Both algorithms solved the instances of up to 101 vertices in shorter times than the algorithm `EXSEPBC`, except for instances with 30 vertices, which were solved in the root node of the branch-and-bound tree by `EXSEPBC`. In some cases, the total time taken by our

Table 2.3: Detailed computational results for instances with up to 101 vertices.

Instance		EXSEPBC						CUTBC						SECBC					
Name	BUB	Gap_r	$Time_r$	Gap_f	$Time_f$	Nodes	Cuts	Gap_r	$Time_r$	Gap_f	$Time_f$	Nodes	Cuts	Gap_r	$Time_r$	Gap_f	$Time_f$	Nodes	Cuts
n30p4d50	626	1.77	0.47	0.00	0.65	0	0	7.47	0.04	0.00	0.24	11	0	7.47	0.07	0.00	0.31	11	0
n30p4d60	555	0.00	0.21	0.00	0.21	0	3	1.75	0.03	0.00	0.17	3	0	1.75	0.07	0.00	0.20	3	0
n30p4d70	472	0.00	0.17	0.00	0.17	0	0	2.69	0.02	0.00	0.13	6	0	2.69	0.04	0.00	0.24	6	0
n30p5d100	387	0.00	0.25	0.00	0.25	0	4	1.09	0.04	0.00	0.23	6	0	1.09	0.11	0.00	0.27	6	0
n30p8d50	472	0.00	0.23	0.00	0.23	0	15	3.39	0.02	0.00	0.22	43	0	3.39	0.03	0.00	0.26	44	0
n30p8d60	404	0.00	0.20	0.00	0.20	0	13	2.18	0.03	0.00	0.22	21	3	2.18	0.07	0.00	0.31	25	3
n30p8d70	372	0.19	0.56	0.00	0.62	0	37	4.67	0.04	0.00	0.53	236	7	4.67	0.09	0.00	0.51	258	5
n51p4d40	1106	6.04	0.57	0.00	4.10	18	0	7.26	0.03	0.00	0.52	36	2	7.26	0.04	0.00	0.45	30	2
n51p4d60	895	3.27	2.31	0.00	7.54	12	0	4.72	0.04	0.00	0.74	47	1	4.72	0.09	0.00	0.92	49	1
n51p4d70	754	0.00	0.14	0.00	0.14	0	0	0.73	0.03	0.00	0.27	5	1	0.73	0.08	0.00	0.37	5	1
n51p5d100	640	1.09	2.06	0.00	8.34	0	0	2.33	0.10	0.00	0.87	67	0	2.33	0.23	0.00	1.13	67	0
n51p8d30	854	1.73	0.91	0.00	2.24	7	10	3.98	0.03	0.00	0.39	36	5	3.98	0.03	0.00	0.42	46	6
n51p8d40	763	0.81	1.86	0.00	3.78	2	8	3.17	0.10	0.00	0.70	61	10	3.17	0.10	0.00	0.70	60	11
n51p8d50	699	1.13	1.52	0.00	3.79	0	3	2.99	0.04	0.00	0.39	47	2	2.99	0.04	0.00	0.58	189	8
n51p10d100	499	2.71	4.38	0.00	15.97	163	11	5.90	0.09	0.00	3.14	5281	65	5.90	0.08	0.00	3.70	6310	75
n76p5d70	1111	2.28	8.35	0.00	33.35	0	4	3.41	0.16	0.00	1.37	35	3	3.41	0.21	0.00	1.47	51	8
n76p5d60	1176	4.04	5.88	0.00	36.09	38	0	5.13	0.24	0.00	2.51	79	1	5.13	0.30	0.00	2.79	79	1
n76p5d100	934	1.82	20.14	0.00	45.88	26	0	2.81	0.38	0.00	5.22	197	3	2.81	0.46	0.00	5.42	201	4
n76p10d50	895	1.08	24.58	0.00	42.99	0	6	3.33	0.34	0.00	1.79	178	35	3.33	0.43	0.00	2.48	435	60
n76p10d60	833	1.79	7.72	0.00	15.41	21	18	3.63	0.82	0.00	3.87	750	35	3.63	1.43	0.00	6.46	777	53
n76p10d70	787	1.40	6.84	0.00	22.75	20	0	3.11	0.45	0.00	2.39	356	45	3.11	0.89	0.00	3.90	447	41
n99p5d50	3719	7.25	60.50	0.00	201.74	128	0	8.31	0.44	0.00	6.28	131	12	8.31	0.47	0.00	6.85	132	9
n99p5d60	3582	7.81	34.32	0.00	284.74	284	7	9.02	1.28	0.00	14.77	316	8	9.02	1.25	0.00	15.81	294	8
n99p5d100	2678	1.53	31.05	0.00	106.35	16	8	2.06	1.02	0.00	7.45	210	7	2.06	0.91	0.00	8.54	222	8
n99p10d50	2617	4.91	42.11	0.00	197.40	422	114	6.90	0.75	0.00	11.32	1932	146	6.90	1.34	0.00	20.72	2010	166
n99p10d60	2517	4.49	6.74	0.00	110.19	471	87	5.94	0.84	0.00	12.63	1947	110	5.94	1.49	0.00	17.07	2126	112
n99p10d70	2220	2.55	5.50	0.00	75.63	143	106	4.10	0.99	0.00	8.78	915	112	4.09	1.78	0.00	14.37	828	96
n101p5d50	1817	8.34	16.76	0.00	171.75	463	0	9.35	0.49	0.00	13.74	431	2	9.35	0.52	0.00	14.58	430	2
n101p5d60	1511	4.99	24.09	0.00	123.99	65	9	5.75	1.17	0.00	11.56	159	6	5.75	1.16	0.00	14.40	167	6
n101p5d100	1177	0.44	95.78	0.00	139.24	5	0	1.93	2.08	0.00	14.97	189	0	1.93	1.79	0.00	16.23	212	0
n101p10d40	1336	6.24	18.62	0.00	150.24	676	8	8.11	0.52	0.00	8.43	1612	122	8.11	0.92	0.00	12.50	1612	107
n101p10d50	1176	4.03	40.46	0.00	142.43	258	10	6.19	0.67	0.00	10.89	1774	71	6.19	1.19	0.00	18.26	1421	74
n101p10d60	1100	4.22	54.20	0.01	236.47	1218	7	6.32	0.68	0.00	19.73	4400	87	6.32	1.19	0.00	29.82	4476	61
n101p10d70	1005	1.92	72.49	0.01	199.62	81	38	3.58	1.99	0.00	12.55	784	65	3.66	1.76	0.00	13.30	725	80

algorithms to solve the instance was smaller than the time spent by EXSEPBC solving the linear programming relaxation obtained at the root node. Despite the short execution times, the lower bounds obtained in the root are weaker, which was expected. However, we can see that the tradeoff between the lower bound quality and the total time spent by the proposed algorithms to solve the linear programming relaxation is advantageous. For this set of instances, the time necessary by algorithm SECBC to solve the instances was slightly worse than algorithm CUTBC.

We report in Table 2.4 the results for the larger instances. For instances up to 127 vertices, the behavior observed for smaller instances remains, our algorithms can solve them in shorter execution times. For instances with more than 152 vertices, the number of nodes in the branch-and-bound tree grows significantly compared to smaller instances, and the previous dominance for the runtime is no longer observed. The algorithm CUTBC cannot solve one instance solved by EXSEPBC in the set of instances with 152 vertices. Nevertheless, CUTBC is capable of solving one instance with 200 vertices (n200p10d50) that EXSEPBC cannot. For instances that remain unsolved, the best UBs were obtained by our proposed algorithms CUTBC and SECBC. The algorithm CUTBC obtained the best primal bounds for instances n180p15d60, n200p10d70, and n200p20d40. The algorithm SECBC obtained the best UB for n200p15d50. The algorithm EXSEPBC obtained the best dual bounds, certainly due to the stronger formulation.

One can notice in Tables 2.3 and 2.4, for all instances in the benchmark, that the time taken to solve the linear programming relaxation in the root node is smaller for the algorithms CUTBC and SECBC. For instances with 152 vertices, although the times to solve the linear relaxation of the root node are shorter, CUTBC solved one instance less than EXSEPBC and SECBC three less than EXSEPBC. We suppose this is because some instances have the highest values for p. The difficulty of the algorithms can be observed by the explosion of the number of nodes. Regarding the time to solve the instances, our algorithms achieve the shortest times for small and medium-sized ones.

In Table 2.5, we aggregate the results obtained by the algorithms CUTBC and SECBC by the number of vertices. The table columns represent the same information as the columns of Table 2.1. Although formulations $\mathcal{F}_1$ and $\mathcal{F}_2$ are equivalent, their correspondent branch-and-cut algorithms show different performances. The algorithm CUTBC solves three more instances of the benchmark than the algorithm SECBC. Moreover, the time required by CUTBC to solve the linear programming relaxation obtained in the root node of the branch-and-bound tree is better.

Comparing Tables 2.1 and 2.5, for instances with 180 and 200 vertices, the algorithms CUTBC and SECBC obtain tighter gaps in the root node than the algorithm of

Table 2.4: Detailed computational results for instances with up to 200 vertices.

Instance		EXSEPBC						CUTBC						SECBC					
Name	BUB	Gap_r	$Time_r$	Gap_f	$Time_f$	Nodes	Cuts	Gap_r	$Time_r$	Gap_f	$Time_f$	Nodes	Cuts	Gap_r	$Time_r$	Gap_f	$Time_f$	Nodes	Cuts
n127p10d50	236637	1.46	16.67	0.00	70.36	11	16	1.88	0.46	0.00	2.71	23	0	1.88	0.94	0.00	3.93	23	0
n127p10d60	228766	2.76	37.93	0.00	213.89	386	5	3.65	0.55	0.00	11.12	693	6	3.65	1.05	0.00	17.07	785	6
n127p10d70	203322	1.93	30.40	0.00	161.21	64	0	2.60	0.86	0.00	9.26	222	2	2.60	1.45	0.00	10.91	249	2
n127p10d100	176740	0.84	13.46	0.00	56.79	0	50	1.53	2.95	0.00	18.08	304	0	1.53	4.22	0.00	25.03	333	0
n127p15d40	212200	2.22	12.56	0.00	73.75	109	0	3.85	0.58	0.00	7.63	1503	44	3.85	1.11	0.00	12.46	1630	41
n127p15d50	201391	2.56	19.23	0.00	100.54	188	18	3.98	0.75	0.00	9.38	931	35	3.98	1.19	0.00	13.03	992	35
n127p15d60	179339	1.71	20.86	0.00	89.36	89	26	2.80	1.59	0.00	9.74	495	19	2.80	2.32	0.00	11.56	483	18
n127p15d70	170419	1.74	41.11	0.00	192.27	253	0	3.58	1.91	0.00	35.52	3879	27	3.57	2.76	0.00	61.32	4521	33
n127p20d40	192822	0.79	8.60	0.00	35.23	3	30	2.65	0.69	0.00	7.47	1168	82	2.65	0.78	0.00	6.49	785	78
n127p20d50	173547	1.37	33.41	0.00	143.04	52	25	3.46	3.35	0.00	47.61	4420	48	3.46	4.17	0.00	59.49	4614	77
n127p20d60	159549	1.12	37.51	0.00	140.06	38	6	3.00	1.44	0.00	24.50	3186	53	3.00	1.73	0.00	20.30	1606	32
n127p20d70	148547	0.53	59.14	0.00	134.23	0	42	2.24	4.96	0.00	49.04	1802	21	2.24	7.06	0.00	32.13	1028	51
n152p15d40	219134	7.78	126.89	0.00	2538.15	4122	54	10.55	57.73	0.00	2213.75	24081	150	10.55	74.46	0.00	5211.97	27146	145
n152p15d60	163976	6.90	74.93	0.00	7704.18	9498	294	8.63	50.01	0.00	2195.53	24783	82	8.63	61.77	0.00	4900.37	21240	93
n152p15d70	143867	4.38	151.41	0.00	2366.98	1078	275	5.80	73.72	0.00	536.97	6369	15	5.80	86.62	0.00	2003.79	8522	10
n152p20d30	232151	7.47	25.78	0.00	2326.42	8719	62	11.33	14.85	0.00	6055.55	210720	269	11.33	17.71	0.00	15480.79	194717	236
n152p20d40	181375	7.67	36.72	0.00	16354.81	45193	266	11.72	36.79	7.91	28800.00	646561	159	11.72	45.41	9.34	28800.00	288609	194
n152p20d70	112631	6.53	143.23	0.00	17063.59	14008	1552	8.78	66.99	0.00	17973.21	480999	80	8.78	80.58	6.74	28800.00	298471	60
n152p30d30	177131	5.08	34.36	0.00	14397.62	31357	1767	8.46	16.57	0.00	20950.12	1313027	269	8.46	16.98	5.98	28800.00	965724	331
n180p5d60	205784	17.17	873.15	0.00	17102.17	4094	0	19.20	211.79	0.00	5106.65	4799	3	19.20	213.00	0.00	5820.58	4688	3
n180p5d70	173597	12.34	288.83	0.00	7609.61	1121	10	13.89	824.21	0.00	9515.58	3887	1	13.89	824.70	0.00	10799.55	4355	1
n180p10d50	116841	9.84	607.04	0.00	11278.83	7335	17	11.40	225.04	0.00	5141.59	18173	54	11.40	210.55	0.00	9440.24	18520	80
n180p10d70	90532	7.86	1350.79	0.00	21723.07	9454	90	8.72	83.48	0.00	12169.89	28473	23	8.72	164.27	0.00	13640.60	33543	33
n180p15d60	78788	9.46	1072.74	2.32	28800.00	10477	226	10.45	342.57	9.53	28800.00	166123	118	9.74	707.05	8.84	28800.00	34042	68
n200p10d50	146060	13.68	1444.83	2.76	28800.00	5400	62	15.68	143.08	0.00	17794.71	67711	120	15.69	135.42	11.14	28800.00	55971	100
n200p10d70	108766	12.37	4308.28	5.43	28800.00	2878	119	12.15	1201.02	11.49	28800.00	68639	64	12.15	1526.32	11.49	28800.00	24336	63
n200p15d50	101713	11.16	2924.53	4.11	28800.00	8211	124	11.77	282.96	10.69	28800.00	144902	194	11.77	297.16	10.69	28800.00	40719	171
n200p20d40	101520	12.13	1145.27	4.61	28800.00	17600	146	13.56	159.56	12.85	28800.00	263421	181	13.56	168.41	12.85	28800.00	64416	177

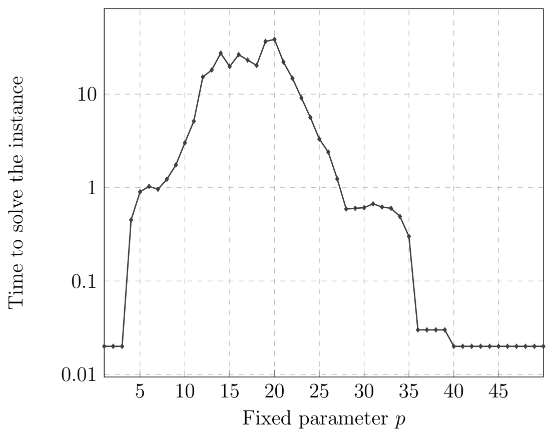
Table 2.5: Average results aggregated by the number of vertices for our proposed algorithms.

	`CUTBC`							`SECBC`						
Vertices	Gap$_r$	Time$_r$	Gap$_f$	Time$_f$	Nodes	Cuts	Solved	Gap$_r$	Time$_r$	Gap$_f$	Time$_f$	Nodes	Cuts	Solved
30	3.32	0.03	0.00	0.25	46.57	1.43	7 / 7	3.32	0.07	0.00	0.30	50.43	1.14	7 / 7
51	3.89	0.06	0.00	0.88	697.50	10.75	8 / 8	3.89	0.09	0.00	1.03	844.50	13.00	8 / 8
76	3.57	0.40	0.00	2.86	265.83	20.33	6 / 6	3.57	0.62	0.00	3.75	331.67	27.83	6 / 6
99	6.06	0.89	0.00	10.21	908.50	65.83	6 / 6	6.05	1.21	0.00	13.89	935.33	66.50	6 / 6
101	5.89	1.09	0.00	13.12	1335.57	50.43	7 / 7	5.90	1.22	0.00	17.01	1291.86	47.14	7 / 7
127	2.93	1.67	0.00	19.34	1552.17	28.08	12 / 12	2.93	2.40	0.00	22.81	1420.75	31.08	12 / 12
152	9.32	45.24	1.13	11246.45	386648.57	146.29	6 / 7	9.32	54.79	3.15	16285.27	257775.57	152.71	4 / 7
180	12.73	337.42	1.91	12146.74	44291.00	39.80	4 / 5	12.59	423.91	1.77	13700.19	19029.60	37.00	4 / 5
200	13.29	446.66	8.76	26048.68	136168.25	139.75	1 / 4	13.29	531.83	11.54	28800.00	46360.50	127.75	0 / 4

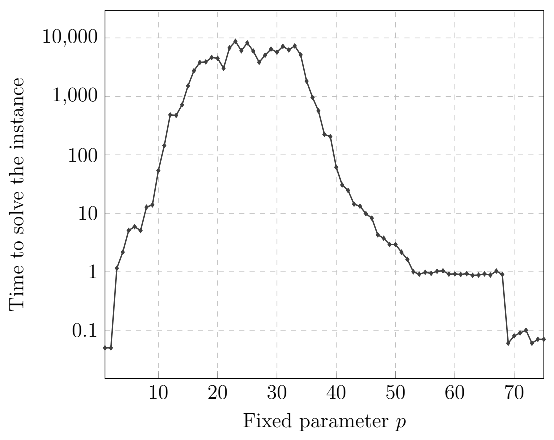
Morais et al. [2019]. Besides, the proposed algorithm `EXSEPBC` with the exact separation achieves the best gaps in the root node for all the instances. Regarding the performance, the algorithms displayed in Table 2.5 have better execution times than the algorithms in Table 2.1 for small and medium-sized instances.

