Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Biológicas

Mathematical models and heuristic algorithms for routing problems with multiple interacting components

Jonatas Batista Costa das Chagas

Ouro Preto, Brazil
April 2021

Jonatas Batista Costa das Chagas

Mathematical models and heuristic algorithms for routing problems with multiple interacting components

Advisor:
Prof. Ph.D. Marcone Jamilson Freitas Souza

Co-advisor:
Prof. Ph.D. André Gustavo dos Santos

Thesis presented to the *Universidade Federal de Ouro Preto* in partial fulfillment of the requirements for the Degree of Doctor of Philosophy in Computer Science

Ouro Preto, Brazil
April 2021

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

C433m Chagas, Jonatas Batista Costa das .
Mathematical models and heuristic algorithms for routing problems with multiple interacting components. [manuscrito] / Jonatas Batista Costa das Chagas. - 2021.
132 f.: il.: color., gráf., tab..

Orientador: Prof. Dr. Marcone Jamilson Freitas Souza.
Coorientador: Prof. Dr. André Gustavo dos Santos.
Tese (Doutorado). Universidade Federal de Ouro Preto. Departamento de Computação. Programa de Pós-Graduação em Ciência da Computação.
Área de Concentração: Ciência da Computação.

1. Programação linear. 2. Heurística. 3. Otimização combinatória . I. Santos, André Gustavo dos. II. Souza, Marcone Jamilson Freitas. III. Universidade Federal de Ouro Preto. IV. Título.

CDU 004:51

Bibliotecário(a) Responsável: Celina Brasil Luiz - CRB6-1589

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E BIOLOGICAS
DEPARTAMENTO DE COMPUTACAO
PROGRAMA DE POS-GRADUACAO EM CIENCIA DA COMPUTACAO

FOLHA DE APROVAÇÃO

Jonatas Batista Costa das Chagas

Mathematical models and heuristic algorithms for routing problems with multiple interacting components

Tese apresentada ao Programa de Pós Graduação em Ciência da Computação da Universidade Federal de Ouro Preto como requisito parcial para obtenção do título de Doutor

Aprovada em 23 de abril de 2021

Membros da banca

Prof. Dr. Marcone Jamilson Freitas Souza - Orientador Universidade Federal de Ouro Preto
Prof. Dr. André Gustavo dos Santos - Universidade Federal de Viçosa
Prof. Dr. Eduardo Uchoa Barboza - Universidade Federal Fluminense
Prof. Dr. Jose Elias Claudio Arroyo - Universidade Federal de Viçosa
Prof. Dr. Thibaut Vidal - Pontifícia Universidade Católica Rio de Janeiro
Prof. Dr. Túlio Ângelo Machado Toffolo - Universidade Federal de Ouro Preto

Prof. Dr. Marcone Jamilson Freitas Souza, orientador do trabalho, aprovou a versão final e autorizou seu depósito no Repositório Institucional da UFOP em 27/05/2021

Documento assinado eletronicamente por **Puca Huachi Vaz Penna**, **COORDENADOR(A) DO CURSO DE PÓS-GRADUACÃO EM CIÊNCIA DA COMPUTAÇÃO**, em 14/06/2021, às 16:13, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0178216** e o código CRC **4844475C**.

Abstract

Many real-world optimization problems have multiple interacting components. Each of these can be an $\mathcal{NP}$-hard problem, and they can be in conflict with each other, i.e., the optimal solution for one component does not necessarily represent an optimal solution for the other components. This can be a challenge once the influence that each component has on the overall solution quality. In this work, we address four complex routing problems with multiple interacting components: the Double Vehicle Routing Problem with Multiple Stacks (DVRPMS), the Double Traveling Salesman Problem with Partial Last-In-First-Out Loading Constraints (DTSPPL), the Traveling Thief Problem (TTP), and the Thief Orienteering Problem (ThOP). While DVRPMS and TTP are already well-known in the literature, DTSPPL and ThOP have recently been proposed in order to introduce and study more realistic variants of DVRPMS and TTP, respectively. The DTSPPL has been proposed as part of this work, while the ThOP has independently been proposed. We present mathematical models and/or heuristic algorithms for solving these problems. Among the results achieved, we can highlight that our mathematical model proposed for the DVRPMS has been able to find better results than those found in the literature. In addition, we have won the first and second places in two recent optimization competitions on a bi-objective version of the TTP. In general, the results achieved by our solutions methods have shown better than those previously presented in the literature considering each problem studied in this work.

Resumo

Muitos problemas de otimização com aplicações reais têm vários componentes de interação. Cada um deles pode ser um problema pertencente à classe $\mathcal{NP}$-difícil, e eles podem estar em conflito um com o outro, ou seja, a solução ótima para um componente não representa necessariamente uma solução ótima para os outros componentes. Isso pode ser um desafio devido à influência que cada componente tem na qualidade geral da solução. Neste trabalho, foram abordados quatro problemas de roteamento complexos com vários componentes de interação: o *Double Vehicle Routing Problem with Multiple Stacks* (DVRPMS), o *Double Traveling Salesman Problem with Partial Last-In-First-Out Loading Constraints* (DTSPPL), o *Traveling Thief Problem* (TTP) e *Thief Orienteering Problem* (ThOP). Enquanto os DVRPMS e TTP já são bem conhecidos na literatura, os DTSPPL e ThOP foram recentemente propostos a fim de introduzir e estudar variantes mais realistas dos DVRPMS e TTP, respectivamente. O DTSPPL foi proposto a partir deste trabalho, enquanto o ThOP foi proposto de forma independente. Neste trabalho são propostos modelos matemáticos e/ou algoritmos heurísticos para a solução desses problemas. Dentre os resultados alcançados, é possível destacar que o modelo matemático proposto para o DVRPMS foi capaz de encontrar inconsistências nos resultados dos algoritmos exatos previamente propostos na literatura. Além disso, conquistamos o primeiro e o segundo lugares em duas recentes competições de otimização combinatória que tinha como objetivo a solução de uma versão bi-objetiva do TTP. Em geral, os resultados alcançados por nossos métodos de soluções mostraram-se melhores do que os apresentados anteriormente na literatura considerando cada problema investigado neste trabalho.

Acknowledgments

I would like to express my greatest thanks to my parents, João Batista and Adelma, and my sister, Jaqueline, for their wise counsel. They have always supported me and given me the strength to continue towards my goals. To Bruna Vilela, I am grateful for her fondness, for always listening to my complaints, and for celebrating with me my personal and academic achievements. I love you all *demais da conta*[1]!

Throughout the writing of this thesis, I have received great assistance. I would like to acknowledge my advisors, Prof. Ph.D. Marcone J. F. Souza, and Prof. Ph.D. André G. Santos, for their support and guidance over these years. I would also like to thank all the authors who have contributed to the research papers produced from this work, in particular, to Prof. Ph.D. Markus Wagner for his great collaboration in some of my projects.

I would like to thank Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), and Universidade Federal de Ouro Preto (UFOP) for funding this project. I thank the Universidade Federal de Viçosa (UFV) for receiving me as a collaborating researcher over these last two years. I could not but offer up my thanks to the Hasso-Plattner-Institut (HPI) Future SOC Lab, the Divisão de Suporte ao Desenvolvimento Científico e Tecnológico (DCT/UFV), and the Programa de Pós-graduação em Ciência da Computação (PPGCC/UFOP) for enabling this research by providing access to their computing infrastructure.

[1] It is a slang from the inhabitants of the state of Minas Gerais in Brazil that means “so much".

Agradecimentos

Gostaria de expressar os meus maiores agradecimentos aos meus pais, João Batista e Adelma, e à minha irmã, Jaqueline, pelos sábios conselhos. Eles sempre me apoiaram e me deram força para continuar seguindo os meus objetivos. À Bruna Vilela, agradeço o carinho, por sempre ouvir minhas reclamações e por comemorar comigo minhas conquistas pessoais e acadêmicas. Amo vocês demais da conta!

Ao longo da redação desta tese, recebi grande ajuda. Gostaria de agradecer aos meus orientadores, Prof. Dr. Marcone J. F. Souza e Prof. Dr. André G. Santos, pelo apoio e orientação ao longo desses anos. Gostaria também de agradecer a todos os autores que contribuíram nos artigos produzidos neste trabalho, em particular ao Prof. Ph.D. Markus Wagner pela sua enorme colaboração em alguns de meus projetos.

Gostaria de agradecer a Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) e Universidade Federal de Ouro Preto (UFOP) pelo financiamento deste projeto. Agradeço à Universidade Federal de Viçosa (UFV) por me receber como pesquisador colaborador nestes últimos dois anos. Não poderia deixar de agradecer ao Hasso-Plattner-Institut (HPI) Future SOC Lab, à Divisão de Suporte ao Desenvolvimento Científico e Tecnológico (DCT/UFV) e ao Programa de Pós-Graduação em Ciência da Computação (PPGCC/UFOP) por viabilizar esta pesquisa, fornecendo acesso às suas infraestruturas computacionais.

Contents

"Guarda i girasoli: loro si inchinano al sole, ma se uno è troppo inchinato vuol dire che è morto. Tu sei un servitore, non un servo. Servire è l'arte suprema. Dio è il primo servitore; Lui è il servitore di tutti gli uomini, ma non è il servo di nessuno."

— Dal film *La vita è bella*

Chapter 1

Introduction

Optimization problems are frequently investigated due to their theoretical and practical relevance. Many studies devoted to solving classical combinatorial optimization problems can be directly applied to solving real-world problems, such as vehicle routing problems (Toth and Vigo, 2014), scheduling problems (Pinedo, 2012), and packing problems (Sweeney and Paternoster, 1992). While this is true, many other real-world problems exhibit multi-component structures, i.e., they consist of several combinatorial optimization problems that interact with each other. These problems are difficult to solve not only because of the contained hard optimization problems, but in particular, because of the interdependencies between the different components. Interdependence complicates decision-making by forcing each sub-problem to influence the quality and feasibility of solutions of the other sub-problems (Bonyadi et al., 2019). These problems are also known and referred to as multi-attribute problems (Vidal et al., 2013, 2014). Examples of multi-component/multi-attribute problems are vehicle routing problems under loading constraints (Iori and Martello, 2010; Pollaris et al., 2015), maximizing material utilization while respecting a production schedule (Cheng et al., 2016; Wang, 2020), and relocation of containers in a port while minimizing idle times of ships (Forster and Bortfeldt, 2012; Jin et al., 2015; Hottung et al., 2020).

In this work, we focus on the class of vehicle routing problems once they subjectively appear to be very prominent. Indeed, among the global logistics processes, the transportation processes of goods and services between the industry and the end-user stand out due to their high factor in logistics costs in most companies (Aggelakakis et al., 2015). In general terms, transportation logistics aims to develop optimized route plans that minimize the costs necessary to meet customer demands, respecting all constraints imposed by the context of each company.

Dantzig and Ramser (1959) were pioneers to study routing problems. Their study have considered a problem that concerned to find the shortest routes for a fleet of trucks in order to serve the demand for oil of several gas stations from a single central distributor. Nowadays, this problem is known and referenced as the Vehicle Routing Problem (VRP), which is one of the most studied combinatorial optimization problems. Since Dantzig and Ramser's study, many extensions and variations of the problem have been introduced by the scientific community. The interest in these problems is due to their logistic importance as well as their theoretical relevance in the field of combinatorial optimization, which has resulted in a large number of studies about them (Eksioglu et al., 2009; Vidal et al., 2020). Since transport planning problems have combinatorial behavior (see Lovász (2007) for useful reference), finding the best plan among all possibilities is an arduous task in most real situations, especially because such situations usually involve interdependencies between the different components in transport logistics.

We tackle four problems with different characteristics and difficulties; however, they all have multi-components in their structures. The first two problems are pickup and delivery problems with loading constraints, while the last two problems are nonlinear problems that combine classic combinatorial optimization problems in their formulations. These four problems are named as Double Vehicle Routing Problem with Multiple Stacks (DVRPMS), Double Traveling Salesman Problem with Partial Last-In-First-Out Loading Constraints (DTSPPL), Traveling Thief Problem (TTP), and Thief Orienteering Problem (ThOP). While DVRPMS and TTP are already well-known in the literature, DTSPPL and ThOP have recently been proposed with the aim of studying more realistic variants of DVRPMS and TTP, respectively. The DTSPPL has been proposed from this thesis, while the ThOP has independently been proposed by Santos and Chagas (2018). In the following chapters, we describe in detail these problems, the investigations in the literature on them, as well as our contributions.