For some instances, despite a large number of nodes, the number of added cuts is small, which indicates that little time is spent reoptimizing the nodes of the branch-and-bound tree. Consequently, in the problem, most of the time is spent solving the relaxations. As we discussed about the implementation details in Section 2.4, focusing on solving the linear programming problems fast is an efficient strategy since the integer solutions obtained are feasible most of the time. The small amount of added cuts is an indication that it is easy to guarantee feasibility in the problem, allowing approaches with special focus on solving the linear programming relaxations quickly.

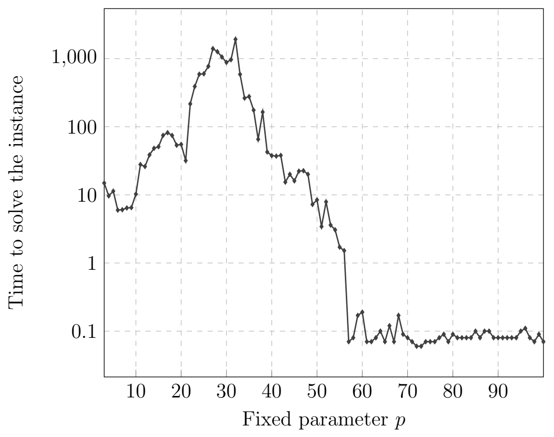
In Figure 2.5, we perform a study about the impact of the variation of parameter p in the difficulty of the problem. We used benchmark instances and varied the value of p up to n. The subcaption of each graphic describes the instance used in the experiment. We display the time (seconds) necessary to solve the instance using a logarithmic scale. From the graphics, we observe that the times required to solve the instances increases as the value of p grows until it reaches a peak. Then, the time starts to decrease until it is negligible. As p approaches n, the problem begins to resemble a minimum weight spanning arborescence. One can observe from the graphics that while for low values of p the algorithm easily solves the instances, for greater values, they become very difficult. When p is approximately 30% of the number of vertices, the instances reach the greatest difficulty. Besides, from the moment that p is at least 60% of n, the instances become easier, and it takes a few seconds to solve them.



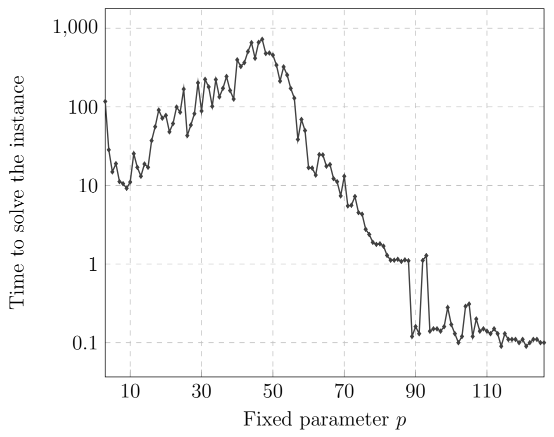
(a) 51 vertices and density 100% (n51p10d100).



(b) 76 vertices and density 100% (n76p5d100).



(c) 101 vertices and density 60% (n101p5d60).



(d) 127 vertices and density 60% (n127p10d60).

Figure 2.5: Time to solve the instance as a function of the fixed paramater p.

2.6 Conclusion

This work addressed the p-arborescence star problem (p-ASP). Initially, we presented two new formulations for the problem that differ in the way they impose connectivity. Then, we proved theoretical results about the relationship of existing formulations for the p-ASP. Based on both formulations, we implemented two exact branch-and-cut algorithms. Moreover, we showed that finding a feasible solution for an arbitrary p-ASP instance is NP-hard and introduced preprocessing procedures for the problem. Also, for the branch-and-cut algorithm found in the literature for the p-ASP, we developed an exact separation procedure for an inequality that was separated using a heuristic method, and implemented the new separation algorithm into the existing branch-and-cut algorithm.

All the algorithms were evaluated and compared using a set of benchmark instances for the p-ASP. The new exact separation algorithm improved the existing branch-and-cut algorithm from the literature. For small and medium-sized instances, our proposed algorithms obtained the best results, being able to solve instances with shorter execution times. For the larger instances, there was no clear dominance between our best algorithm and the improved algorithm.

As we discussed in the computational experiments section, new strategies for addressing the problem may involve formulations and methods to quickly solve the linear programming problems obtained in the nodes of the branch-and-bound tree. One approach we want to investigate is to apply Lagrangian relaxation to existing formulations for the problem. It is worth mentioning that the problem is also an interesting field for heuristic methods since the existing Benders decomposition based heuristic for the problem obtains weak primal bounds. Besides, the problem of obtaining a feasible solution for the p-ASP is NP-hard. Thus, developments in heuristics can also contribute to advances in exact algorithms for the problem. From our computational experiments, many benchmark instances remain unsolved. Therefore, the problem is interesting to study and propose new solution methods.

Chapter 3

The pickup and delivery traveling salesman problem with multiple stacks

The focus of this work is the study of valid inequalities and exact approaches for the pickup and delivery traveling salesman problem with multiple stacks. In the problem, a single vehicle must fulfill a set of client requests. Each request states that the vehicle must pick up an item at a given location, and later deliver it to another particular location. Inside the vehicle, items are stored in horizontal stacks of limited capacity. The loading and unloading operations of items follow the last-in-first-out policy. Every item is picked up on top of a stack and can only be delivered if it is also on top of its stack. The goal of this problem is to find a vehicle route of minimum cost. We propose a new formulation along with new valid inequalities for the problem. Several of these valid inequalities are lifted versions of inequalities previously proposed in the literature. The branch-and-cut algorithm based on the formulation and the valid inequalities is compared with the state-of-the-art solution methods for the problem. Computational experiments performed on benchmark instances show that the implemented algorithm outperforms all other competing exact algorithms for this problem.

3.1 Introduction

We define the problem on a complete digraph $G = (V, A)$ where $V = \{0, 1, \ldots, 2n+1\}$ is the vertex set and $A = \{(i, j) : i \in V \wedge j \in V \wedge i \neq j\}$ is the set of arcs. The subsets $P = \{1, \ldots, n\} \subset V$ and $D = \{n+1, \ldots, 2n\} \subset V$ represent the sets of pickup and delivery locations, respectively. Vertices 0 and $2n+1$ denote, respectively, the initial and final depots of the route. A nonnegative travel cost c_{ij} is associated with the use of each arc $(i,j) \in A$. A single vehicle must visit each location satisfying n customer requests. Each customer request establishes to pick up an item of size $d_i \geq 0$ at location $i \in P$ and deliver it at location $n+i \in D$. Besides, we refer to the item picked up at location $i \in P$, and delivered at $n+i \in D$, as item i. The vehicle contains a set M of stacks, each one of capacity Q, used to load and unload the items. The loading and unloading operations in each stack must follow the last-in-first-out (LIFO) policy. Every item is loaded on top of a stack and can only be delivered if it is on top of its stack. The aim of the pickup and delivery traveling salesman problem with multiple stacks (PDTSPMS) is to find a minimum cost route, starting at depot 0 and ending at depot $2n+1$, satisfying all customer requests.

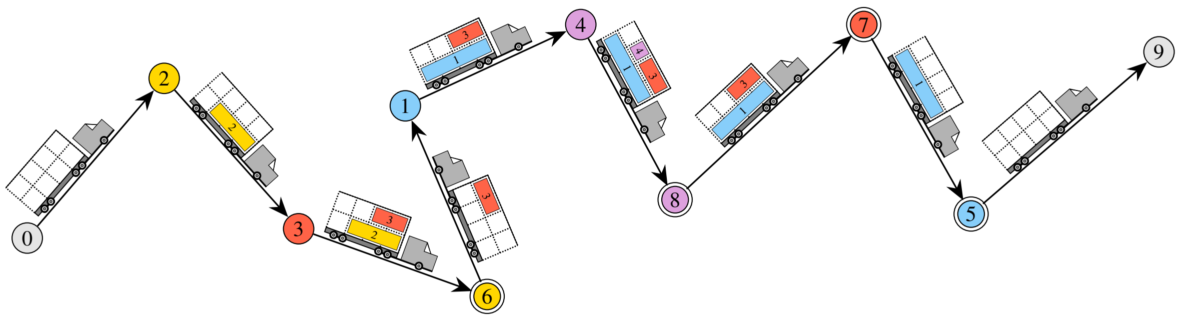


Figure 3.1: Diagram showing a feasible PDTSPMS route.

Figure 3.1 displays a feasible PDTSPMS solution for an instance with $n = 4$ requests, pickup set $P = \{1,2,3,4\}$, delivery set $D = \{5,6,7,8\}$, and initial and final depots equal to 0 and 9, respectively. The vehicle contains a set $M = \{1,2\}$ of identical stacks with capacity $Q = 4$. The item sizes are $d_1 = 4$, $d_2 = 3$, $d_3 = 2$, and $d_4 = 1$. Above the arcs, we show the loading compartment of the vehicle after visiting each route location. The stacks are drawn with dashed lines and ordered from bottom to top. The pickup location and its corresponding delivery location have the same color in the route, with the deliveries possessing a double circle around it. We use this same convention throughout the text. The vehicle starts from depot 0 visiting pickup 2 and loading its item in stack 1. Then, it visits location 3 and loads its corresponding item into stack 2. Note that the capacity constraint would be violated if item 3 was loaded in the same stack as item 2. Soon after, the vehicle delivers item 2 and afterward loads item 1 in the same stack. Next, item 4 is loaded in the second stack. Due to the LIFO policy, item 3 cannot be delivered since it is not at the top of its stack. Finally, the vehicle delivers the remaining items and moves to the final depot.

The PDTSPMS models situations where the loading and unloading of items can only be carried out from the rear of the vehicle. Hence, only the last item loaded in each stack is available for delivery and must be delivered before all the other items in the same stack. The constraint that only the last loaded item is available for delivery avoids the reorganization of items inside the loading compartment, as this might incur in delays and costs. Moreover, reorganizing items outside the depot is costly, as items can be large, heavy, fragile, or dangerous. In the situation where it might be possible to relocate items, they must be small packages. If they are large, once the vehicle is outside the depot, there may be no equipment or place to move them. Within this context, it is important to design the vehicle route to avoid such complications.

Ladany and Mehrez [1984] addressed a practical application of the problem faced by a freight company where a single truck collects cargos in Tel Aviv and delivers it in Haifa. The truck can be loaded only through the rear door, and the internal area is narrow. Therefore, the loading and unloading operations follow the LIFO policy. Levitin and Abezgaouz [2003] studied an application of the PDTSPMS arising in a warehouse where cargos are placed on pallets, and each pallet is placed on top of the previous one by an automated guided vehicle (AGV). Automated guided vehicles are widely used in automated industries to move pallet batches between different locations. To avoid wasting time and space, the AGV route is planned to prevent the rearrangement of items. These routes follow the LIFO policy to avoid the excessive use of space and time to rearrange the pallets in a batch.

The main contribution of our work is a new branch-and-cut algorithm that incorporates new valid inequalities and separation strategies for the PDTSPMS. We provide separation algorithms for our proposed inequalities and new separation procedures for inequalities previously used in the problem. Additionally, we introduce several strategies that are integrated into our algorithm and improve its overall performance. The proposed algorithm is compared with the state-of-the-art exact algorithms for the problem in the same computational environment. The algorithm outperforms all other algorithms for all instances in the benchmark classes. Our branch-and-cut obtains shorter times and solves 20 more instances in total from the benchmark, several of which for the first time. In addition to obtaining better times and solving more instances, our algorithm achieves stronger dual bounds for all instance families in the benchmark and closes the performance gap between unitary demand sized instances and non-unitary demand sized instances.

The remainder of the text is structured as follows. In Section 3.2, we review the existing PDTSPMS literature. We describe our defined notation and present a new linear integer programming formulation in Section 3.3. After presenting the formulation, in Section 3.4, we present new valid inequalities for the PDTSPMS. Next, in Section 3.5, we describe our implemented branch-and-cut algorithm. In Section 3.6, we report the computational results for the benchmark instances and compare all the branch-and-cut algorithms for the problem. Finally, we state our concluding remarks in Section 3.7.

3.2 Literature review

The PDTSPMS first appeared in the literature in a particular case with only one stack called pickup and delivery traveling salesman problem with LIFO loading (PDTSPL). Carrabs et al. [2007b] studied local search operators for the PDTSPL. They implemented three local search operators in a variable neighborhood search (VNS) heuristic. Carrabs et al. [2007a] introduced the first exact algorithm for the PDTSPL, a branch-and-bound algorithm that uses additive lower bounds based on assignment and arborescence relaxations. They also used elimination rules to remove arcs that cannot belong to feasible solutions. Cordeau et al. [2010] investigated the PDTSPL with a special focus on the uncapacitated case. They analyzed the structure of the problem and presented the first formulations with constraints for the LIFO policy and capacity. Moreover, they introduced several valid inequalities that were incorporated into their three branch-and-cut algorithms. The authors compared their algorithm with the one of Carrabs et al. [2007a] and remarked that the proposed algorithm is capable of solving the instances in less computational time. Li et al. [2011] showed how to represent PDTSPL solutions in a tree structure rather than a list of vertices. They adapted the operators of Carrabs et al. [2007b] and introduced three new operators based on the tree representation. These neighborhood operators are used in a VNS heuristic. The computational experiments showed that the implemented heuristic obtains better quality solutions than the one of Carrabs et al. [2007b].

Petersen and Madsen [2009] introduced the double traveling salesman problem with multiple stacks (DTSPMS). The DTSPMS is a particular variant of the PDTSPMS in which all pickup operations must be completed before any delivery operation can take place. Petersen and Madsen [2009] implemented four heuristics based on local search for the DTSPMS. The authors also provided a compact flow formulation for the problem. Petersen et al. [2010] presented several formulations and exact approaches for the DTSPMS. The algorithm that obtained the best results decomposes the DTSPMS into two levels. At the upper level, the algorithm obtains the pickup and delivery routes separately for the problem. At the lower level, it checks whether there is a feasible loading plan for the two routes together. Ángel Felipe et al. [2009] presented a hybrid VNS heuristic along with neighborhood operators. The VNS heuristic uses the operators proposed by Petersen and Madsen [2009] and four operators introduced by the authors. The computational experiments showed that it obtains better results than the heuristic of Petersen and Madsen [2009]. Lusby et al. [2010] designed an exact algorithm that pairs the best routes for the two regions iteratively. Casazza et al. [2012] studied theoretical properties of the DTSPMS and proposed polynomial algorithms for subcases of the problem. Carrabs et al. [2013] implemented an additive branch-and-bound algorithm for the DTSPMS considering that the vehicle contains only two stacks. Alba Martínez et al. [2013] presented several valid inequalities which were integrated into a branch-and-cut algorithm. The branch-and-cut algorithm outperformed the previous exact algorithms for the DTSPMS considering the set of benchmark instances. Urrutia et al. [2015] proposed a two-stage heuristic for the DTSPMS. The computational results showed that the heuristic is competitive with the other existing ones for the problem. Barbato et al. [2016] devised a branch-and-cut algorithm that uses several valid inequalities for the case where the vehicle has only two stacks.