1.1 Contributions

The contribution of this thesis consists of developing computational methods to solve four combinatorial optimization problems with practical and theoretical relevance. All problems investigated follow the trends of recent years, which consider increasingly

realistic, integrated, and complex aspects. In the following, we describe the main contribution regarding each problem:

- DVRPMS: We have developed a new integer linear programming formulation and a heuristic algorithm based on the Variable Neighborhood Search (VNS) metaheuristic. Our solution approaches have been able to find better results than those previously proposed in the literature.
- DTSPPL: We have introduced this problem and proposed two integer linear programming formulations and a heuristic algorithm based on the Biased Random-Key Genetic Algorithm (BRKGA) to address it. These contributions open perspectives for further research about the DTSPPL and also its variants and extensions.
- TTP: We have developed two heuristic algorithms for a bi-objective formulation of the TTP. The first algorithm has been based on the BRKGA and the Non-Dominated Sorting Genetic Algorithm II (NSGA-II). By proposing this algorithm, we have won the first and second places, respectively, in the competitions held in 2019 at Evolutionary Multi-Criterion Optimization (EMO2019)[1] and The Genetic and Evolutionary Computation Conference (GECCO2019)[2]. More recently, we have proposed a weighted-sum method combined with a two-stage heuristic for solving the problem. This last algorithm has reached significantly better results than those presented in the competitions.
- ThOP: It has recently been proposed in 2018 at Congress on Evolutionary Computation (CEC2018) as a more realistic formulation of the TTP. We have presented mathematical formulations for it, as well as heuristic algorithms based on BRKGA, Iterated Local Search (ILS), and Ant Colony Optimization (ACO) techniques. Our algorithms have shown able to find high-quality solutions.

The research presented in this thesis has resulted in the following papers:

- **J. B. C. Chagas**, U. E. F. Silveira, A. G. Santos, and M. J. F. Souza. A variable neighborhood search heuristic algorithm for the double vehicle routing problem with multiple stacks. International Transactions in Operational Research, 27.1 (2020): 112-137. Available at https://doi.org/10.1111/itor.12623

[1] *EMO-2019* https://www.egr.msu.edu/coinlab/blankjul/emo19-thief/
[2] *GECCO-2019* https://www.egr.msu.edu/coinlab/blankjul/gecco19-thief/

- **J. B. C. Chagas**, T. A. M. Toffolo, M. J. F. Souza, and M. Iori. The double traveling salesman problem with partial last-in-first-out loading constraints. International Transactions in Operational Research, (2020). Available at https://doi.org/10.1111/itor.12876
- **J. B. C. Chagas**, J. Blank, M. Wagner, M. J. F. Souza, and K. Deb. A non-dominated sorting based customized random-key genetic algorithm for the bi-objective traveling thief problem. Journal of Heuristics, (2020). Available at https://doi.org/10.1007/s10732-020-09457-7
- **J. B. C. Chagas**, M. Wagner. A weighted-sum method for solving the bi-objective traveling thief problem. **Submitted to a journal**.
- **J. B. C. Chagas**, and M. Wagner. Ants can orienteer a thief in their robbery. Operations Research Letters, 48.6 (2020): 708-714. Available at https://doi.org/10.1016/j.orl.2020.08.011
- **J. B. C. Chagas**, and M. Wagner. Efficiently solving the thief orienteering problem with a max-min ant colony optimization approach. **Submitted to a journal**.

1.2 Document organization

The remaining of this document is primarily based on four research papers. Each paper describes the last results and contributions we have reached on each problem investigated. Chapters 2, 3, 4, and 5 are compiled from the content of the aforementioned papers, with some minor changes and reference updates when needed. Specifically, Chapter 2 covers the DVRPMS, Chapter 3 covers the DTSPPL, while Chapters 4, and 5 cover, respectively, the TTP and the ThOP. In Chapter 6, we present our final conclusions and suggest further investigations.

Chapter 2

The double vehicle routing problem with multiple stacks

In this chapter[1], we address the Double Vehicle Routing Problem with Multiple Stacks (DVRPMS). It arose in its simplest form as the Double Traveling Salesman Problem with Multiple Stacks (DTSPMS), proposed by Petersen and Madsen (2009) when a software company that set up routes in its intermodal traffic encountered this problem in the context of one of its customers.

In the DTSPMS, a single vehicle, which has its load compartment (container) divided into rows (horizontal stacks) of fixed depth (horizontal heights), must collect all items spread in a region known as pickup region and, henceforth, deliver all these collected items in another region, denominated delivery region. All items have the same size and shape. The items are stored in the stacks as they are collected. It is important to state that the items are not stored on top of each other, they are arranged in the same plane (container base) according to rows (horizontal stacks) and their depths (horizontal heights). The items' positions cannot be changed when they are already inside the container, that is, the items should be stationary until their unloading. The delivery must then respect the Last-In-First-Out (LIFO) policy. Thus, the delivery route is limited by the stack configurations set up by the pickup route.

The DTSPMS is a variation of the Pickup and Delivery Traveling Salesman Problem with Multiple Stacks (PDTSPMS) (Cordeau et al., 2010; Côté et al., 2012; Sampaio and

[1]It has been compiled from paper "A variable neighborhood search heuristic algorithm for the double vehicle routing problem with multiple stacks". J. B. C. Chagas, U. E. F. Silveira, A. G. Santos, and M. J. F. Souza. International Transactions in Operational Research, 27.1 (2020): 112-137. Available at https://doi.org/10.1111/itor.12623

Urrutia, 2016), where the pickup and delivery operations must be completely separate. This is due to the fact that the DTSPMS arises in the context where the pickup and delivery regions are widely separated. In this way, all items in the pickup region must be gathered prior to any unloading in the delivery region. The transportation cost between the two regions is fixed, and it is not considered as part of the optimization problem. The problem has applicability in deliveries where items are loaded and unloaded from the rear of the vehicle and item relocation is prohibited due to the fact that they are heavy, fragile, and/or their handling is dangerous. Its objective is to find two Hamiltonian cycles, one for the pickup region and the other for the delivery region, so that the sum of the distances traveled in both regions is the minimum possible, respecting the precedence constraints imposed by the vehicle's stacks.