The PDTSPL was generalized to multiple vehicles before the PDTSPMS in the literature. Eventually, the PDTSPMS also received a version with multiple vehicles. Cheang et al. [2012] extended the PDTSPL by considering multiple vehicles and a limitation on the total distance traveled by each vehicle. They proposed several neighborhood operators and devised a two-stage heuristic for the problem. It is worth noting that, unlike the PDTSPL, the capacity of the stack in each vehicle is unlimited in their work. Cherkesly et al. [2014] studied exact algorithms for a version of the PDTSPL with multiple vehicles and time windows. Differently from Cheang et al. [2012], there is no constraint on the maximum length of the routes and the stacks have finite capacities. They proposed three branch-price-and-cut algorithms and adapted valid inequalities from the literature. Similarly, Benavent et al. [2015] extended the PDTSPL by adding multiple vehicles and maximum route length. They provided two formulations and a multi-start heuristic based on tabu search for their problem variant. For the formulation with exponentially many constraints, they developed a branch-and-cut algorithm. Cherkesly et al. [2016] introduced a PDTSPMS variant with multiple vehicles and time windows. The authors proposed two branch-price-and-cut algorithms to tackle their version of the problem. Veenstra et al. [2017b] introduced handling costs in the PDTSPL. If an item is not on top of its stack, additional handling operations are allowed to unload and reload items blocking its access. They implemented a large neighborhood search (LNS) heuristic to solve the problem. Veenstra et al. [2017a] added multiple vehicles, time windows, and handling operations, in the standard definition of the PDTSPL. The difference to the problem studied by Veenstra et al. [2017b] is the incorporation of multiple vehicles and time windows. The authors developed branch-price-and-cut algorithms for the two rehandling policies analyzed.

Côté et al. [2012a] performed the first study regarding exact algorithms and valid inequalities for the PDTSPMS. They proposed three formulations for the problem. Additionally, they adapted valid inequalities from other problems to the PDTSPMS and proposed new ones. These valid inequalities were analyzed and employed in a branch-and-cut algorithm. The best algorithm is based on the formulation that depends on the resolution of an NP-hard packing problem to guarantee the feasibility of a solution. When the algorithm finds a route that satisfies the connectivity and precedence, the packing problem is called to check whether there is a feasible assignment of stacks for the items such that the capacity and LIFO policy are satisfied. If the problem is infeasible, the algorithm adds a constraint prohibiting this infeasible route. Since the algorithm of Côté et al. [2012a] was the first exact algorithm for the problem, they compared it with existing exact approaches for the DTSPMS. The results showed that the branch-and-cut obtains superior results for the DTSPMS instances even though it was not designed specifically for the problem. Sampaio and Urrutia [2017] presented a formulation for the PDTSPMS that uses a set of variables to indicate the stack operations of the vehicle in the arcs of the graph. Thus, instead of solving a packing problem to guarantee the feasibility of a route like Côté et al. [2012a], they model the capacity and LIFO policy using this set of variables. Since the formulation is based on several sets of exponentially many constraints, Sampaio and Urrutia [2017] implemented a branch-and-cut algorithm. The algorithm is executed in a different computational environment than the algorithm of Côté et al. [2012a]. Therefore, it was not possible to make a direct comparison between them. Pereira and Urrutia [2018] conducted a study about formulations and exact algorithms for the PDTSPMS. The authors proposed three new formulations along with valid inequalities for the problem. The first formulation is an adaptation of the formulation of Sampaio and Urrutia [2017] where a set of variables is eliminated. The second formulation is a combination of the models of Côté et al. [2012a] and Sampaio and Urrutia [2017], and the third formulation is a compact model. They implemented three branch-and-cut algorithms based on each of the formulations. The algorithms incorporate their proposed valid inequalities and the inequalities already used for the PDTSPMS. The proposed branch-and-cut algorithms are compared to the algorithms of Côté et al. [2012a] and Sampaio and Urrutia [2017] in the same computational environment. The algorithms implemented by Pereira and Urrutia [2018] are capable of solving the instances in a shorter computational time. Moreover, within the one hour limit, the algorithms were able to solve more instances of the benchmark than the algorithms of Côté et al. [2012a] and Sampaio and Urrutia [2017].

In this work, we present a new formulation for the PDTSPMS along with new valid inequalities. Some of these new valid inequalities are lifted versions of inequalities previously proposed in the literature. Then, we implement a branch-and-cut algorithm based on the formulation that includes the new valid inequalities. Besides, we introduce new separation procedures and implement several strategies which improve the overall performance of our branch-and-cut algorithm. Our proposed algorithm closes the performance gap that existed previously for the other algorithms between the two classes of instances in the benchmark. We compare our algorithm in the same computational environment with all the other existing algorithms for the PDTSPMS. The new algorithm outperforms the other algorithms for all the benchmark instances and solves several instances for the first time.

3.3 Mathematical formulation

In order to formulate the problem mathematically, we define the set $\{x_{ij} \in \{0,1\} : (i,j) \in A\}$ of binary variables, which indicate whether an arc $(i,j) \in A$ belongs to the route $(x_{ij} = 1)$ or not $(x_{ij} = 0)$, and the set of binary variables $\{y_i^k \in \{0,1\} : i \in P \wedge k \in M\}$, which indicate whether the item $i \in P$ is loaded into stack $k \in M$ $(y_i^k = 1)$ or not $(y_i^k = 0)$. To define our model, we associate an item of size $-d_i$ with each delivery location $n+i \in D$, and items $d_0 = d_{2n+1} = 0$ with the depots. Given two subsets $S \subseteq V$ and $T \subseteq V$, such that $S \cap T = \emptyset$, we define $\overline{S} = V \setminus S$ as the complement of set S, $x(S) = \sum_{i \in S} \sum_{j \neq i, j \in S} x_{ij}$ as the sum of the arc variables inside set S, and $x(S,T) = \sum_{i \in S} \sum_{j \in T} x_{ij}$, as the sum of the arc variables in the cut between the sets S and T. Let $\ddagger$ denote the collection of all subsets $S \subset V$, such that $0 \in S$, $2n+1 \notin S$, and there is a pickup $i \in P$ such that $i \notin S$ and $n+i \in S$. Finally, define the function $\gamma\colon P \cup D \to P$, $\gamma(i) = i$ for all $i \in P$, and $\gamma(n+i) = i$ for all $n+i \in D$. For simplicity, we use i when we refer to the set $\{i\}$, $i \in V$.

Taking into account the previous definitions, the PDTSPMS is formulated as follows:

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

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

$$
\begin{aligned}
&x(V \setminus \{i\}, i) = 1, && i \in P \cup D \cup \{2n+1\}, && (3.3)\\
&x(S, \overline{S}) \geq 1, && S \subset P \cup D, |S| \geq 2, && (3.4)\\
&x(S, \overline{S}) \geq 2, && S \in \ddagger, && (3.5)\\
&\sum_{k \in M} y_i^k = 1, && i \in P, && (3.6)\\
&y_i^k + y_j^k + x(S) + x(S, n+i) \leq |S| + 1, && S \subset P \cup D, i, j \in P, j \in S, && (3.7)\\
& && i, n+i, n+j \notin S, k \in M, &&\\
&x(S, \overline{S}) \geq \frac{\sum_{u \in S} d_u y_{\gamma(u)}^k}{Q}, && S \subset P \cup D, k \in M, && (3.8)\\
&y_i^k \in \{0, 1\}, && i \in P, k \in M, && (3.9)\\
&x_{ij} \in \{0, 1\}, && (i, j) \in A. && (3.10)
\end{aligned}
$$

The objective function (3.1) minimizes the total cost of the route. According to equalities (3.2) and (3.3), the vehicle leaves and enters each vertex once, respectively. Inequalities (3.4) are the standard cutset constraints and guarantee the connectivity of the route by forbidding subtours. Inequalities (3.5) establish the precedence relationship between a pickup and its corresponding delivery. The inequalities forbid any path from depot 0 which visits a delivery before visiting its corresponding pickup. Equalities (3.6) require each pickup item to be loaded in a stack.

Inequalities (3.7) are responsible for enforcing the LIFO policy in the route. To show how the LIFO policy can be violated in a route, consider two different pickup locations, $i \in P$ and $j \in P$. Let the notation $u \prec v$ mean that u is visited before v in the route. We say that items i and j cross each other if they are visited according to the pattern $i \prec j \prec n+i \prec n+j$. If such visitation pattern occurs, items i and j cannot be in the same stack, because it would violate the LIFO policy since item j is loaded after item i, but item i is delivered first.

Let $S \subset P \cup D$ be a subset such that $i, j \in P$, $j \in S$, and $i, n+i, n+j \notin S$. Also, let $k \in M$ be a stack of the vehicle. If the vehicle visits i and loads its item in stack k, traverses set S without leaving it, also loading item j in stack k, and immediately after visits $n+i$, then the LIFO policy is violated, since item j is loaded after item i in stack k, but $n+i$ is visited before $n+j$. Observe that pickup i has to be visited before entering S, otherwise the precedence between i and $n+i$ would be violated.

The capacity constraint of each stack in the vehicle is imposed through inequalities (3.8). These fractional capacity inequalities are similar to the ones found in the capacitated vehicle routing problem (CVRP). The difference is that, in the CVRP, the right-hand side of the inequalities is an integer. Given a subset $S \subset P \cup D$ of vertices, and a stack $k \in M$, these constraints, together with the integrality of the variables, impose that at least $\sum_{u \in S} d_u y^k_{\gamma(u)}/Q$ vehicle visits are needed to satisfy the demand in S using only stack k. We use function γ because the y variables are not defined for delivery locations. Therefore, when $u \in S$ and $u \in D$, we use the function to obtain the variable corresponding to its pickup.

3.4 Valid inequalities

In this section, we present valid inequalities for the PDTSPMS. Initially, we describe inequalities previously proposed in the literature, which we incorporate in our algorithm. Then, we introduce new valid inequalities that explore particular properties of the problem.

Given that every solution for the PDTSPMS is also feasible for the pickup and delivery traveling salesman problem (PDTSP), all valid inequalities for this problem can be used. We use the following additional notation: given a subset $S \subseteq P \cup D$, let $\pi(S) = \{i \in P : n+i \in S\}$ be the set of pickup vertices such that their corresponding delivery vertices belong to S, and $\sigma(S) = \{n+i \in D : i \in S\}$ be the set of delivery vertices such that their respective pickup vertices are in S. We consider two classes of inequalities introduced by Balas et al. [1995]:

$$x(S \setminus \pi(S), \overline{S} \setminus \pi(S)) \geq 1, \qquad S \subseteq P \cup D, \tag{3.11}$$
$$x(\overline{S} \setminus \sigma(S), S \setminus \sigma(S)) \geq 1, \qquad S \subseteq P \cup D. \tag{3.12}$$

Given a subset $S \subseteq P \cup D$, in a feasible route, inequalities (3.11) state that the last node visited in S and the node visited immediately after S cannot be a pickup with its corresponding delivery in S. Similarly, inequalities (3.12) assert that the last node visited before entering S and the first node visited in S cannot be a delivery with its corresponding pickup inside S. Inequalities (3.11) and (3.12) are referred to as predecessor and successor inequalities, respectively.

Côté et al. [2012a] adapted the classical rounded capacity inequalities of the CVRP to the PDTSPMS. Let $q(S) = \sum_{u \in S} d_u$ be the total sum of the item size of each vertex inside $S \subseteq P \cup D$. Using the improved lower bound (LB) of Ropke and Cordeau [2009], on the number of times that the vehicle must leave subset S to satisfy its demand, the rounded capacity inequalities are:

$$x(S, \overline{S}) \geq \left\lceil \max\left(1, \frac{q(\pi(S) \setminus S)}{|M|Q}, \frac{-q(\sigma(S) \setminus S)}{|M|Q}\right) \right\rceil, \qquad S \subset P \cup D. \tag{3.13}$$

Inequalities (3.13) consider the total capacity $|M|Q$ of the vehicle. Observe that $\pi(S) \setminus S$ is the set of items the vehicle delivers in S, and $\sigma(S) \setminus S$ is the set of items the vehicle delivers from S.

Inequalities (3.11), (3.12), and (3.13), are used by all the other algorithms from the literature for the PDTSPMS. The difference between the algorithms is in the way they are separated and managed. In the next sections, we present new valid inequalities for the PDTSPMS.

3.4.1 Strengthening the fractional capacity inequalities

The sum $\sum_{u\in S} d_u y^k_{\gamma(u)}$ in the right-hand side of inequalities (3.8) considers the item sizes of both pickups and deliveries in set S. Hence, the right-hand side of the inequalities is reduced due to the sum of negative sizes from deliveries. We can split the sum into two other sums, $\sum_{u\in\pi(S)\setminus S} d_u y^k_u$ and $\sum_{u\in\sigma(S)\setminus S} d_u y^k_{\gamma(u)}$, the total demand from $\overline{S}$ delivered in S using stack k and the total demand from $\overline{S}$ picked up in S using stack k, respectively. Note that $\sum_{u\in\sigma(S)\setminus S} d_u y^k_{\gamma(u)} < 0$, since we add the demands for deliveries whose corresponding pickups belong to S. Therefore, given a subset $S \subset P \cup D$, and a stack $k \in M$, we have that $\sum_{u\in S} d_u y^k_{\gamma(u)} \leq \max(1, -\sum_{u\in\sigma(S)\setminus S} d_u y^k_{\gamma(u)}, \sum_{u\in\pi(S)\setminus S} d_u y^k_u)$. As a result, we obtain the inequalities:

$$x(S,\overline{S}) \geq \max\left(1, \frac{-\sum_{u\in\sigma(S)\setminus S} d_u y^k_{\gamma(u)}}{Q}, \frac{\sum_{u\in\pi(S)\setminus S} d_u y^k_u}{Q}\right), S \subset P \cup D, k \in M. \quad (3.14)$$

Note that the resulting inequalities (3.14) are nonlinear due to the maximum function. In Section 3.5, we explain how we use such inequalities in our algorithm.

The number of times the vehicle leaves a subset of vertices is an integer. Therefore, the lower bound $\sum_{u\in\pi(S)\setminus S} d_u y^k_u/Q$, the number of vehicle visits needed to satisfy the demand from $\overline{S}$ delivered in S using only stack k, can be increased to $\lceil\sum_{u\in\pi(S)\setminus S} d_u y^k_u/Q\rceil$. This same reasoning can be used to the lower bound $-\sum_{u\in\sigma(S)\setminus S} d_u y^k_{\gamma(u)}/Q$, the number of vehicle visits needed to satisfy the demand from $\overline{S}$ picked up in S using stack k, to obtain $\lceil-\sum_{u\in\sigma(S)\setminus S} d_u y^k_{\gamma(u)}/Q\rceil$. Since there is a term containing variables inside the ceil function, if we substitute these lower bounds in the right-hand side of inequalities (3.8) directly, it results in nonlinear inequalities. To obtain linear inequalities, we use the following theorem from Baldacci et al. [2007].

Theorem 6. *Let a, b, and c be three nonnegative integer values with $a > c$ and $\mathrm{mod}(a,c) \neq 0$. Then,*

$$\left\lceil\frac{a-b}{c}\right\rceil \geq \left\lceil\frac{a}{c}\right\rceil - \frac{b}{\mathrm{mod}(a,c)}. \quad (3.15)$$

Although stated in Theorem 6, it is not necessary for b to be an integer. Given a subset $S \subset P \cup D$ of vertices, and a fixed stack $k \in M$, we have the identity $\sum_{i\in\pi(S)\setminus S} d_i = \sum_{i\in\pi(S)\setminus S} d_i y_i^k + \sum_{i\in\pi(S)\setminus S} \sum_{l\in M\setminus\{k\}} d_i y_i^l$. From equalities (3.6), for each $i \in P$, note that $\sum_{l\in M} y_i^l = y_i^k + \sum_{l\in M\setminus\{k\}} y_i^l = 1$. Therefore, multiplying the previous equation by d_i and summing for each vertex $i \in \pi(S)\setminus S$, we obtain the previous identity, which can be rewritten as $\sum_{i\in\pi(S)\setminus S} d_i y_i^k = \sum_{i\in\pi(S)\setminus S} d_i - \sum_{i\in\pi(S)\setminus S} \sum_{l\in M\setminus\{k\}} d_i y_i^l$. Using this identity and Theorem 6, we can derive the following valid inequalities:

$$x(S,\overline{S}) \geq \left\lceil \frac{\sum_{i\in\pi(S)\setminus S} d_i}{Q} \right\rceil - \frac{\sum_{i\in\pi(S)\setminus S} \sum_{l\in M\setminus\{k\}} d_i y_i^l}{\operatorname{mod}(\sum_{i\in\pi(S)\setminus S} d_i, Q)}, \qquad S \subset P \cup D, k \in M. \quad (3.16)$$

Similarly, using the same reasoning applied to obtain the previous identity, we deduce that $\sum_{i\in\sigma(S)\setminus S} d_i y_{\gamma(i)}^k = \sum_{i\in\sigma(S)\setminus S} d_i - \sum_{i\in\sigma(S)\setminus S} \sum_{l\in M\setminus\{k\}} d_i y_{\gamma(i)}^l$. Now, using Theorem 6 again, we obtain the following valid inequalities:

$$x(S,\overline{S}) \geq \left\lceil \frac{-\sum_{i\in\sigma(S)\setminus S} d_i}{Q} \right\rceil + \frac{\sum_{i\in\sigma(S)\setminus S} \sum_{l\in M\setminus\{k\}} d_i y_{\gamma(i)}^l}{\operatorname{mod}(-\sum_{i\in\sigma(S)\setminus S} d_i, Q)}, \quad S \subset P \cup D, k \in M. \quad (3.17)$$

Note that we multiplied the terms by -1 because $d_i < 0$, for $i \in \sigma(S)$. We can use the same rationale of inequalities (3.14) to employ the maximum function between the right-hand side of inequalities (3.16) and (3.17). As we explain in Section 3.5, we use this idea to add in the model the most violated inequality among the two.

3.4.2 Lifted LIFO inequalities

Initially, we show how to lift the constraints proposed by Cordeau et al. [2010] to model the LIFO policy in the PDTSPL. Côté et al. [2012a] used these constraints to derive LIFO inequalities for the PDTSPMS. Then, from our lifted version of the inequalities proposed by Cordeau et al. [2010], we lift the inequalities of Côté et al. [2012a].

Let Ω be the collection of all sets $S \subset P \cup D$ for which there is at least one pickup $j \in P$, such that $j \in S$ and $n+j \notin S$, or $j \notin S$ and $n+j \in S$. The LIFO policy in the PDTSPL is guaranteed by the following inequality:

$$x(i,S) + x(S) + x(S,n+i) \leq |S|, \qquad S \in \Omega, i \in P, i, n+i \notin S. \quad (3.18)$$

To model the LIFO policy when only one stack is available, it is necessary just one of the conditions used in the definition of Ω, i.e. $j \in S$ and $n+j \notin S$ or $n+j \in S$ and $j \notin S$. The constraint establishes that there cannot be a path that leaves i, visits subset S, and exits through $n+i$. In the case where $j \in S$ and $n+j \notin S$, the

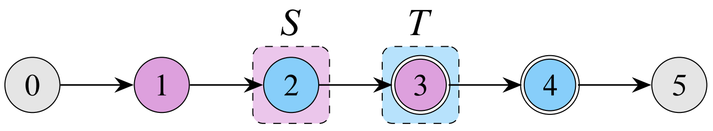
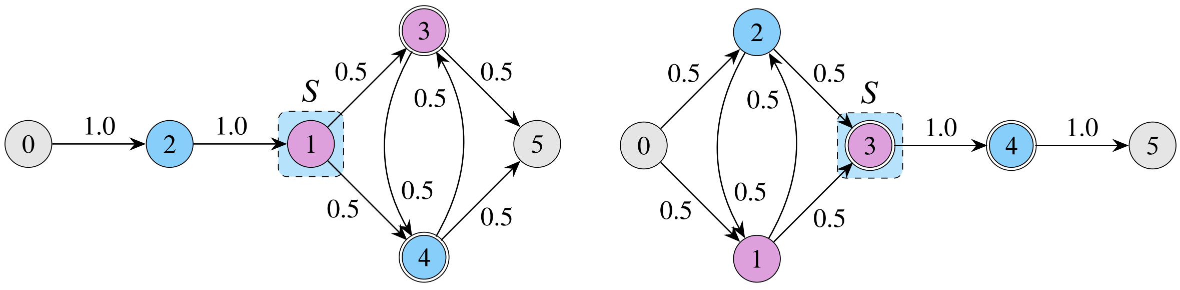


Figure 3.2: Solution that violates inequalities (3.18).



(a) Inequality (3.18) violated such that $j = 1 \in S$, $n + j = 3 \notin S$, $i = 2 \notin S$, and $n + i = 4 \notin S$.

(b) Inequality (3.18) violated such that $j = 1 \notin S$, $n + j = 3 \in S$, $i = 2 \notin S$, and $n + i = 4 \notin S$.

Figure 3.3: Two different fractional solutions that violate inequalities (3.18).

vehicle delivers i before j, despite j being visited after i. In the case where $j \notin S$ and $n + j \in S$, the vehicle delivers j before i, despite i being visited after j. The LIFO policy is violated in both situations. Figure 3.2 shows an example of this situation. We can use either set S or set T with their corresponding paths to define a violated inequality (3.18).

Although we can consider only one of the conditions to impose the LIFO policy in the PDTSPL. For fractional solutions, given an inequality violated for one of the conditions, there is not necessarily an inequality violated for the other. In Figure 3.3, we show an example of two fractional solutions that violate inequalities (3.18). In Figure 3.3a, there is a subset $S \subset P \cup D$ such that $j \in S$ and $n + j \notin S$, and the defined inequality (3.18) is violated. However, there is no $S \subset P \cup D$ such that $n + j \in S$ and $j \notin S$, and an inequality (3.18) is violated. In Figure 3.3b, we show the opposite situation. There is $S \subset P \cup D$ such that $n + j \in S$ and $j \notin S$, and the defined constraint (3.18) is violated. On the other hand, there is no $S \subset P \cup D$ such that $j \in S$ and $n + j \notin S$, in a manner that the defined constraint is violated.

After this discussion, we can lift inequalities (3.18) by treating each one of the cases separately, obtaining:

$$x(S) + x(S, n+i) \leq |S| - 1, \qquad S \subset P \cup D, i, j \in P, j \in S, i, n+i, n+j \notin S, \quad (3.19)$$

$$x(i, S) + x(S) \leq |S| - 1, \qquad S \subset P \cup D, i, j \in P, n+j \in S, i, n+i, j \notin S. \quad (3.20)$$

Although inequalities (3.18) are valid when the vehicle has only one stack, they are invalid for the PDTSPMS since items that cross each other can be loaded in different stacks. For example, if the vehicle has two stacks, two pickups can cross each other, since they can be loaded in different stacks. However, there cannot be a scenario where three items cross each other, since three stacks would be necessary and the vehicle has only two. Consequently, Côté et al. [2012a] extended inequalities (3.18) for the case where the vehicle has multiple stacks.

Consider the following visitation pattern in which all pickups cross each other: $i_1 \prec i_2 \prec \cdots \prec i_{|M|+1} \prec n+i_1 \prec n+i_2 \prec \cdots \prec n+i_{|M|+1}$. Let S_{i_h}, $2 \leq h \leq |M|+1$, be a set that contains vertices i_h to $i_{|M|+1}$ and $n+i_1$ to $n+i_{h-2}$ (we define the sequence from $n+i_1$ to $n+i_0$ as empty when $h=2$) but without vertices i_1 to i_{h-1} and $n+i_{h-1}$ to $n+i_{|M|+1}$. As all pickups cross each other, at least $|M|+1$ stacks are required. Otherwise, by the pigeonhole principle, two items that cross each other will be loaded in the same stack. Thus, to prohibit all paths that include this visitation pattern, Côté et al. [2012a] proposed the following inequalities:

$$\sum_{h=1}^{|M|} \left[x(i_h, S_{i_{h+1}}) + x(S_{i_{h+1}}) + x(S_{i_{h+1}}, n+i_h)\right] \leq \sum_{h=1}^{|M|} \left|S_{i_{h+1}}\right| + |M| - 1. \qquad (3.21)$$

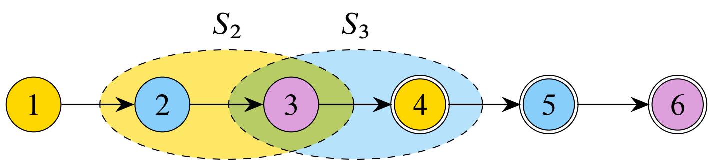
Inequalities (3.21) prohibit all paths from i_h to $n+i_h$ going through vertices in S_{h+1}, for $h = 1, \ldots, |M|$, from being made at the same time. We show an example of a visitation pattern for $|M| = 2$ in Figure 3.4a. Although the vehicle has only two stacks, the three pickups in the figure cross each other. In the bottom, in Figure 3.4c, excluding symmetrical alternatives, we show all possible options for loading three items in a vehicle with two stacks. Note that none of them satisfy all three requests simultaneously.

In order to lift inequalities (3.21), consider the previously discussed visitation pattern where $|M|+1$ requests cross each other and the sets S_{i_h}, $2 \leq h \leq |M|+1$. Consider the following inequality, clearly valid for the PDTSPMS, that adds 1 unit to the right-hand side of inequalities (3.19):

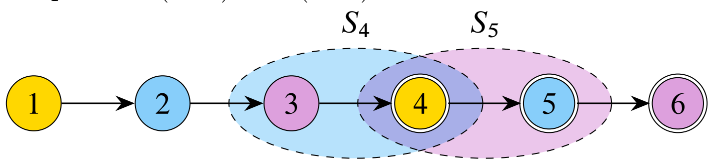
$$x(S) + x(S, n+i) \leq |S|, \qquad S \subset P \cup D, i, j \in P, j \in S, i, n+i, n+j \notin S. \quad (3.22)$$

Summing inequalities (3.22) defined for each i_h and $S_{i_{h+1}}$, $1 \leq h \leq |M|$, we obtain:

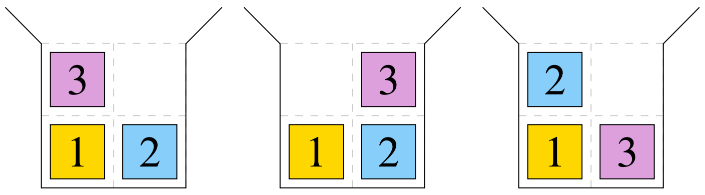
$$\sum_{h=1}^{|M|} \left[x\left(S_{i_{h+1}}\right) + x\left(S_{i_{h+1}}, n+i_h\right)\right] \leq \sum_{h=1}^{|M|} \left|S_{i_{h+1}}\right|. \qquad (3.23)$$



(a) Infeasible visitation pattern which violates inequalities (3.21) and (3.24).



(b) Infeasible visitation pattern which violates inequalities (3.25).



(c) All possible loading options for three items crossing each other in a vehicle with two stacks.

Figure 3.4: Infeasible pattern with all possible loadings for a vehicle with two stacks.

The only possibility of equality in inequalities (3.23) is if the vehicle executes the visitation pattern where all requests cross each other. This means that all inequalities (3.22) defined for each i_h and $S_{i_{h+1}}$, $1 \leq h \leq |M|$, are also at equality. Since the vehicle has $|M|$ stacks and $|M|+1$ items cross each other, we have an infeasible visitation pattern. Therefore, inequalities (3.24) follow:

$$\sum_{h=1}^{|M|} \left[x(S_{i_{h+1}}) + x(S_{i_{h+1}}, n + i_h)\right] \leq \sum_{h=1}^{|M|} |S_{i_{h+1}}| - 1. \tag{3.24}$$

Comparing inequalities (3.24) with inequalities (3.21), note that we reduce a fractional amount on the left-hand side, while we reduce an integral amount on the right-hand side.

Let S_{n+i_h}, $1 \leq h \leq |M|$, be a set that contains vertices i_{h+2} to $i_{|M|+1}$ and $n + i_1$ to $n + i_h$ (we define the sequence from i_{h+2} to $i_{|M|+1}$ as empty when $h + 2 > |M| + 1$) but without vertices i_1 to i_{h+1} and $n + i_{h+1}$ to $n + i_{|M|+1}$. Proceeding for inequalities (3.20) equivalently to what we did for inequalities (3.19), we obtain the following valid inequalities:

$$\sum_{h=1}^{|M|} \left[x(i_{h+1}, S_{n+i_h}) + x(S_{n+i_h})\right] \leq \sum_{h=1}^{|M|} |S_{n+i_h}| - 1 \tag{3.25}$$

An example of a prohibited pattern for inequalities (3.25) together with the defined sets is shown in Figure 3.4b. Observe in Figure 3.4 that for both inequalities (3.24) and (3.25) the defined sets S_{i_h} and S_{n+i_h}, respectively, move along the path determined by the visitation pattern as h increases.

3.4.3 Lifted conflict capacity inequalities

We also lift and use in our branch-and-cut algorithm the conflict capacity inequalities proposed by Côté et al. [2012a]:

$$x(i,S) + x(S) + x(S,n+i) \leq |S|\,, \qquad \begin{array}{l} S \subset P \cup D, i \in P, i, n+i \notin S, \\ z(S) > Q(|M|-1), \end{array} \tag{3.26}$$

where $z(S) = \max(q(\pi(S)\setminus S), -q(\sigma(S)\setminus S))$. The total demand $q(\pi(S)\setminus S)$ is the sum of the sizes of items picked up before i and delivered before $n+i$, and $-q(\sigma(S)\setminus S)$ is the sum of the sizes of items picked up between i and $n+i$ and delivered after $n+i$. The inequalities forbid items in S that cross with i from being loaded in the same stack as i. Therefore, they must fit in the remaining capacity available $Q(|M|-1)$. By treating each case separately as we did for inequalities (3.18), we can lift inequalities (3.26).

Given a subset $S \subset P \cup D$ of vertices, and a pickup vertex $i \in P$, suppose that $q(\pi(S)\setminus S) > Q(|M|-1)$, its not necessary to visit $n+i$ immediately after S as long as i is visited immediately before S. All crossings are preserved since $n+i$ has to be visited after S to respect the precedence. All the items in $\pi(S)\setminus S$ are still loaded in the vehicle before visiting i and entering S. Now, assume that $-q(\sigma(S)\setminus S) > Q(|M|-1)$, its not necessary to visit i immediately before entering S as long as $n+i$ is visited immediately after S. All crossings are preserved since i has to be visited before entering S to respect the precedence. Hence, all pickups whose deliveries are in $\sigma(S)\setminus S$ will be loaded in the vehicle along with item i. Therefore, we obtain the new inequalities:

$$x(i,S) + x(S) \leq |S| - 1, \qquad \begin{array}{l} S \subset P \cup D, i \in P, i, n+i \notin S, \\ q(\pi(S)\setminus S) > Q(|M|-1), \end{array} \tag{3.27}$$

$$x(S) + x(S,n+i) \leq |S| - 1, \qquad \begin{array}{l} S \subset P \cup D, i \in P, i, n+i \notin S, \\ -q(\sigma(S)\setminus S) > Q(|M|-1). \end{array} \tag{3.28}$$

Côté et al. [2012a] extended inequalities (3.26) to the case where several items cross each other in a path. We first present such inequalities, then we show how to lift them. Consider the following visitation pattern performed by the vehicle $i_1 \prec i_2 \prec$

$\cdots \prec i_k \prec n+i_1 \prec n+i_2 \prec \cdots \prec n+i_k$, $1 \leq k \leq |M|-1$, such that $|M| \geq 2$. Let S_{i_h}, $2 \leq h \leq k+1$, be a set that contains vertices i_h to i_k and $n+i_1$ to $n+i_{h-2}$ (the sequence from i_h to i_k and from $n+i_1$ to $n+i_0$ is empty when $h=k+1$ and $h=2$, respectively) but without vertices i_1 to i_{h-1} and $n+i_{h-1}$ to $n+i_k$. Due to the visitation pattern, these k items must be loaded into k different stacks inside the vehicle. Thus, all the other items in $S_{i_{h+1}}$ that cross with items i_h, $1 \leq h \leq k$, must be loaded in the remaining $|M|-k$ stacks. Note that these items does not need to cross with each other.

We have to consider the items that were already loaded in the vehicle before entering S_{i_2}, and the items which will be in the vehicle after visiting $S_{i_{k+1}}$. In the first case, we have items loaded before i_1 and delivered between i_k and $n+i_1$. The deliveries associated with these items are in $S_{i_2} \cap \cdots \cap S_{i_{k+1}}$, as we want only those that are loaded before i_1, their corresponding pickups are not in $S_{i_2} \cup \cdots \cup S_{i_{k+1}}$. In the second case, the items are loaded between i_k and $n+i_1$ and delivered after $n+i_k$. The pickups for these items are in $S_{i_2} \cap \cdots \cap S_{i_{k+1}}$, given that we want only those that are delivered after $n+i_k$, their corresponding deliveries are not in $S_{i_2} \cup \cdots \cup S_{i_{k+1}}$. Accordingly, let

$$\begin{aligned} z(S_{i_2}, \ldots, S_{i_{k+1}}) = \max(&q(\pi(S_{i_2} \cap \cdots \cap S_{i_{k+1}}) \setminus (S_{i_2} \cup \cdots \cup S_{i_{k+1}})), \\ &-q(\sigma(S_{i_2} \cap \cdots \cap S_{i_{k+1}}) \setminus (S_{i_2} \cup \cdots \cup S_{i_{k+1}}))) \end{aligned} \tag{3.29}$$

be the maximum between the total demand for items delivered and picked up in $S_{i_2}, \ldots, S_{i_{k+1}}$. If $z(S_{i_2}, \ldots, S_{i_{k+1}}) > Q(|M|-k)$, it is impossible to fit these items in the remaining stacks. Consequently, we can eliminate such infeasible solution with the following inequalities:

$$\sum_{h=1}^{k} \left[x(i_h, S_{i_{h+1}}) + x(S_{i_{h+1}}) + x(S_{i_{h+1}}, n+i_h)\right] \leq \sum_{h=1}^{k} \left|S_{i_{h+1}}\right| + k - 1. \tag{3.30}$$

In the case where $-q(\sigma(S_{i_2} \cap \cdots \cap S_{i_{k+1}}) \setminus (S_{i_2} \cup \cdots \cup S_{i_{k+1}})) > Q(|M|-k)$, using inequalities (3.28), we can lift inequalities (3.30) to:

$$\sum_{h=1}^{k} \left[x(S_{i_{h+1}}) + x(S_{i_{h+1}}, n+i_h)\right] \leq \sum_{h=1}^{k} \left|S_{i_{h+1}}\right| - 1. \tag{3.31}$$

Similarly, if $q(\pi(S_{i_2} \cap \cdots \cap S_{i_{k+1}}) \setminus (S_{i_2} \cup \cdots \cup S_{i_{k+1}})) > Q(|M|-k)$, we can also obtain a lifted version using inequalities (3.27):

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

In Section 3.5.8, we explain how to check if it is feasible to use inequalities (3.31) and (3.32) when $z(S_{i_2}, \ldots, S_{i_{k+1}}) \leq Q(|M| - k)$. This situation only occurs in instances where items have non-unitary sizes.