Figure 2.1 depicts a feasible solution for the DTSPMS in a case involving 16 items. Each item is associated with a pickup client and a delivery client (item 1 is associated with pickup client 1 and with delivery client 1, item 2 is associated with pickup client 2 and with delivery client 2, and so on). The container of this vehicle is divided into two stacks of height eight, that is, the container has dimensions (2×8). The vehicle always starts its route in the pickup region at the vertex 0 (depot) and, in this example, the vehicle visits the customer 15, collects and stores the item in the first stack, then visits the customer 1, storing the item in the first stack (on top of item 15). Next, customer 7 is visited and his item stored on the (bottom of the) second stack. The gathering continues as shown in the figure until all customers have been served and their items stored in the vehicle container. At the end, the vehicle returns to the depot from where the entire container is transported to the delivery region depot. In the delivery region, the container is loaded into a vehicle that also always starts its route at the vertex 0 (depot) and, in this example, from the depot, the vehicle has only two possible customers to visit since only items 6 and 16 are accessible from the top of the two stacks. As illustrated in the figure, the vehicle initially satisfies the customer 16, unloads its item from the second stack, and the customer 12 becomes available to be served. The process continues as illustrated, always satisfying a customer who has its item on top of any stack. At the end of delivery, the vehicle returns to the depot in its respective region (in this case, the delivery region).

Petersen and Madsen (2009) proposed the DTSPMS, presented a mathematical formulation for the problem, and also proposed four heuristic approaches to solve it. The four heuristic approaches have been based on the Iterated Local Search (ILS), Tabu Search (TS), Simulated Annealing (SA), and Large Neighborhood Search (LNS)

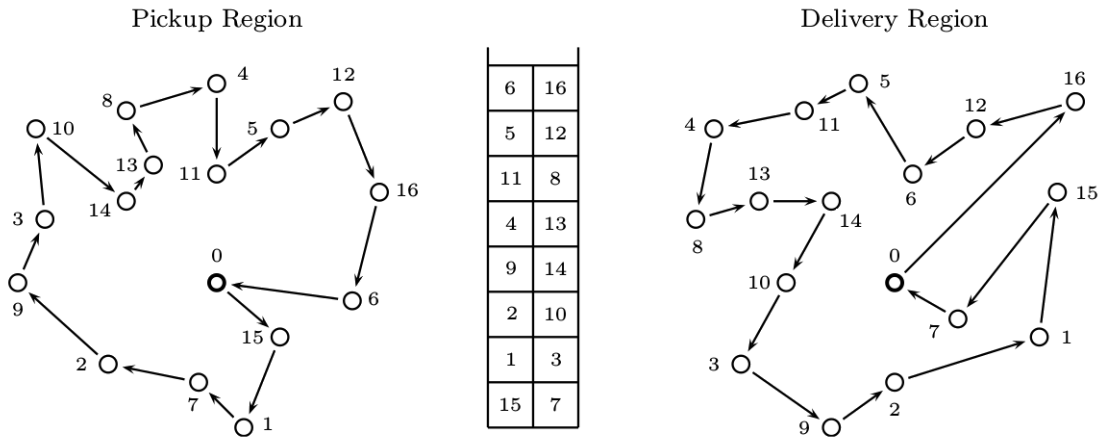


Figure 2.1: DTSPMS example - Adapted from Iori and Riera-Ledesma (2015).

metaheuristics. These heuristics have been tested on a set of instances that became the reference for future works. Among the proposed heuristics, the one based on the LNS had a better overall performance.

Since it was presented, the DTSPMS aroused great interest in the academic community, with several exact and heuristic approaches. Felipe et al. (2009) proposed a VNS-based heuristic approach, in which six neighborhood structures were used. Computational results showed that the VNS approach overcame the results presented by Petersen and Madsen (2009). Petersen et al. (2010) proposed different exact mathematical formulations for the DTSPMS, including a branch-and-cut algorithm to solve the DTSPMS instances to optimality. Based on these results, it was determined that the difficulty of a given instance depends not only on the number of items, but also heavily depends on the height of the stacks. Lusby et al. (2010) presented an exact method based on matching k-best tours for each of the regions separately. This method consists of repeatedly finding solutions for the two separate TSP (delivery region and pickup region) until a feasible loading plan is found. The results showed significant superiority of this method as compared to the previous one proposed on Petersen et al. (2010). Carrabs et al. (2010) developed a branch-and-bound algorithm for the Double Traveling Salesman Problem with Two Stacks (DTSP2S), a special case of the DTSPMS in which the vehicle has exactly two stacks. The results showed that the branch-and-bound algorithm performed better than the other exact approaches in the literature (Lusby et al., 2010; Petersen et al., 2010) in terms of computational time and number of global optima. According to Carrabs et al. (2010), for the exact approaches proposed by Lusby et al. (2010) and Petersen et al. (2010), the difficulty of an instance

depends on the capacity of the stacks, consequently, it depends on the number of items and on the number of stacks in the container. In this case, the performance of the algorithms improves when the number of stacks increases. This is probably due to the fact that the construction of the routes in the pickup region and in the delivery region becomes less restricted. Casazza et al. (2012) studied the theoretical properties of the DTSPMS, analyzing the structure of DTSPMS solutions in two separate components: routes and loading plan. It has been shown that some DTSPMS components can be solved in polynomial time, considering specific cases of the problem. Alba Martínez et al. (2013) proposed improvements to the branch-and-cut algorithm of Petersen et al. (2010), adding new valid inequalities (cut planes) that allowed greater efficiency. More recently, the exact algorithm proposed by Barbato et al. (2016) was able to solve instances involving containers of two stacks, which have not been previously solved in the literature. Hvattum et al. (2020) investigated the effect on transportation costs if an open side container could be used when transporting the pallets. Their experiments have shown that the total transportation cost is highly dependent on how the container is loaded, from the rear or from the side.

The Double Vehicle Routing Problem with Multiple Stacks (DVRPMS), a generalization of the DTSPMS, was proposed by Iori and Riera-Ledesma (2015). The DVRPMS has the same characteristics and constraints as the DTSPMS, except that now there is a fleet of vehicles available to meet the demand of the customers. According to the authors, the DVRPMS was motivated by the fact that not always a single vehicle is enough to transport all items. In addition, using multiple vehicles can be interesting even if all products could be transported by a single vehicle, as the addition of more vehicles causes an increase in the flexibility of the loading/unloading process and this can lead to a reduction in the operating costs of transport.

Figures 2.2 and 2.3 show two feasible solutions for the same instance involving 16 requests. Figure 2.2 represents a solution using three heterogeneous vehicles with containers of size (2×4), (1×4) and (2×2), where all containers are completely filled. Figure 2.3 represents a solution using four vehicles with containers of (2×4), (2×4), (2×2) and (2×2), but not all containers spaces are filled. In both examples, for each vehicle are associated a route in the pickup region, a route in the delivery region and a loading plan of the container. The pickup and delivery route for each vehicle respect the LIFO policy on the respective container.

Besides proposing the DVRPMS, Iori and Riera-Ledesma (2015) presented three exact algorithms: branch-and-cut, branch-and-price and branch-and-cut-and-price. As

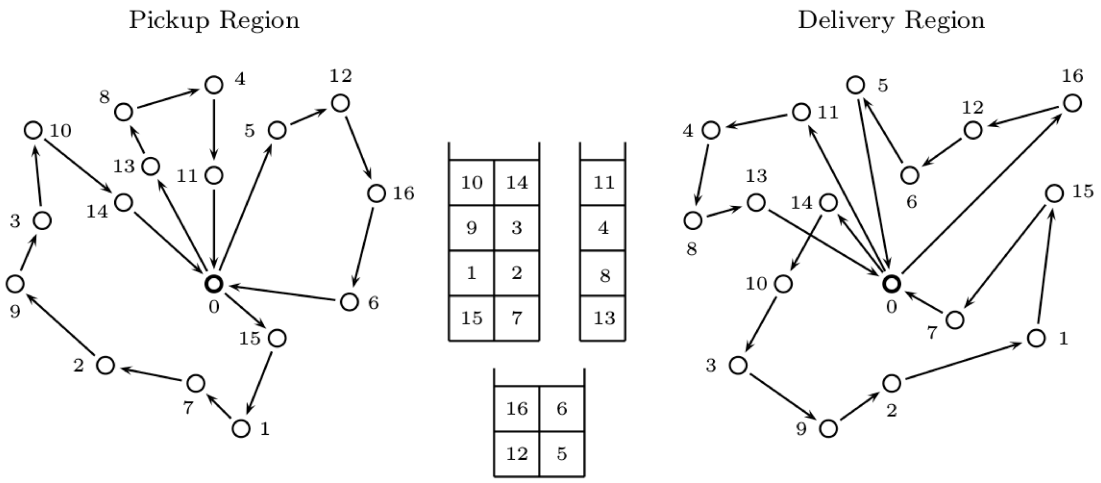


Figure 2.2: DVRPMS example (completely filled containers) - Adapted from (Iori and Riera-Ledesma, 2015).

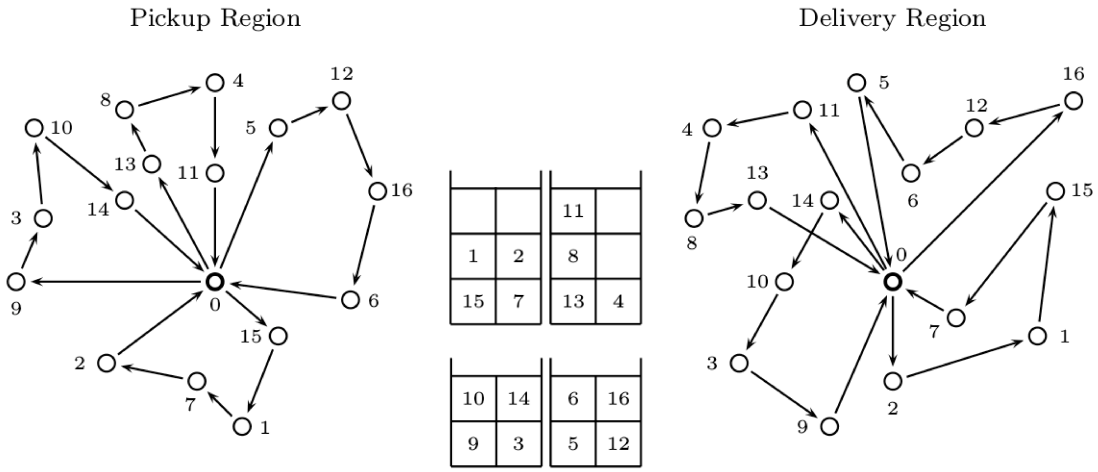


Figure 2.3: DVRPMS example (non-completely filled containers) - Adapted from (Iori and Riera-Ledesma, 2015).

stated by the authors, the three exact algorithms had different behavior and efficiency with respect to different instances, created to evaluate the quality of their algorithms.

To the best of our knowledge, only two works have addressed the DVRPMS by heuristic algorithms. Silveira et al. (2015) proposed three methods based on Iterated Local Search (ILS), Simulated Annealing (SA) and Variable Neighborhood Descent (VND) metaheuristics, while Chagas et al. (2016) proposed another method based on the SA metaheuristic, which outperformed all heuristic algorithms proposed by Silveira et al. (2015) and was able to find several solutions with the same quality as

those found by Iori and Riera-Ledesma (2015). Chagas and Santos (2016) introduced the Double Vehicle Routing Problem with Multiple Stacks and Heterogeneous Demand (DVRPMSHD), a generalization of the DVRPMS which occurs when customers have heterogeneous demands and the demand of each customer cannot be divided among two or more vehicles. The authors have also proposed a simple branch-and-price algorithm that was able to solve only instances of up to 15 customer requests. A simple and effective heuristic based on the Simulated Annealing was proposed by the same authors in Chagas and Santos (2017). Their algorithm has been able to overcome several results found by the branch-and-price algorithm. Posteriorly, Souza et al. (2018) proposed a heuristic algorithm based on the Late Acceptance Hill-Climbing metaheuristic that overcame some results found by the previous works.

Here, we address the DVRPMS, propose an Integer Linear Programming (ILP) formulation to solve it and a heuristic algorithm for obtaining high quality solutions on large instances.