3.4.4 Generalized LIFO inequalities with loading

Consider the following visitation pattern where k requests cross each other $i_1 \prec i_2 \prec \cdots \prec i_k \prec n + i_1 \prec n + i_2 \prec \cdots \prec n + i_k$, $2 \leq k \leq |M|$. Let S_{i_h}, $2 \leq h \leq k$, be a set that contains vertices i_h to i_k and $n + i_1$ to $n + i_{h-2}$ (the sequence from $n + i_1$ to $n + i_0$ is empty when $h = 2$) but without vertices i_1 to i_{h-1} and $n + i_{h-1}$ to $n + i_k$. If the vehicle has $|M| \geq 2$ stacks and $k \leq |M|$ requests cross each other, then all items mutually crossing must be loaded in different stacks in the vehicle. This fact leads to inequalities:

$$\sum_{h=1}^{k} y^l_{i_h} + \sum_{h=1}^{k-1} \left[x(S_{i_{h+1}}) + x(S_{i_{h+1}}, n + i_h)\right] \leq \sum_{h=1}^{k-1} |S_{i_{h+1}}| + 1, \quad l \in M. \tag{3.33}$$

Given a stack $l \in M$, if the vehicle performs the previous visitation pattern, then two items crossing each other cannot be loaded in stack l. Inequalities (3.33) generalize constraints (3.7) for $k \geq 3$ requests crossing each other. For the case where $|M|+1$ requests cross each other, we have the stronger inequalities (3.24), which do not consider the loading variables since the number of items crossing each other is greater than the number of available stacks.

3.5 Branch-and-cut algorithm

We now present the proposed branch-and-cut algorithm based on formulation (3.1)-(3.10) which incorporates the valid inequalities of Section 3.4. We use the branch-and-cut framework available through the Concert library in IBM ILOG CPLEX 12.10 to implement the algorithm. We provide an upper bound (UB) obtained by the Large Neighborhood Search heuristic of Côté et al. [2012b] at the beginning of the algorithm. We implemented the algorithm of Dinic to solve our max-flow problems [Dinic, 1970].

3.5.1 Arc elimination

Although the problem is defined in a complete graph, some arcs never appear in a feasible solution. Therefore, we can exclude such arcs from the formulation by not defining variables for them. The initial depot 0 cannot be the predecessor of any delivery, so we remove all arcs $(0, n+i) \in A$, such that $n+i \in D$. Also, the final depot $2n+1$ cannot be the successor of any pickup, so we eliminate all arcs $(i, 2n+1) \in A$, such that $i \in P$. The successor of a delivery cannot be its corresponding pickup, since the precedence is violated. Therefore, we remove all arcs $(n+i, i)$, such that $i \in P$. We also delete arcs $(i, 0)$ and $(2n+1, i)$ such that $i \in V \setminus \{0\}$ and $i \in V \setminus \{2n+1\}$, respectively.

3.5.2 Symmetry

The y variables introduce a degree of symmetry in formulation (3.1)-(3.10). Given a feasible solution for the formulation, it is possible to obtain an equivalent set of solutions exchanging items from stacks. For instance, if an item is loaded in a stack $k \in M$, just change the stack to $k+1 \bmod |M|$.

The first picked up item can be loaded in any stack inside the vehicle. Hence, to reduce the degree of symmetry in our model, we add constraints (3.34). The constraints state that the first pickup is performed using the first stack.

$$x_{0j} \leq y_j^1, \qquad j \in P. \tag{3.34}$$

Depending on the choices of which fractional variables to branch, we can create unbalanced branch-and-bound trees. Thus, it is beneficial to define branching priorities to avoid this issue [Leitner et al., 2017]. As discussed earlier, the loading variables introduce a degree of symmetry in the formulation. Consequently, it is interesting to first define a route and then determine the loading for it. Otherwise, we can obtain the same route on several nodes in the branch-and-bound tree. Therefore, in our branch-and-cut algorithm, we define the highest branching priority for x variables.

3.5.3 Initial model

The initial model for formulation (3.1)-(3.10) consists of the objective function (3.1), degree constraints (3.2) and (3.3), equalities (3.6) requiring each item to be loaded in a single stack, and constraints (3.34) to reduce symmetry. The integrality constraints (3.9) and (3.10) of the variables are removed during the enumeration of the branch-and-bound tree. We relax inequalities (3.4), (3.5), (3.7), and (3.8), in the branch-and-cut algorithm, and add them dynamically throughout the branch-and-bound tree using our separation procedures.

3.5.4 Cut pool

Inequalities (3.11), (3.12), and (3.13), are separated heuristically at the root node of the branch-and-bound tree. Therefore, the bound provided by the linear relaxation can vary according to how they are separated and managed. In our branch-and-cut algorithm, we add the inequalities for which the set defining it does not contain a set already found for another violated inequality. For each class of inequality, if the current set is contained in a set previously found, then we replace the previous set with the current one. Therefore, we keep all sets of violated inequalities found but not used in a cut pool. When the separation step performed at the root node of the branch-and-bound tree finds no further violated cut, we check if there is still any violated inequality in the cut pool. If it exists, we add the violated inequalities to the model and continue to solve the linear relaxation until no more violated inequality is found.

3.5.5 Variable fixing

If we add either the constraints $x_{ij} = 1$ or $x_{ij} = 0$ for an arc $(i, j) \in A$ in the model and the dual bound obtained is greater than an UB, then we can fix $x_{ij} = 0$ or $x_{ij} = 1$, respectively. We can use the reduced cost of an arc variable to decide if we can fix it to zero. However, for the case where we fix an arc variable to one, we must solve the linear relaxation obtained. Despite the fast solution of the linear relaxation, the number of arcs is enormous. Therefore, we perform this variable fixing only for the arcs that leave the initial depot and those that go to the final depot. We execute our variable fixing after the separation phase of the root node of the branch-and-bound tree has ended.

3.5.6 Separation procedures

Given a fractional solution $(\overline{x}, \overline{y})$ for formulation (3.1)-(3.10), we construct the associated support graph $\overline{G} = (V, \overline{A})$, with the same vertex set V and arcs $\overline{A} = A$, where each arc $(i, j) \in \overline{A}$ has capacity equal to $\overline{x}_{ij}$. Using the associated support graph $\overline{G}$, we use the separation procedures described by Pereira and Urrutia [2018] for inequalities (3.4), (3.5), (3.11), and (3.12). For inequalities (3.11) and (3.12), since they are separated heuristically, we also use the separation procedures proposed by Balas et al. [1995].

When a violated inequality (3.4) or (3.5) is found for a vertex set S, we can check whether $\max(\lceil q(\pi(S) \setminus S)/|M|Q \rceil, \lceil -q(\sigma(S) \setminus S)/|M|Q \rceil)$ is greater than the lower bound already provided by the right-hand of the correspondent inequality. We define the inequality with the maximum lower bound calculated.

3.5.6.1 Fractional capacity constraints

To show how to separate inequalities (3.8) exactly, we first rewrite them. Given a subset $S \subset P \cup D$, and a stack $k \in M$, the inequalities are equivalent to:

$$Qx(S, \overline{S}) + \sum_{i \in S \cap D} -d_i y^k_{\lambda(i)} + \sum_{i \in S \cap P} d_i(1 - y^k_i) + \sum_{i \in P \cap \overline{S}} d_i \geq \sum_{i \in P} d_i. \qquad (3.35)$$

Let $f(S) = Qx(S, \overline{S}) + \sum_{i \in S \cap D} -d_i y^k_{\lambda(i)} + \sum_{i \in S \cap P} d_i(1 - y^k_i) + \sum_{i \in P \cap \overline{S}} d_i$. The separation problem is equal to compute $\min\{f(S) : S \subset P \cup D \wedge S \neq \emptyset\}$. The fractional capacity constraint is violated if and only if $f(S) < \sum_{i \in P} d_i$. On the other hand, if $f(S) \geq \sum_{i \in P} d_i$, then no fractional capacity constraint is violated. It is possible to find the subset $S \subset P \cup D$ and stack $k \in M$ for which $f(S)$ is minimum using a max-flow algorithm.

From the definition of inequalities (3.8), the subset $S \subset P \cup D$ must necessarily contain at least one pickup to violate the inequality for a given stack $k \in M$. Otherwise, if it contains only deliveries, the sum $\sum_{i \in S} d_i y^k_{\lambda(i)}$ is negative and the constraint is trivially satisfied. For each pickup $h \in P$, and each stack $k \in M$, we proceed in the following way:

1. Create the associated support graph $\overline{G}$.
2. Multiply the capacity $\overline{x}_{ij}$ of each arc $(i, j) \in \overline{A}$ by Q.
3. Increase the capacity of each arc $(h, j) \in \overline{A}$, $j \in P$, by the value d_j.
4. Increase the capacity of each arc $(n + i, 2n + 1) \in \overline{A}$, $n + i \in D$, by the value $d_{\lambda(n+i)} \overline{y}^k_{\lambda(n+i)}$.
5. For each $i \in P$, create the arc $(i, 2n + 1)$ with capacity $d_i(1 - \overline{y}^k_i)$.
6. Run the max-flow from h to $2n + 1$. If the flow value is smaller than $\sum_{i \in P} d_i$, then by the max-flow min-cut theorem, we can obtain a set $S \subset P \cup D$ such that the inequality (3.8) defined is violated.

Given a subset $S \subset P \cup D$ and a stack $k \in M$, which define a violated inequality (3.8) for a fractional solution, we do not add the corresponding inequality (3.8) in the model. Instead, we check which of the inequalities (3.16) and (3.17) provide the maximum violation for the particular solution, and add it to the model. Thus, the previous procedure works as a heuristic method for separating inequalities (3.16) and (3.17).

If the solution in the current node of the branch-and-bound tree is integral, we can separate inequalities (3.8) by simply visiting the route and keeping the residual capacity of each stack in the vehicle. If the capacity of a stack is exceeded at any pickup location, the subset of nodes determined by the path between the moment the stack was first used and this pickup defines a violated inequality (3.8). Similarly to fractional solutions, in this case, instead of adding the corresponding violated inequality (3.8), we compute the maximum of the respective inequality (3.14) and add it to the model.

3.5.6.2 LIFO constraints

To show how to separate inequalities (3.7), note that they are equivalent to:

$$\sum_{l \in M\setminus\{k\}} y_i^l + y_j^l + x(S,\overline{S}) + x(\overline{S}, n+i) \geq 2, \quad \begin{array}{l} S \subset P \cup D, i,j \in P, j \in S, \\ i, n+i, n+j \notin S, k \in M. \end{array} \tag{3.36}$$

Define $f(S) = x(S,\overline{S}) + x(\overline{S}, n{+}i)$. Note that the term $\sum_{l \in M\setminus\{k\}} y_i^l + y_j^l$ does not depend from S. The separation problem of inequalities (3.36) is equivalent to compute $\min\{f(S) : S \subset P \cup D \wedge j \in S \wedge i \notin S \wedge n+i \notin S \wedge n+j \notin S\}$. If $f(S) < 2 - \sum_{l \in M\setminus\{k\}} y_i^l + y_j^l$, then the LIFO inequality (3.7) defined is violated. Otherwise, if $f(S) \geq 2 - \sum_{l \in M\setminus\{k\}} y_i^l + y_j^l$, then no LIFO inequality is violated. For a given pair of pickup vertices, we compute the minimum of $f(S)$ using a max-flow algorithm.

For each pair of different pickup vertices, $i \in P$ and $j \in P$, constraints (3.36) are separated exactly in the following way:

1. Create the associated support graph $\overline{G} = (V, \overline{A})$.
2. Increase the capacity of each arc $(j,u) \in \overline{A}$, $u \in V \setminus \{0, j, n+i, 2n+1\}$, by the value $\overline{x}_{u,n+i}$.
3. Set the capacity of arcs $(i, n+i)$, $(n+j, n+i)$, and $(2n+1, n+i)$ equal to 2.
4. Execute the max-flow from source j to sink $n+i$. If the flow value is smaller than $2 - \sum_{l \in M\setminus\{k\}} y_i^l + y_j^l$, then by the max-flow min-cut theorem, we can obtain

a set S such that $j \in S$, $i \notin S$, $n+j \notin S$, $n+i \notin S$, and S violates the inequality (3.7) defined.

If the solution of the current node of the branch-and-bound tree is integral, we can separate inequalities (3.7) with a more efficient algorithm. Given a route that satisfies connectivity and precedence, visit each location using the loading assignment given by the solution. If at any moment we try to deliver an item that is not at the top of its stack, the LIFO constraint has been violated.

3.5.6.3 Lifted LIFO inequalities

We separate inequalities (3.24) using an enumerative procedure. For each pair of pickups $i \in P$ and $j \in P$, we check if there is a set $S \subset P \cup D$ such that $j \in S$, $n+j \notin S$, $i \notin S$, $n+i \notin S$, and $x(S) + x(S, n+i) > |S| - 1$. Note that this is the definition of a violated inequality (3.19). To compute this set, we adapted the procedure of Cordeau et al. [2010] proposed for inequalities (3.18). If such a set exists, we store the tuple (i, j, S). Once all tuples have been computed for each pair i and j, we check whether there are $|M|+1$ pickups that cross each other. All ordered sequences of size $|M|+1$ defined by pickups are enumerated to check if they define a violated inequality (3.24). This checking can be done in $\mathcal{O}(n^{|M|+1})$.

The exact separation procedure for inequalities (3.25) works in a similar fashion. The difference is that we separate $x(i, S) + x(S) > |S| - 1$, $S \subset P \cup D$, for each pair pickups $i \in P$ and $j \in P$ such that $n+j \in S$, $j \notin S$, $i \notin S$, and $n+i \notin S$. Again, to compute this set, we adapted the procedure of Cordeau et al. [2010]. After, all ordered sequences of size $|M|+1$ are checked to see if they define a violated inequality (3.25). We employ inequalities (3.25) only for fractional solutions. We do not consider them for integer solutions because in this case inequalities (3.24) are equivalent. Hence, inequalities (3.25) would be redundant.

3.5.6.4 Lifted conflict capacity inequalities

The separation of conflict capacity inequalities (3.31) and (3.32) is similar to the procedures described in Section 3.5.6.3. The difference is that we have to check ordered sequences of size k and determine the intersection of the sets to compute $z(S_{i_2}, \ldots, S_{i_{k+1}})$. We use $k \in \{1, 2, \ldots, |M| - 1\}$.

For each pair of vertices $i \in P$ and $j \in P$, we compute the tuple (i, j, S) such that $j \in S$, $n+j, i, n+i \notin S$, and $x(S) + x(S, n+i) > |S|$. Next, we check all ordered pickup sequences of size k to determine if they mutually cross. If all k pickups in the ordered sequence cross each other, we compute $-q(\sigma(S_{i_2} \cap \cdots \cap S_{i_{k+1}}) \setminus (S_{i_2} \cup \cdots \cup S_{i_{k+1}}))$ and check if its greater than $Q(|M| - k)$. In this case, a violated inequality (3.31) has been found.

We separate inequalities (3.32) with an equivalent procedure. But in this case, we have to compute $q(\pi(S_{i_2} \cap \cdots \cap S_{i_{k+1}}) \setminus (S_{i_2} \cup \cdots \cup S_{i_{k+1}}))$, if its greater than $Q(|M|-k)$ and all the k pickups in the ordered sequence cross each other, then the inequality (3.32) defined is violated.

For inequalities (3.27) and (3.28) ($k = 1$), we check if each tuple defines a violated inequality. We also check the tuples computed in the separation procedure of inequalities (3.25) described in Section 3.5.6.3.

3.5.6.5 Generalized LIFO inequalities with loading

We separate inequalities (3.33) adapting the separation procedure for inequalities (3.24) described in Section 3.5.6.3. After computing the tuples (i, j, S) for each pair of pickups i and j, such that $j \in S$, $n + j, i, n + i \notin S$, and $x(S) + x(S, n + i) > |S| - 1$, we check all ordered pickup sequences of size k to determine if they cross each other. Then, we compute the sum $\sum_{h=1}^{k} y_{i_h}^{l}$ to check if we can define a violated inequality (3.33). In our algorithm, we use the procedure for $k \in \{3, 4, \ldots, |M|\}$.