The remainder of this chapter is organized as follows. In Section 2.1, we formally describe the DVRPMS and presents our ILP formulation. In Section 2.2, the details of the proposed heuristic algorithm are described. The computational experiments are reported in Section 2.3, in which we make a comparative analysis between our methods here proposed and the ones already described in the literature. Finally, in Section 2.4 we present our conclusions and emphasize the contributions of this chapter.

2.1 Problem description and mathematical formulation

As introduced by Iori and Riera-Ledesma (2015), the DVRPMS can be formally described as follows. Let $I = \{1, 2, \ldots, n\}$ be the set of customer requests carried by the vehicles in the pickup and delivery regions. Let also $V_c^P = \{1^P, 2^P, \ldots, n^P\}$ be the set of customers related to the pickup regions and $V_c^D = \{1^D, 2^D, \ldots, n^D\}$ the corresponding customers in the delivery region. Following the region dependencies, each request $i \in I$ corresponds to its $i^T \in V_c^T$ vertex, where T refers to any of the two regions.

It is possible to represent the DVRPMS as a directed graph $G = (V, A)$, where V is the set of vertices given by $V = V^P \cup V^D$, where $V^P = \{0^P\} \cup V_c^P$ and $V^D = \{0^D\} \cup V_c^D$. The members 0^P and 0^D are the depots for the pickup and delivery regions and members V_c^P and V_c^D are, respectively, the sets of vertices excluding the depots for the

pickup and delivery regions. Likewise, the set of arcs is given by $A = A^P \cup A^D$, where $A^P = \{(i^P, j^P) \in V^P \times V^P \mid i^P \neq j^P\}$ and $A^D = \{(i^D, j^D) \in V^D \times V^D \mid i^D \neq j^D\}$. For each arc $(i, j) \in A^P$ and $(i, j) \in A^D$ there is an associated routing cost of c_{ij}^P and c_{ij}^D, respectively.

Let K be the set of vehicles available to meet the transportation requirements. The loading compartment (container) of each vehicle $k \in K$ is divided into R_k stacks, all with the same height L_k. All vehicles in the K set must start and end their routes in the depots defined in each region. The vehicles must collect all items from customers located in the pickup region, store those items in their container, and then deliver the items to the respective customers located in the delivery region.

The routes of each vehicle must satisfy the LIFO policy in all its stacks, that is, if a client located at the vertex i^P (pickup region) is visited before the client located at vertex j^P (pickup region), and the requested item j is stored in the same stack in which the i item was stored, then the request client j must be visited (vertex j^D) before the client of the requisition i (vertex i^D) in the delivery region.

The objective of the DVRPMS is to serve all $|I|$ requests, so that the total distance traveled by the $|K|$ vehicles is the smallest possible one.

The DVRPMS can be formally modeled as a binary integer linear programming (ILP) problem. In the rest of this section, we describe an ILP formulation which was based on the mathematical formulation described by Chagas and Santos (2016) for the DVRPMSHD which in turn was based on the mathematical formulation of the DTSPMS proposed by Petersen and Madsen (2009) and on the mathematical formulation for the DVRPMS proposed by Iori and Riera-Ledesma (2015). For convenience of notation, we also refer to sets V_c^P and V_c^D as the set of requests I, i to denote both i^P and i^D, and also (i, j) to denote both (i^P, j^P) and (i^D, j^D). The variables used in the ILP formulation are described below:

- x_{ij}^{kT} : binary variable that gets 1 if the vehicle k crosses the arc (i, j) in the region T, and 0 otherwise.
- y_{ij}^{T} : binary variable that gets 1 if the vertex i is visited before the vertex j in region T, and 0 otherwise.
- w_i^k : binary variable that gets 1 if the vehicle k carries out the requested item i, and 0 otherwise.

- z_{ir}^{k} : binary variable that gets 1 if the item referring to the request i is stored in the r-th stack of the vehicle k, and 0 otherwise.

With these variables, we can describe the following integer linear programming formulation for the DVRPMS:

$$\min \sum_{k \in K} \sum_{T \in \{P,D\}} \sum_{(i,j) \in A^T} c_{ij}^{T} \cdot x_{ij}^{kT} \tag{2.1}$$

$$\sum_{j \in V^T} x_{0j}^{kT} = 1 \qquad k \in K, T \in \{P,D\} \tag{2.2}$$

$$\sum_{i \in V^T} x_{i0}^{kT} = 1 \qquad k \in K, T \in \{P,D\} \tag{2.3}$$

$$\sum_{j \in V^T \setminus \{i\}} x_{ji}^{kT} = w_i^k \qquad k \in K, T \in \{P,D\}, i \in V_c^T \tag{2.4}$$

$$\sum_{j \in V^T \setminus \{i\}} x_{ij}^{kT} = \sum_{j \in V^T \setminus \{i\}} x_{ji}^{kT} \qquad k \in K, T \in \{P,D\}, i \in V_c^T \tag{2.5}$$

$$\sum_{k \in K} w_i^k = 1 \qquad i \in I \tag{2.6}$$

$$\sum_{i \in I} z_{ir}^k \leq L_k \qquad k \in K, r = 1, \ldots, R_k \tag{2.7}$$

$$\sum_{r=1}^{R_k} z_{ir}^k = w_i^k \qquad k \in K, i \in I \tag{2.8}$$

$$y_{ij}^{T} + y_{ji}^{T} = 1 \qquad T \in \{P,D\}, i \in V_c^T, j \in V_c^T \setminus \{i\} \tag{2.9}$$

$$y_{i\ell}^{T} + y_{\ell j}^{T} \leq y_{ij}^{T} + 1 \qquad T \in \{P,D\}, \ell \in V_c^T, i \in V_c^T \setminus \{\ell\}, j \in V_c^T \setminus \{i,\ell\} \tag{2.10}$$

$$x_{ij}^{kT} \leq y_{ij}^{T} \qquad k \in K, T \in \{P,D\}, i \in V^T, j \in V^T \setminus \{i\} \tag{2.11}$$

$$y_{ij}^{P} + z_{ir}^k + z_{jr}^k \leq 3 - y_{ij}^{D} \qquad k \in K, i \in I, j \in I \setminus \{i\}, r = 1, \ldots, R_k \tag{2.12}$$

$$\sum_{j \in I} j \cdot x_{0j}^{kP} \leq \sum_{j \in I} j \cdot x_{0j}^{k'P} \qquad k \in K, k' \in K \mid k < k', R_k = R_{k'}, L_k = L_{k'} \tag{2.13}$$

$$x_{ij}^{kT} \in \{0,1\} \qquad k \in K, T \in \{P,D\}, (i,j) \in A^T \cup \{(0^T,0^T)\} \tag{2.14}$$

$$y_{ij}^{T} \in \{0,1\} \qquad T \in \{P,D\}, i \in V^T, j \in V^T \setminus \{i\} \tag{2.15}$$

$$z_{ir}^{k} \in \{0,1\} \qquad k \in K, i \in I, r = 1,\ldots,R_k \tag{2.16}$$

$$w_{i}^{k} \in \{0,1\} \qquad k \in K, i \in I \tag{2.17}$$

The objective function of the problem is defined by the Equation (2.1), which minimizes the total distance traveled by the vehicles. Constraints (2.2) and (2.3) ensure that each vehicle starts and ends its route in the depot of each region. Note that for instances where the total capacity of the vehicles is greater than the total customer demand, a vehicle k may not be used, in this case, we use a additional variable $x_{00}^{kT}\ \forall\, T \in \{P,D\}$ that gets 1 if the vehicle did not serve any customer request. Constraints (2.4) ensure that each request i is served by a vehicle k only if the vehicle k reaches the vertex i. Constraints (2.5) guarantee that the same vehicle must arrive and leave a vertex that represents the location of a client. Constraints (2.6) ensure that each request is served by one and only one vehicle. Constraints (2.7) ensure that the capacity of the stacks is not extrapolated. Constraints (2.8) ensure that the item of a request served by a vehicle k must be stored in its container. Constraints (2.9) and (2.10) establish a visit order between all vertex pairs in both regions and ensure a transitivity in this order, respectively. In other words, if i precedes l and l precedes j, then i precedes j. Constraints (2.11) ensure that if an arc (i,j) is traversed by a vehicle, then i is strictly visited before j. Constraints (2.12) indicate the restrictions regarding the LIFO policy applied in all stacks of all vehicles. If the item related to the requests i and j are stored in the same stack r of vehicle k, being i visited before j in the pickup region, then i cannot be visited before j in the delivery region. Constraints (2.13) break the resulting symmetry from the formulation, imposing a lexicographic order on the routes of vehicles with the same container configuration. And, finally, constraints (2.14) to (2.17) define the scope and domain of the decision variables.

Although this formulation does not use advanced optimization methods (e.g. branch-cut, branch-and-price, among others), it may be used as an alternative to the exact methods proposed by Iori and Riera-Ledesma (2015).

2.2 Heuristic approach

Since the DVRPMS is a complex combinatorial problem, for some large-size instance none of the exact methods proposed by Iori and Riera-Ledesma (2015) and neither our ILP formulation could solve the problem within an acceptable time. To work around this difficulty, implementing heuristic algorithms to treat large scale instances is inevitable. Considering that the VNS metaheuristic is widely used in the literature to obtain high quality solutions in combinatorial problems (Hansen and Mladenović, 2001), we also have been motivated to propose a method based on VNS to address the DVRPMS. The remainder of this section describes it in detail.

2.2.1 Solution representation

The solution representation defined by Chagas et al. (2016) is used in this work, where a solution for the DVRPMS is represented by the items of the $|I|$ customer requests which are distributed and allocated in the containers of the $|K|$ vehicles in the fleet. This representation defines the loading plan of the items in each container and, consequently, defines the loading/unloading constraints that must be obeyed in order to respect the LIFO policy.

Figure 2.4 shows an example of representation for a solution involving 16 transport requests and three vehicles with containers of dimensions (2×4), (1×4) and (2×2). As seen in the figure, the representation of the solution only informs the designation of the requests and the container loading plan for each vehicle. For each vehicle, the routes in the pickup and delivery regions are obtained by the evaluation functions described in Section 2.2.2 in the following.

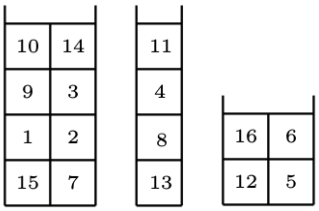


Figure 2.4: DVRPMS solution representation example (Chagas et al., 2016).

2.2.2 Evaluation function

As previously mentioned, the solution representation does not report the routes performed by each vehicle. We assign this task to the evaluation function, which is responsible to evaluate and determine a route in the pickup region and another in the delivery region for each vehicle from its container (solution representation) and thus to return the total distance traveled by the vehicles.

Finding the shortest route in the pickup region (or the delivery region) with LIFO constraints imposed by a known loading plan can be seen as the Traveling Salesman Problem with Precedence Constraints (TSPPC), proposed by Savelsbergh and Sol (1995), where the precedence constraints are given by the loading plan. According to Moon et al. (2002), TSPPC belongs to the $\mathcal{NP}$-Hard class of problems, therefore the optimal solution to the problem cannot be obtained within a reasonable computational time when large-size instances are considered, unless $\mathcal{P} = \mathcal{NP}$.

In this work, we have developed two different evaluation functions. At first, we consider the evaluation function, denoted as $f_{opt}(.)$, that consists in applying the dynamic programming algorithm described by Casazza et al. (2012) in each container $k \in K$ to determine the shortest pickup (delivery) route that satisfies the LIFO constraints imposed by the loading plan.

From preliminary tests, we have concluded that evaluating all solutions explored by the VNS algorithm using the evaluation function $f_{opt}(.)$ is impracticable due to its exponential complexity, that is, though the number of stacks is small, during the execution of the algorithm a countless number of solutions are evaluated. Therefore, we have proposed a simple heuristic evaluation, denoted as $f_{heur}(.)$, in order to find good quality routes in shorter computational time. This heuristic evaluation can be divided into two phases that are subsequently performed: a greedy phase and a local search phase.

In the greedy phase, for each vehicle, a pickup route (respectively a delivery route) is constructed choosing at each time, among the items at the bottom (top) of each stack, the item that has the least impact (least distance) on the pickup (delivery) route. In other words, the pickup route delivery (route) is constructed choosing at each moment the item collected (delivered) that is the closest one to the partially constructed route.