3.5.6.6 Rounded capacity inequalities

We can obtain the fractional version of rounded capacity inequalities (3.13) disregarding the ceiling function and using the weaker lower bound, $q(S)/|M|Q$:

$$x(S, \overline{S}) \geq \frac{q(S)}{|M|Q}, \qquad S \subset P \cup D. \tag{3.37}$$

We can separate inequalities (3.37) in a similar way as fractional capacity inequalities (3.8). To this end, we rewrite inequalities (3.37) in an equivalent form as:

$$|M|Qx(S, \overline{S}) - q(S \cap D) + q(\overline{S} \cap P) \geq q(P), \qquad S \subset P \cup D. \tag{3.38}$$

Note that $-q(S \cap D) > 0$, since $d_{n+i} < 0$, for $n + i \in D$. We separate inequalities (3.38) exactly and use as a heuristic procedure for inequalities (3.13). Inequalities (3.38) are separated exactly for each $h \in P$ in the following way:

1. Create the associated support graph $\overline{G}$.
2. Multiply the capacity $\overline{x}_{ij}$ of each arc $(i, j) \in \overline{A}$ by $|M|Q$.

3. Increase the capacity of each arc $(h, j) \in \overline{A}$, $j \in P$, by the value d_j.
4. Increase the capacity of each arc $(n+i, 2n+1) \in \overline{A}$, $n+i \in D$, by the value $-d_{n+i}$.
5. Run the max-flow from h to $2n+i$. If the flow value is smaller than $q(P)$, then by the max-flow min-cut theorem, we can obtain a set $S \subset P \cup D$ such that the inequality (3.38) defined is violated.

Since inequalities (3.13) are stronger than inequalities (3.37), given a subset $S \subset P \cup D$ which violates an inequality (3.37), S also defines a violated inequality (3.13). Therefore, we add the corresponding inequality (3.13) in the model.

While the lower bound in the right-hand side of inequalities (3.38) consider the total load picked up in a set $S \subset P \cup D$, we can also consider the load delivered in S using the lower bound $-q(S)/|M|Q$. This leads to the following inequalities:

$$|M|Qx(S, \overline{S}) + q(S \cap P) - q(\overline{S} \cap D) \geq -q(D), \qquad S \subset P \cup D. \tag{3.39}$$

The separation procedure for inequalities (3.39) is performed similarly to the one for inequalities (3.38). We also use the approach of Ropke and Cordeau [2009] to separate inequalities (3.13). From a randomly chosen initial vertex, the algorithm maximizes function (3.40) below. At each iteration, the node that maximizes function (3.40) is added to the set. The heuristic stops when a violated inequality (3.13) is found, or when the S set remains the same between two iterations. We execute the heuristic five times.

$$\begin{aligned} f(S) = {} & \lambda_1 \left(\max\{q(\pi(S) \setminus S), -q(\sigma(S) \setminus S)\} - Qx(S, \overline{S})\right) \\ & + \lambda_2 Q \left(\max\left\{\left\lceil \frac{q(\pi(S) \setminus S)}{Q} \right\rceil, \left\lceil \frac{-q(\sigma(S) \setminus S)}{Q} \right\rceil\right\} - x(S, \overline{S})\right). \\ & + \lambda_3 \left(\min\{q(\pi(S) \setminus S), -q(\sigma(S) \setminus S)\} - Qx(S, \overline{S})\right) \end{aligned} \tag{3.40}$$

The function $f(S)$ contains three parameters λ_1, λ_2, and λ_3. At the beginning of each execution, λ_1 and λ_2 receive random values in the range $[1, 5]$, and λ_3 receives a random value in the range $[0.1]$. We refer the reader to the work of Ropke and Cordeau [2009] for more details.

3.5.7 Checking feasibility

Given an integer solution that satisfies constraints (3.4) and (3.5), we verify whether the solution is feasible for the other constraints heuristically. We check if there is a stack assignment such that the LIFO and capacity constraints are respected. The motivation for the procedure is that, if a valid stack assignment is found, there is no need to separate LIFO or capacity inequalities. Besides, if the values of variables y in the current node are fractional, we avoid the need to branch. Although the problem is NP-hard, Côté et al. [2012a] solve this feasibility problem exactly in their algorithm. However, to mitigate the computational cost, we approach the problem heuristically.

The procedure works in a greedy fashion. Initially, we compute all crossings of the route. Then, we visit the route loading an item in the first available stack. For availability, given a stack, the item to be loaded in this stack cannot cross with any other item and its size cannot cause the capacity to be exceeded. If there is no stack available for the item, we check if moving an item from the top of a stack to the next one allows us to load it. If at any moment there is no stack available for the item, we abort the procedure.

3.5.8 Lifting capacity inequalities

The problem of determining whether a set I of item sizes can be packed into B stacks of size W in the PDTSPMS can be reduced to a bin packing problem. The solution of this problem allows to lift several capacity inequalities for the PDTSPMS. The motivation of this procedure is because inequalities (3.13), (3.27), (3.28), (3.31), and (3.32), estimate if the load can be packed in the vehicle by assuming that non-unitary sized items can be divided among the stacks. Therefore, the procedure applies only to instances that contain non-unitary sized items.

The lower bound on the right-hand side of inequalities (3.13) considers that non-unitary sized items can be divided among the stacks. Figure 3.5 shows an example. The figure shows a vehicle with $|M| = 2$ stacks of capacity $Q = 2$, and 3 items of size 2. Despite the total load of 6 being equal to the total capacity of the vehicle $|M|Q$, one of the items is divided between the stacks, which is forbidden in the PDTSPMS. The items in the vehicle would provide a lower bound of 1 for inequalities (3.13). However, for the input $I = \{2, 2, 2\}$, $B = 2$, and $W = 2$, the answer of the respective bin packing problem would be no. Consequently, the lower bound of the inequalities could be lifted by one.

Examining Figure 3.5 for inequalities (3.27), (3.28), (3.31) and (3.32), we would assume that the load fit into the two available stacks and the respective path would be considered feasible. However, if the corresponding bin packing problem was solved, such a path would be eliminated since it is impossible to load the items in two stacks without dividing one of them.

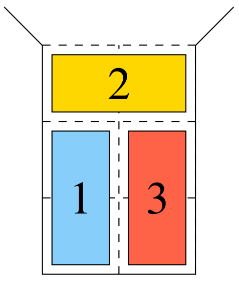


Figure 3.5: Items cannot be divided among stacks in the PDTSPMS.

To solve the associated bin packing problem we execute a heuristic with the aim of fixing items in or out of bins first. The heuristic fixes items in bins and also prohibits items from being placed in certain bins. If the sum of the sizes of two items is greater than W, then we can fix these items in different bins (valid only for the first pair of incompatible items). Then, for a new item considered, we check if it is in conflict (being placed in the same bin would violate the capacity) with all the other items already fixed. In a positive case, it can also be fixed in a different bin. If it is possible to put the item in an already open bin, we cannot fix such an item. Even if we cannot fix an item to a new bin, we can still prohibit it from being placed into bins where the capacity would be exceeded.

At the end of the previous heuristic, we check if the number of fixed items is greater than the number of available bins. If so, the problem is infeasible. If the number of fixed items is less than the number of available bins, we solve the remaining problem with an exact backtracking algorithm trying to assign the remaining items to the available bins. If at any time we find a feasible solution, we stop the recursion. Since the bin packing is a NP-hard problem, the execution time may become impracticable depending on the size of the instance. Therefore, although it was not necessary for our algorithm, after the fixing procedure, the size of the remaining problem can be checked to decide whether to solve it or not.

Whenever possible violated inequalities (3.13), (3.27), (3.28), (3.31), and (3.32), are found, we invoke the procedure to check if the corresponding inequality can be lifted. For inequalities (3.13), the right-hand side is increased by one, and for inequalities (3.27), (3.28), (3.31) and (3.32), we add the corresponding inequality since the visitation pattern induced by the sets involved in the crossings is infeasible. We also invoke the bin packing procedure for inequalities (3.4) and (3.5).

3.6 Computational results

In this section, we present the results from the computational experiments performed to evaluate our branch-and-cut algorithm. We compare our implemented algorithm with all the other algorithms in the literature for the PDTSPMS. Namely, the algorithms of Côté et al. [2012a], Sampaio and Urrutia [2017], and the two algorithms of Pereira and Urrutia [2018], `B&C Arc` and `B&C Node`. We refer to our branch-and-cut algorithm based on formulation (3.1)-(3.10) and inequalities from Section 3.4 as `B&C New`.

The proposed algorithm was implemented in the C++ programming language and compiled with GCC 4.8.4 using the optimization flag -O3. We implemented our algorithm using CPLEX 12.10 branch-and-cut framework. The algorithms were executed with a single thread. We disabled all preprocessing procedures, cut generation, and heuristics of the solver. We set Dual simplex as the default algorithm to solve the linear programs. Our computational experiments were carried out with an Intel i7 4790K 4.00GHz machine with 16GB of RAM available, under the Ubuntu 18.04 LTS operating system. Since the source codes from Côté et al. [2012a], Sampaio and Urrutia [2017], and Pereira and Urrutia [2018], were all kindly shared with us, all the algorithms were compiled and executed in the same computational environment. In our experiments, we set the maximum computational time to five hours (18000 seconds).

We use the set of benchmark instances proposed by Côté et al. [2012a] to evaluate the effectiveness of our branch-and-cut. The authors generated them based on the instances of the PDTSPL in Cordeau et al. [2010]. Two classes of instances were generated. Each class has a total of 54 instances, divided equally into 9 families. In the first class, C1, the size of each pickup item is 1, the number of stacks is a random number between 2 and 4, and the capacity of each stack is a random number between 1 and 3. In the second class, C2, the size of each pickup item 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.

Table 3.1 and 3.2 report the results of the algorithms considering all the instances in classes C1 and C2, respectively. The values displayed are the average of the results for all the instances in the corresponding class. The columns for each algorithm in the table represent: the average gap in the root node of the branch-and-bound tree (Gap_r), calculated as $(\text{BUB} - \text{LBR})/(\text{BUB})$, where BUB is the best upper bound known for the instance, and LBR is the lower bound obtained in such node; the average final gap (Gap_f), computed as $(\text{BUB} - \text{BLB})/(\text{BUB})$, where BLB is the best lower bound obtained by the algorithm during the search in the branch-and-bound tree; the average execution time, in seconds, of the algorithm (Time); the average number of explored

Table 3.1: Summarized results of all the algorithms for class C1.

Algorithm	Gap_r	Gap_f	Time	Nodes	Cuts	Solved
`B&C New`	3.88	0.50	2871.32	26841.24	5258.94	46 / 54
`B&C New - LIFO`	3.87	0.52	2889.80	23175.20	5706.80	46 / 54
`B&C New - CAPACITY`	4.32	0.73	4050.72	37543.20	7414.59	44 / 54
`B&C New - CUTPOOL`	4.91	0.63	3438.69	32552.80	6037.07	45 / 54
`B&C New - BINPACKING`	3.88	0.49	2871.97	26893.46	5261.22	46 / 54
`B&C New - ALL`	5.44	1.04	4800.52	41042.22	8730.81	41 / 54
`B&C Arc` [Pereira and Urrutia, 2018]	8.52	2.57	5181.18	53943.56	9909.46	40 / 54
`B&C Node` [Pereira and Urrutia, 2018]	8.52	1.62	5252.06	69087.67	8325.46	39 / 54
Côté et al. [2012a]	6.07	1.12	5609.30	28084.89	383002.33	40 / 54
Sampaio and Urrutia [2017]	8.84	3.09	6414.74	71993.04	10167.72	36 / 54

Table 3.2: Summarized results of all the algorithms for class C2.

Algorithm	Gap_r	Gap_f	Time	Nodes	Cuts	Solved
`B&C New`	4.74	0.44	2893.44	36052.41	5365.94	47 / 54
`B&C New - LIFO`	4.71	0.47	3102.01	35989.93	6151.07	47 / 54
`B&C New - CAPACITY`	5.24	0.76	4387.65	46928.98	8869.87	43 / 54
`B&C New - CUTPOOL`	5.64	0.56	3629.96	45187.63	6231.24	46 / 54
`B&C New - BINPACKING`	5.79	0.90	4586.46	56683.85	7836.33	42 / 54
`B&C New - ALL`	7.01	1.65	6905.64	78784.69	12645.91	36 / 54
`B&C Arc` [Pereira and Urrutia, 2018]	10.06	3.19	7785.17	101711.20	14455.85	33 / 54
`B&C Node` [Pereira and Urrutia, 2018]	10.20	2.84	7958.74	117587.00	12262.06	33 / 54
Côté et al. [2012a]	8.33	2.04	8727.75	59207.48	700194.59	32 / 54
Sampaio and Urrutia [2017]	10.35	3.52	8653.01	125171.61	13205.81	30 / 54

nodes in the branch-and-bound tree (Nodes); the average amount of cuts added in the entire branch-and-bound tree (Cuts); and the number of solved instances by each algorithm together with the total of instances in the class (Solved).

To evaluate the impact of the main components of our proposed algorithm, we removed each one of them from the final version. Namely, `B&C New - LIFO` removes inequalities (3.24), (3.25), and (3.33) from the algorithm; `B&C New - CAPACITY` removes inequalities (3.16), (3.17), (3.31), and (3.32); `B&C New - CUTPOOL` removes the cut pool from Section 3.5.4; `B&C New - BINPACKING` removes the lifting procedure based on bin packing from Section 3.5.8; `B&C New - ALL` removes all of the previous components.

We show in Tables 3.1 and 3.2 the results of the versions of the proposed algorithm. We can see that removing the cut pool from the root node of the branch-and-bound tree weakens the linear relaxation bound and decreases the total number of instances solved for both classes of instances. The same behavior is observed for the removal of the bin packing procedure for class C2, the gap in the root node increases and the algorithm solves 5 instances less than `B&C New`. Removing the proposed capacity inequalities also worsens algorithm `B&C New`. The removal of LIFO inequalities has no impact like the other components, but it negatively impacts the algorithm if removed together with all the previous components.

Analyzing the results of Tables 3.1 and 3.2 it is clear that the best of the algorithms is `B&C New`. For all the metrics evaluated it obtains the best results for both instance classes. The algorithm solves more instances than all other algorithms, especially in class C2, where the other algorithms solve fewer instances than in class C1. Besides, it obtains stronger dual bounds at the root node of the branch-and-bound tree and also at the end of the execution. The computational time for all classes is also smaller. As the algorithm of Côté et al. [2012a] adds the cuts locally, the amount of cuts is considerably greater compared to the other algorithms. Similar to the results obtained in Pereira and Urrutia [2018], comparing the existing algorithms from the literature, `B&C Arc` obtains the best overall results by solving more instances. Thus, in the next tables, we compare its results with our proposed `B&C New` algorithm.

We display in Tables 3.3 and 3.4 the results of `B&C New` and `B&C Arc` for class C1 and C2, respectively. The table columns have the same meaning as in Tables 3.1 and 3.2, but this time considering each specific instance. Moreover, we added a block with three columns to describe each instance. The column n indicates the number of requests. Therefore, the corresponding instance has $2n + 2$ vertices. The column BUB displays the best upper bound known for the instance. This value is the optimal solution if one of the algorithms has a final gap equal to zero. It is worth mentioning that the best primal bounds for the unsolved instances were obtained by our algorithm. If the instance has 18000 in its time column, the time limit was reached and the optimization aborted.

We can see the strength of our proposed formulation and inequalities through the fact that our algorithm obtains the best dual bounds at the root node of the branch-and-bound tree and the end of the algorithm for all the instances. We highlight that this also holds when compared with the other existing algorithms. For several instances, the gap of algorithm `B&C New` at the root of the branch-and-bound tree is smaller than the final gap obtained by `B&C Arc`. In addition to the stronger formulation, we see that the total number of enumerated nodes and added cuts is smaller for our algorithm. As we can see from the tables, our `B&C New` algorithm was able to drastically reduce the time needed to solve each instance. Reducing the time needed to reach optimality from several hours to just a few seconds.

In addition to solving all instances that `B&C Arc` solves, `B&C New` solves 20 more instances in total. Our branch-and-cut algorithm `B&C New` closed the difficulty gap that existed between classes C1 and C2 for the other algorithms. Unlike the other algorithms from the literature, our algorithm `B&C New` solves almost the same number

Table 3.3: Detailed results for the instances in class C1.

Instance			`B&C New`					`B&C Arc` [Pereira and Urrutia, 2018]				
Name	n	BUB	Gap$_r$	Gap$_f$	Time	Nodes	Cuts	Gap$_r$	Gap$_f$	Time	Nodes	Cuts
a280	11	449	2.38	0.00	0.27	46	219	9.63	0.00	0.40	157	408
	13	468	2.31	0.00	0.33	37	193	10.47	0.00	1.07	239	656
	15	542	2.60	0.00	0.56	34	246	13.28	0.00	15.14	1508	1945
	17	624	2.45	0.00	0.87	30	235	10.90	0.00	6.67	653	960
	19	669	2.67	0.00	2.08	72	364	10.24	0.00	53.13	2781	2525
	21	739	4.00	0.00	16.48	1079	1054	11.77	0.00	347.66	13314	4505
att532	11	4177	0.00	0.00	0.07	0	102	0.49	0.00	0.04	3	106
	13	4937	0.00	0.00	0.17	0	112	0.73	0.00	0.07	3	77
	15	5151	0.65	0.00	0.44	3	172	2.17	0.00	0.34	16	187
	17	5294	0.17	0.00	0.39	5	188	0.83	0.00	0.22	11	197
	19	5587	0.87	0.00	0.95	4	199	2.16	0.00	1.05	34	309
	21	9266	2.88	0.00	14.33	875	1111	3.45	0.00	77.44	5839	3732
brd14051	11	4396	3.57	0.00	0.84	113	384	3.74	0.00	1.17	283	533
	13	4439	3.94	0.00	3.13	389	670	4.48	0.00	4.66	728	969
	15	4797	6.89	0.00	498.37	35770	4530	10.16	2.66	18000.00	268075	17231
	17	4922	5.93	0.00	2763.55	97369	7388	11.99	6.97	18000.00	115085	26671
	19	6452	0.99	0.00	153.28	7218	3561	31.09	28.86	18000.00	85909	50039
	21	6733	1.74	0.69	18000.00	118796	18616	33.54	10.56	18000.00	54829	41368
d15112	11	74603	4.10	0.00	0.41	45	293	4.93	0.00	0.26	62	271
	13	80690	6.71	0.00	1.88	205	544	7.85	0.00	3.77	1320	752
	15	89754	3.40	0.00	1.51	98	489	4.97	0.00	5.53	1405	998
	17	96804	4.61	0.00	23.86	2148	1582	8.78	0.00	321.94	29339	4769
	19	103356	7.20	0.00	3177.93	100568	10411	10.84	3.43	18000.00	177864	17822
	21	108081	9.96	3.85	18000.00	112841	25584	14.21	7.73	18000.00	98445	27930
d18512	11	4280	0.00	0.00	0.12	0	246	1.02	0.00	0.17	18	219
	13	4301	0.10	0.00	0.25	4	214	1.20	0.00	0.42	46	330
	15	4638	4.77	0.00	10.94	1524	1171	6.99	0.00	129.79	15420	2769
	17	4741	5.28	0.00	59.13	4659	2847	8.44	0.00	3992.78	123967	9394
	19	4917	6.02	0.00	1026.47	40181	7496	9.85	0.97	18000.00	227930	17124
	21	5100	8.60	3.96	18000.00	160011	22532	12.60	8.34	18000.00	110206	28222
fnl4461	11	1889	0.00	0.00	0.15	0	192	0.44	0.00	0.08	37	228
	13	2088	0.00	0.00	0.39	0	240	2.27	0.00	0.68	243	678
	15	2356	1.36	0.00	1.52	24	560	6.58	0.00	228.20	40546	5441
	17	2517	1.36	0.00	3.63	79	856	9.37	0.00	7306.75	305653	15073
	19	2933	4.54	0.00	231.40	4772	8186	18.57	13.71	18000.00	217608	29383
	21	3561	3.93	0.53	18000.00	43199	42765	29.87	25.97	18000.00	115567	45324
nrw1379	11	2690	1.97	0.00	0.26	6	323	2.04	0.00	0.11	8	202
	13	3061	9.99	0.00	2915.82	102951	11428	10.24	0.00	11592.79	199662	17843
	15	3117	10.07	3.85	18000.00	174567	27949	10.67	4.05	18000.00	225373	23811
	17	3197	9.03	1.51	18000.00	186067	21370	10.24	3.91	18000.00	184910	24326
	19	3413	11.97	6.57	18000.00	137634	27688	14.27	10.14	18000.00	95038	45607
	21	3634	11.70	5.75	18000.00	106128	22609	15.09	11.25	18000.00	61546	49031
pr1002	11	13718	0.35	0.00	0.07	3	94	1.47	0.00	0.05	13	139
	13	15436	1.47	0.00	0.30	14	171	2.63	0.00	0.32	51	208
	15	16268	3.36	0.00	3.05	526	601	4.36	0.00	7.93	2011	876
	17	17601	3.16	0.00	4.37	345	756	4.23	0.00	15.55	2570	1216
	19	18673	3.20	0.00	8.68	669	926	4.50	0.00	21.46	2596	1087
	21	20150	3.27	0.00	19.27	1108	1219	4.33	0.00	93.10	7678	1765
ts225	11	22000	0.00	0.00	0.07	1	119	0.00	0.00	0.07	14	178
	13	29395	0.00	0.00	0.20	2	190	0.07	0.00	0.10	6	167
	15	32541	0.65	0.00	0.41	11	197	6.57	0.00	0.57	65	226
	17	36405	4.53	0.00	1.17	47	264	8.98	0.00	2.71	213	472
	19	40395	8.81	0.00	18.99	1638	975	14.61	0.00	387.20	28189	3077
	21	42878	9.83	0.00	83.03	5512	1552	15.66	0.00	3162.24	87866	5735

Table 3.4: Detailed results for the instances in class C2.

Instance			`B&C New`					`B&C Arc` [Pereira and Urrutia, 2018]				
Name	n	BUB	Gap$_r$	Gap$_f$	Time	Nodes	Cuts	Gap$_r$	Gap$_f$	Time	Nodes	Cuts
a280	11	455	3.45	0.00	0.36	66	217	11.76	0.00	0.84	361	512
	13	479	4.00	0.00	0.88	163	335	12.53	0.00	1.32	352	655
	15	553	3.99	0.00	2.08	233	475	15.01	0.00	17.04	2138	1890
	17	635	3.51	0.00	2.92	233	481	12.44	0.00	11.13	1198	1289
	19	677	3.29	0.00	4.09	254	500	11.74	0.00	33.17	1728	2421
	21	753	5.35	0.00	120.81	9713	2293	13.41	0.00	469.00	20834	4881
att532	11	4190	0.00	0.00	0.07	0	99	0.80	0.00	0.05	7	122
	13	5033	0.21	0.00	0.20	4	151	2.62	0.00	0.32	65	205
	15	5665	6.35	0.00	9.42	1318	947	11.05	0.00	1274.99	81587	7959
	17	5865	6.47	0.00	26.99	2717	1705	10.87	2.17	18000.00	233489	28060
	19	6184	6.99	0.00	100.65	6597	2967	11.69	3.35	18000.00	175469	31904
	21	9998	4.68	0.00	52.48	2440	1350	10.36	5.94	18000.00	94660	35528
brd14051	11	4386	3.39	0.00	0.75	106	383	3.52	0.00	0.84	278	412
	13	4458	3.16	0.00	3.95	452	736	4.83	0.00	29.65	7780	2043
	15	4795	8.35	0.00	9994.86	213182	14598	10.08	2.64	18000.00	254081	16500
	17	4889	7.48	1.29	18000.00	200716	19261	11.41	6.52	18000.00	157854	24724
	19	6274	2.37	0.00	30.83	1340	2493	29.25	26.99	18000.00	75457	45509
	21	6320	2.64	0.00	316.51	10061	6455	29.56	27.22	18000.00	72934	48441
d15112	11	73872	3.23	0.00	0.60	185	288	3.85	0.00	0.51	251	319
	13	81657	7.78	0.00	13.80	2855	1467	8.94	0.00	94.85	22595	3861
	15	91799	5.37	0.00	83.69	15968	2253	7.09	0.00	368.83	50576	6551
	17	97040	8.04	0.00	244.41	18482	4988	9.00	0.70	18000.00	381724	28817
	19	99660	6.51	0.00	376.56	15664	6214	7.53	0.00	7116.20	187521	21158
	21	104213	9.54	1.52	18000.00	130632	24171	11.15	3.97	18000.00	98356	23506
d18512	11	4341	0.00	0.00	0.12	2	141	2.41	0.00	5.91	2523	803
	13	4572	0.89	0.00	0.52	8	263	7.05	0.00	9939.80	521831	15468
	15	4893	4.69	0.00	34.48	3444	2117	11.80	3.66	18000.00	226858	19206
	17	5096	4.30	0.00	2075.23	70414	9030	14.80	6.03	18000.00	105240	27403
	19	5359	5.78	1.67	18000.00	209390	15282	17.29	10.11	18000.00	82044	35743
	21	5726	9.59	7.23	18000.00	122913	22286	22.16	17.95	18000.00	87505	43847
fnl4461	11	1883	0.16	0.00	0.12	11	164	0.16	0.00	0.06	13	153
	13	2088	1.92	0.00	1.79	435	600	2.18	0.00	6.49	2538	1365
	15	2262	1.89	0.00	1.73	237	468	2.81	0.00	25.97	6675	2286
	17	2428	4.26	0.00	41.97	4558	2630	6.14	0.00	2470.22	168073	10175
	19	2634	8.63	0.00	3947.73	106946	14122	9.32	4.29	18000.00	278274	26247
	21	2773	7.66	0.00	2284.65	53855	11365	9.94	3.76	18000.00	227438	23997
nrw1379	11	2690	1.28	0.00	0.29	10	318	2.06	0.00	0.13	12	234
	13	3055	9.84	0.00	2655.20	92720	12444	10.10	0.00	5024.02	225699	12224
	15	3108	9.64	2.81	18000.00	169794	28480	10.41	3.73	18000.00	242344	23098
	17	3197	7.13	0.00	7355.26	196598	14594	10.24	4.01	18000.00	226279	23672
	19	3421	8.66	4.50	18000.00	130877	25764	14.47	10.16	18000.00	92140	44006
	21	3769	7.68	4.87	18000.00	120668	27172	18.11	14.29	18000.00	58480	49231
pr1002	11	13527	1.35	0.00	0.16	35	152	1.92	0.00	0.24	127	279
	13	15221	2.36	0.00	0.84	181	264	2.89	0.00	3.83	1556	1217
	15	15676	1.87	0.00	0.89	100	282	2.33	0.00	6.77	1841	1784
	17	17009	1.73	0.00	3.46	499	517	2.29	0.00	28.31	4513	4541
	19	18136	2.45	0.00	7.33	554	868	3.04	0.00	102.46	9486	8623
	21	19613	2.40	0.00	14.45	906	1109	2.95	0.00	156.42	9814	11438
ts225	11	22000	0.00	0.00	0.09	5	114	0.00	0.00	0.09	33	289
	13	34000	3.08	0.00	0.34	9	200	13.61	0.00	188.89	34804	3720
	15	37703	5.73	0.00	1.27	51	307	19.36	0.00	1525.97	126138	7129
	17	41703	5.16	0.00	3.88	290	418	17.71	0.00	13494.63	443364	13526
	19	45703	8.62	0.00	43.22	3741	1152	21.59	5.59	18000.00	230165	16118
	21	49097	11.32	0.00	383.64	24198	2311	23.76	9.15	18000.00	155303	15527

Table 3.5: Summarized results for the instance families in class C1.

Family	`B&C New`						`B&C Arc` [Pereira and Urrutia, 2018]					
	Gap_r	Gap_f	Time	Nodes	Cuts	Solved	Gap_r	Gap_f	Time	Nodes	Cuts	Solved
a280	2.73	0.00	3.43	216.33	385.17	6 / 6	11.05	0.00	70.68	3108.67	1833.17	6 / 6
att532	0.76	0.00	2.73	147.83	314.00	6 / 6	1.64	0.00	13.19	984.33	768.00	6 / 6
brd14051	3.84	0.12	3569.86	43275.83	5858.17	5 / 6	15.83	8.17	12000.97	87484.83	22801.83	2 / 6
d15112	6.00	0.64	3534.27	35984.17	6483.83	5 / 6	8.60	1.86	6055.25	51405.83	8757.00	4 / 6
d18512	4.13	0.66	3182.82	34396.50	5751.00	5 / 6	6.68	1.55	6687.19	79597.83	9676.33	4 / 6
fnl4461	1.87	0.09	3039.52	8012.33	8799.83	5 / 6	11.18	6.61	7255.95	113275.67	16021.17	4 / 6
nrw1379	9.12	2.95	12486.01	117892.17	18561.17	2 / 6	10.42	4.89	13932.15	127756.17	26803.33	2 / 6
pr1002	2.47	0.00	5.96	444.17	627.83	6 / 6	3.59	0.00	23.07	2486.50	881.83	6 / 6
ts225	3.97	0.00	17.31	1201.83	549.50	6 / 6	7.65	0.00	592.15	19392.17	1642.50	6 / 6

Table 3.6: Summarized results for the instance families in class C2.

Family	`B&C New`						`B&C Arc` [Pereira and Urrutia, 2018]					
	Gap_r	Gap_f	Time	Nodes	Cuts	Solved	Gap_r	Gap_f	Time	Nodes	Cuts	Solved
a280	3.93	0.00	21.86	1777.00	716.83	6 / 6	12.82	0.00	88.75	4435.17	1941.33	6 / 6
att532	4.12	0.00	31.64	2179.33	1203.17	6 / 6	7.90	1.91	9212.56	97546.17	17296.33	3 / 6
brd14051	4.57	0.21	4724.48	70976.17	7321.00	5 / 6	14.78	10.56	12005.08	94730.67	22938.17	2 / 6
d15112	6.75	0.25	3119.84	30631.00	6563.50	5 / 6	7.93	0.78	7263.40	123503.83	14035.33	4 / 6
d18512	4.21	1.48	6351.73	67695.17	8186.50	4 / 6	12.59	6.29	13657.62	171000.17	23745.00	2 / 6
fnl4461	4.09	0.00	1046.33	27673.67	4891.50	6 / 6	5.09	1.34	6417.12	113835.17	10703.83	4 / 6
nrw1379	7.37	2.03	10668.46	118444.50	18128.67	3 / 6	10.90	5.37	12837.36	140825.67	25410.83	2 / 6
pr1002	2.03	0.00	4.52	379.17	532.00	6 / 6	2.57	0.00	49.67	4556.17	4647.00	6 / 6
ts225	5.65	0.00	72.07	4715.67	750.33	6 / 6	16.01	2.46	8534.93	164967.83	9384.83	4 / 6

of instances in both classes.

We group the results by the instance family and summarize them in Tables 3.5 and 3.6. The table columns represent the same information as in Table 3.1. The displayed values are the average of the results for all the instances in the corresponding family. The algorithm `B&C New` clearly outperformed `B&C Arc` for all instance families. We can see from Table 3.5 that all the metrics for all the instance families improved significantly. We can see that the families that have unsolved instances by `B&C New` are the same for classes C1 and C2. Moreover, family nrw1379 remains the most difficult family for the problem. This same family of instances is also very hard in other variants of the PDTSPMS.

3.7 Conclusion

This work addressed the pickup and delivery traveling salesman problem with multiple stacks (PDTSPMS). The main contributions of our work were a new branch-and-cut algorithm along with new valid inequalities. Some of these valid inequalities are lifted versions of existing inequalities for the PDTSPMS. New separation procedures for them were also presented. Besides, we show how to lift several capacity inequalities using a procedure based on the solution of a bin packing problem. The proposed branch-and-cut algorithm solves more instances from the benchmark than all the other algorithms from the literature. In addition to being able to solve more instances, our implemented algorithm obtained a better performance for all evaluated metrics. We showed that our proposed algorithm significantly outperforms the other exact algorithms from the literature for this problem. We pushed the boundary on what is now solvable in five hours for the PDTSPMS.

The algorithms developed in this work could be adapted to other routing problems and versions of the PDTSPMS. Using a fixing heuristic, we demonstrated that is practicable to couple the exact solution of a bin packing problem in our implemented algorithm. This shows that is possible to apply the lifting procedure used in this work for other routing problems that involve rounded capacity inequalities or the constraint of items not being divisible between stacks. Also, since the gap in the root node of the branch-and-bound tree can still be improved for most of the instances in the benchmark, new studies may involve proposing new valid inequalities for the problem.

There are still unsolved instances for the problem benchmark, making new approaches necessary to deal with the complexity of the problem. In addition to the previous proposals, we intend to develop algorithms based on column generation for the PDTSPMS. To the best of our knowledge, there is still no work in the literature using such an approach in an exact algorithm specific to the problem. All the exact approaches for the PDTSPMS are based on branch-and-cut algorithms. Consequently, we intend to apply a Dantzig-Wolfe reformulation using specific aspects of the problem.

Chapter 4

Conclusion and future research directions

We discuss in this chapter the final remarks about our work and point out further research directions we might follow. In the next section, we summarize the results obtained in this thesis. Following, we address possible future works regarding the studied problems.

4.1 Concluding remarks

We carried out the study of two combinatorial optimization problems in this work. Our study focused on formulations, valid inequalities, and exact algorithms for the considered problems. For both studied problems, our implemented approaches outperformed the previously existing methods from the literature. All computational experiments involving the algorithms were performed in the same computational environment, which allowed a direct comparison between the various approaches.

For the first problem, the p-arborescence star problem (p-ASP), with applications in the design of wireless sensor networks, we proposed two new formulations along with branch-and-cut algorithms based on them. For the branch-and-cut algorithms, we introduced preprocessing procedures that provide subsets of nodes for which at least one of them is a cluster-head. We use these subsets to create inequalities that are incorporated in the initial models used by the branch-and-cut algorithms. Also, we proved the theoretical result that finding a feasible solution to the problem is NP-hard. Our last contribution to the p-ASP was to improve the branch-and-cut algorithm from the literature with an exact polynomial-time separation algorithm.

We compared the algorithms for the p-ASP in the same computational setting. Our proposed version of the algorithm from the literature obtained better dual bounds at the root of the branch-and-bound tree than its previous counterpart that uses a heuristic separation algorithm. Also, it solved more instances of the benchmark. Regarding the branch-and-cut algorithms based on our proposed formulations, for small and medium-sized instances, our algorithms obtained shorter execution times, despite worse bounds in the root node the branch-and-bound tree. For the harder instances of the benchmark, the two best algorithms do not show dominance over the other, each of the two algorithms solves an instance that the other does not.

The second studied problem is the pickup and delivery traveling salesman problem with multiple stacks (PDTSPMS). We introduced a new formulation along with new valid inequalities. Several of these new valid inequalities are lifted versions from previous works addressing the problem and its variants. The derived inequalities focused on the capacity and LIFO policy aspects of the PDTSPMS. Using the formulation and inequalities as a starting point, we implemented a branch-and-cut algorithm. The devised algorithm also has several acceleration strategies. One of these acceleration strategies is based on the solution of a bin packing problem to lift several capacity inequalities. Although we solved the bin packing problem exactly, our implementation showed that it is possible to solve it in negligible time with the support of a fixation heuristic. As a result, we were able to integrate the procedure into our branch-and-cut algorithm without any computational loss.

Since the authors of the other exact algorithms for the PDTSPMS kindly shared their source codes with us, we compiled and executed all the algorithms using the same machine and tools. Our algorithm outperforms all other algorithms for the benchmark instances. The proposed algorithm solved more instances in total from the benchmark set, obtained the strongest dual bounds at the root node of the branch-and-bound tree for all instance families, and drastically reduced the time required to solve most instances of the benchmark. Moreover, for unsolved instances of the benchmark, it achieved the best upper bounds and optimality gaps.

4.2 Future research directions

As we discussed in Chapter 2, new approaches for the p-ASP may involve the study of methods that allow a faster solution of the linear programs obtained from relaxing the nodes in the branch-and-bound tree. One of the possible approaches is to apply a Lagrangian relaxation to existing formulations. However, at the moment, we have not progressed on any idea in this direction. To move in this direction, we believe that we need new inequalities to strengthen the formulations since the linear programs obtained are already solved in short times. The problem is also interesting for the development of heuristics, since finding a feasible solution is NP-hard. One approach we will consider in the future is to develop a Feasibility Pump heuristic. However, we believe that we still need to improve this contribution to pursue the direction. Finally, new preprocessing procedures might be investigated.

One of the main motivations for defining the p-ASP is to extend the lifetime of a dynamic network. Thus, an alternative line of study is to extend the problem to consider a rotation of nodes per period. In this case, a time interval where a set of p nodes remains as cluster-heads on the backbone can be considered, at the end of that period, p new nodes are chosen to be cluster-heads. By adding periods to the problem, we make it more representative of a real situation, since a node does not need to remain as a cluster-head throughout the whole network lifetime. With both problems at hand, a comparative study could evaluate if the inclusion of periods in the problem increases the network lifetime significantly.

Our main focus as a next research step for the PDTSPMS is the development of an exact algorithm based on column generation for the problem. To the best of our knowledge, there is still no work in the literature using such an approach in an exact algorithm for the problem. All exact approaches to the PDTSPMS are based on branch-and-cut algorithms. In the benchmark of the problem, there are still unsolved instances. Thus, it is interesting to study new exact approaches and analyze how they behave. Our initial idea is to apply a Dantzig-Wolfe decomposition on the precedence polytope.

Alongside the column generation investigation, we intend to study new valid inequalities and separation algorithms for the PDTSPMS. As we observed in the results of the computational experiments in Chapter 3, although our algorithm obtained the best dual bounds at the root node of the branch-and-bound tree, there is still a lot of margin to strengthen them. As we use the most efficient valid inequalities in the literature for the problem, we suppose that to improve the formulations, new valid inequalities are necessary. This same argument applies to the heuristic methods used to separate several inequalities. More efficient methods of separation and cut management can improve the dual bounds of the formulations.

While the tree [Magnanti and Wolsey, 1995], cycle [Grötschel and Padberg, 1979; Bauer, 1997; Ruland and Rodin, 1998], and precedence [Balas et al., 1995; Dumitrescu et al., 2010] polytopes are well studied, the LIFO polytope has not been investigated. Therefore, this is a valid and interesting venue for research, since facets for the LIFO polytope can streghten the formulations and improve the performance of algorithms for the PDTSPMS and its variants.

Even though the routing problem research field is rich in heuristics for several problems, the PDTSPMS has little work related to this topic. Thus, an alternative way of contributing to the problem literature is through the development of heuristics. The PDTSPMS is complex and has several aspects that must be taken into account for a route to be feasible. Thus, it is an interesting problem to propose and test the efficiency of heuristic approaches.

Recently, the concept of handling operations was introduced in problem variants of the PDTSPMS [Veenstra et al., 2017a; Chagas et al., 2020]. These operations allow items blocking access to be rearranged within the vehicle. It is usually considered a cost or time to reposition an item. Despite the fact that this aspect has been used for other variants, it has not yet been studied in the classic version of the PDTSPMS. For this reason, one line of research is to extend the problem with handling operations. In this case, it is also possible to conduct an investigation from an operations research perspective. The objective would be to assess whether repositioning items within the vehicle can decrease the total costs involved in operating the vehicle to meet customer requests.

Bibliography

Abbasi, A. A. and Younis, M. (2007). A survey on clustering algorithms for wireless sensor networks. *Computer Communications*, 30(14):2826--2841.

Akyildiz, I., Su, W., Sankarasubramaniam, Y., and Cayirci, E. (2002). Wireless sensor networks: a survey. *Computer Networks*, 38(4):393--422.

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

Azi, N., Gendreau, M., and Potvin, J.-Y. (2007). An exact algorithm for a single-vehicle routing problem with time windows and multiple routes. *European Journal of Operational Research*, 178(3):755--766.

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

Baldacci, R., Dell'Amico, M., and González, J. S. (2007). The capacitated m-ring-star problem. *Operations Research*, 55(6):1147--1162.

Barbato, M., Grappe, R., Lacroix, M., and Wolfler Calvo, R. (2016). Polyhedral results and a branch-and-cut algorithm for the double traveling salesman problem with multiple stacks. *Discrete Optimization*, 21:25--41.

Bauer, P. (1997). The circuit polytope: Facets. *Mathematics of Operations Research*, 22(1):110--145.

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

Calik, H., Leitner, M., and Luipersbeck, M. (2017). A benders decomposition based framework for solving cable trench problems. *Computers & Operations Research*, 81:128--140.

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., Cerulli, R., and Speranza, M. G. (2013). A branch-and-bound algorithm for the double travelling salesman problem with two stacks. *Networks*, 61(1):58--75.

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.

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.

Chagas, J. B. C., Toffolo, T. A. M., Souza, M. J. F., and Iori, M. (2020). The double traveling salesman problem with partial last-in-first-out loading constraints. *International Transactions in Operational Research.*

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

Cherkesly, M., Desaulniers, G., Irnich, S., and Laporte, G. (2016). Branch-price-and-cut algorithms for the pickup and delivery problem with time windows and multiple stacks. *European Journal of Operational Research*, 250(3):782--793.

Cherkesly, M., Desaulniers, G., and Laporte, G. (2014). Branch-price-and-cut algorithms for the pickup and delivery problem with time windows and last-in-first-out loading. *Transportation Science*, 49(4):752--766.

Chu, Y. J. and Liu, T. H. (1965). On the shortest arborescence of a directed graph. *Scientia Sinica*, 14:1396--1400.

Cook, W., Cunningham, W., Pulleyblank, W., and Schrijver, A. (2011). *Combinatorial Optimization.* Wiley Series in Discrete Mathematics and Optimization. Wiley.

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

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.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms*. The MIT Press, 3rd edition.

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.

Dinic, E. A. (1970). Algorithm for solution of a problem of maximum flow in networks with power estimation. In *Soviet Mathematics Doklady*, volume 11, pages 1277--1280.

Duarte-Melo, E. J. and Liu, M. (2003). Data-gathering wireless sensor networks: organization and capacity. *Computer Networks*, 43(4):519--537.

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.

Edmonds, J. (1967). Optimum branchings. *Journal of Research of the national Bureau of Standards B*, 71(4):233--240.

Even, S. (2011). *Graph Algorithms*. Cambridge University Press, 2 edition.

Fujie, T. (2003). An exact algorithm for the maximum leaf spanning tree problem. *Computers & Operations Research*, 30(13):1931--1944.

Fujie, T. (2004). The maximum-leaf spanning tree problem: Formulations and facets. *Networks*, 43(4):212--223.

Garey, M. R. and Johnson, D. S. (1990). *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA.

Gendron, B., Lucena, A., da Cunha, A. S., and Simonetti, L. (2014). Benders decomposition, branch-and-cut, and hybrid algorithms for the minimum connected dominating set problem. *INFORMS Journal on Computing*, 26(4):645--657.

Golden, B., Naji-Azimi, Z., Raghavan, S., Salari, M., and Toth, P. (2012). The generalized covering salesman problem. *INFORMS Journal on Computing*, 24(4):534--553.

Gollowitzer, S. and Ljubić, I. (2011). Mip models for connected facility location: A theoretical and computational study. *Computers & Operations Research*, 38(2):435--449.

Gomory, R. E. (1958). Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64(5):275--278.

Gouveia, L. and Simonetti, L. (2017). Spanning trees with a constraint on the number of leaves. a new formulation. *Computers & Operations Research*, 81:257--268.

Grötschel, M. and Padberg, M. W. (1979). On the symmetric travelling salesman problem i: inequalities. *Mathematical Programming*, 16(1):265--280.

Hakimi, S. L. (1965). Optimum distribution of switching centers in a communication network and some related graph theoretic problems. *Operations Research*, 13(3):462--475.

Heinzelman, W., Chandrakasan, A., and Balakrishnan, H. (2000). Energy efficient communication protocol for wireless microsensor networks. *33rd Hawaii International Conference on System Sciences*, pages 8020--8030.

Heinzelman, W. B., Chandrakasan, A. P., and Balakrishnan, H. (2002). An application-specific protocol architecture for wireless microsensor networks. *IEEE Transactions on wireless communications*, 1(4):660--670.

Kek, A. G., Cheu, R. L., and Meng, Q. (2008). Distance-constrained capacitated vehicle routing problems with flexible assignment of start and end depots. *Mathematical and Computer Modelling*, 47(1-2):140--152.

Khuller, S. and Zhu, A. (2002). The general steiner tree-star problem. *Information Processing Letters*, 84(4):215--220.

Labbé, M., Laporte, G., Martín, I. R., and González, J. J. S. (2004). The ring star problem: Polyhedral analysis and exact algorithm. *Networks*, 43(3):177--189.

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.

Land, A. H. and Doig, A. G. (1960). An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497--520.

Leitner, M., Ljubić, I., Salazar-González, J.-J., and Sinnl, M. (2017). An algorithmic framework for the exact solution of tree-star problems. *European Journal of Operational Research*, 261(1):54--66.

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

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.

Lucena, A., Maculan, N., and Simonetti, L. (2010). Reformulations and solution algorithms for the maximum leaf spanning tree problem. *Computational Management Science*, 7(3):289--311.

Lucena, A., Simonetti, L., and da Cunha, A. S. (2016). The tree-star problem: A formulation and a branch-and-cut algorithm. *Electronic Notes in Discrete Mathematics*, 52:285--292. ISSN 1571–0653. INOC 2015 – 7th International Network Optimization Conference.

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.

Magnanti, T. L. and Wolsey, L. A. (1995). Chapter 9 optimal trees. In *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science*, pages 503--615. Elsevier.

Magnanti, T. L. and Wong, R. T. (1984). Network design and transportation planning: Models and algorithms. *Transportation Science*, 18(1):1--55.

Marianov, V., Gutiérrez-Jarpa, G., Obreque, C., and Cornejo, O. (2012). Lagrangean relaxation heuristics for the p-cable-trench problem. *Computers & Operations Research*, 39(3):620--628.

Matos, V. O., Arroyo, E. C., and Gonçalves, L. B. (2012). An energy-efficient clustering algorithm for wireless sensor networks. volume 12, pages 6--15.

Morais, V., Gendron, B., and Mateus, G. R. (2019). The p-arborescence star problem: Formulations and exact solution approaches. *Computers & Operations Research*, 102:91--101.

Morais, V. and Mateus, G. R. (2019). Configuration-based approach for topological problems in the design of wireless sensor networks. *International Transactions in Operational Research*, 26(3):836--855.

Morrison, D. R., Jacobson, S. H., Sauppe, J. J., and Sewell, E. C. (2016). Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79--102.

Nuggehalli, P., Srinivasan, V., and Chiasserini, C.-F. (2003). Energy-efficient caching strategies in ad hoc wireless networks. In *Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking &Amp; Computing*, MobiHoc '03, pages 25--34, New York, NY, USA. ACM.

Padberg, M. W. and Wolsey, L. A. (1983). Trees and cuts. In Berge, C., Bresson, D., Camion, P., Maurras, J., and Sterboul, F., editors, *Combinatorial Mathematics*, volume 75 of *North-Holland Mathematics Studies*, pages 511--517. North-Holland.

Pereira, A. H., Mateus, G. R., and Urrutia, S. (2020). Branch-and-cut algorithms for the *p*-arborescence star problem. *International Transactions in Operational Research.*

Pereira, A. H. and Urrutia, S. (2018). Formulations and algorithms for the pickup and delivery traveling salesman problem with multiple stacks. *Computers & Operations Research*, 93:1--14.

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. (2009). The double travelling salesman problem with multiple stacks – formulation and heuristic solution approaches. *European Journal of Operational Research*, 198(1):139--147.

Pollaris, H., Braekers, K., Caris, A., Janssens, G. K., and Limbourg, S. (2015). Vehicle routing problems with loading constraints: state-of-the-art and future directions. *OR Spectrum*, 37(2):297--330.

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

Ruland, K. and Rodin, E. (1998). Survey of facial results for the traveling salesman polytope. *Mathematical and Computer Modelling*, 27(8):11--27.

Sampaio, A. H. and Urrutia, S. (2017). New formulation and branch-and-cut algorithm for the pickup and delivery traveling salesman problem with multiple stacks. *International Transactions in Operational Research*, 24(1-2):77--98.

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *International conference on principles and practice of constraint programming*, pages 417--431. Springer.

Simonetti, L., Da Cunha, A. S., and Lucena, A. (2011). The minimum connected dominating set problem: Formulation, valid inequalities and a branch-and-cut algorithm. In *Proceedings of the 5th International Conference on Network Optimization*, INOC'11, pages 162--169, Berlin, Heidelberg. Springer-Verlag.

Simonetti, L., Frota, Y., and de Souza, C. (2009). Upper and lower bounding procedures for the minimum caterpillar spanning problem. *Electronic Notes in Discrete Mathematics*, 35:83--88.

Swamy, C. and Kumar, A. (2004). Primal-dual algorithms for connected facility location problems. *Algorithmica*, 40(4):245--269.

Urrutia, S., Milanés, A., and Løkketangen, A. (2015). A dynamic programming based local search approach for the double traveling salesman problem with multiple stacks. *International Transactions in Operational Research*, 22(1):61--75.

Vasko, F. J., Barbieri, R. S., Rieksts, B. Q., Reitmeyer, K. L., and Stott, K. L. (2002). The cable trench problem: combining the shortest path and minimum spanning tree problems. *Computers & Operations Research*, 29(5):441--458.

Veenstra, M., Cherkesly, M., Desaulniers, G., and Laporte, G. (2017a). The pickup and delivery problem with time windows and handling operations. *Computers & Operations Research*, 77:127--140.

Veenstra, M., Roodbergen, K. J., Vis, I. F., and Coelho, L. C. (2017b). The pickup and delivery traveling salesman problem with handling costs. *European Journal of Operational Research*, 257(1):118--132.

Wolsey, L. (1998). *Integer Programming*. Wiley Series in Discrete Mathematics and Optimization. Wiley.

Younis, M. and Akkaya, K. (2008). Strategies and techniques for node placement in wireless sensor networks: A survey. *Ad Hoc Networks*, 6(4):621--655.

Ángel Felipe, Ortuño, M. 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.