Figure 2.5 illustrates the greedy phase of the evaluation heuristic when applied to the pickup region (a) and other in the delivery region (b), considering a vehicle with a

container of dimensions (2×4). The dashed positions in the container indicate that the items already have been inserted in the routes by previous iterations. In the case shown in Figure 2.5 (a), the last vertex inserted in the pickup route is vertex 9^P, which refers to the request 9. From this vertex, only vertices 3^P and 10^P, associated with requests 3 and 10, respectively, are accessible in the pickup region, since the constraints of the loading plan must be satisfied. Between vertices 3^P and 10^P, vertex 3^P will be chosen, since vertex 3^P is the closest to the vertex 9^P. Similarly, in Figure 2.5 (b) the vertex 2^D will be chosen, since, among the candidates 1^D and 2^D, 2^D is the closest to the last vertex (9^D) inserted in the delivery route.

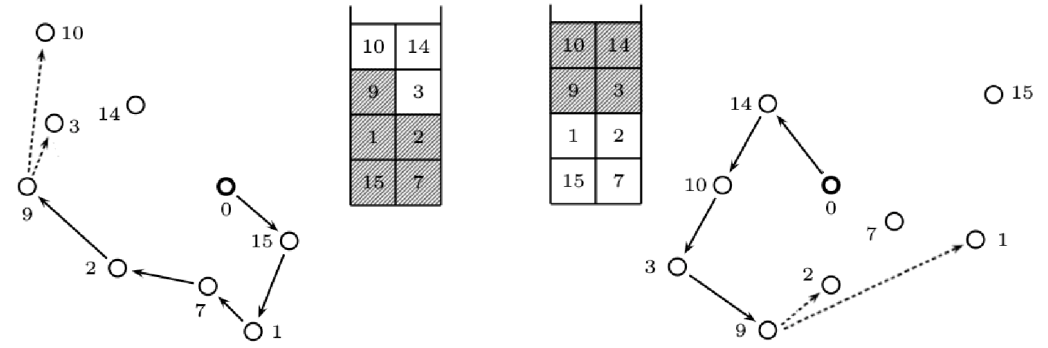
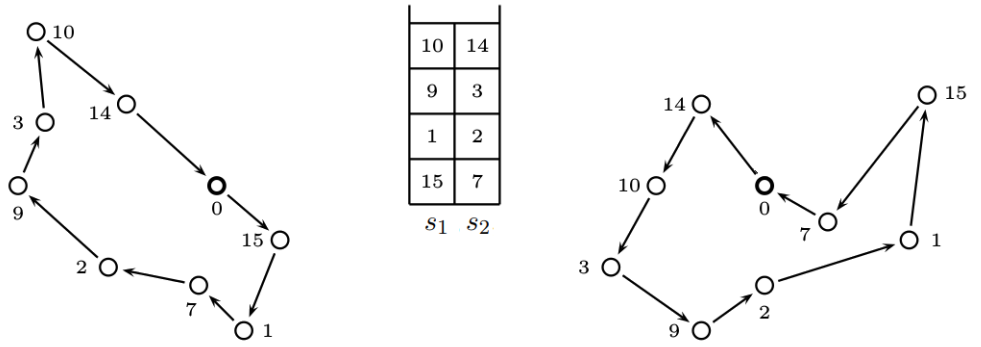


Figure 2.5: Greedy phase of the evaluation heuristic.

Note that the routes assigned to a vehicle can be represented by the order that the operations are performed on the container stacks. Figure 2.6 illustrates this statement, where the stacks of a container of size (2×4) are named s_1 and s_2 to facilitate the appropriate referencing of each stack. The sequences $\mathcal{S}^P$ and $\mathcal{S}^D$ represent, respectively, the route assigned to the vehicle in the pickup and delivery regions.

The second phase of the evaluation heuristic consists of applying a refinement algorithm to the routes determined by the greedy phase. In this phase, a local search is applied separately in each one of the sequences $\mathcal{S}^P$ and $\mathcal{S}^D$ in order to find better routes. The neighborhood structure defined to perform this local search consists of exchanging two operations s_i and $s_j \mid i \neq j$ of their positions. Figure 2.7 shows a neighbor $\mathcal{S}^{P'}$ from $\mathcal{S}^P$, as well as the changes caused in the pickup route.

In the local search procedure, the inspection of neighbors is done casually (a neighbor is chosen randomly from the neighbors of a neighborhood structure) and the neighborhood of the current sequence is explored until a sequence that represents a



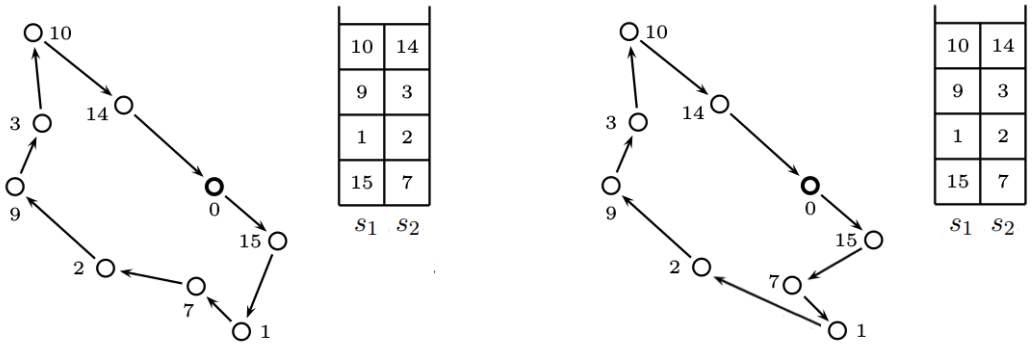
Pickup route

$$\mathcal{S}^P = \{s_1, s_1, s_2, s_2, s_1, s_2, s_1, s_2\}$$

Delivery route

$$\mathcal{S}^D = \{s_2, s_1, s_2, s_1, s_2, s_1, s_1, s_2\}$$

Figure 2.6: Representation of the routes through the operations carried out in the stacks.



Pickup route

$$\mathcal{S}^P = \{s_1, s_1, s_2, s_2, s_1, s_2, s_1, s_2\}$$

Pickup route with swapped operations

$$\mathcal{S}^{P'} = \{s_1, \mathbf{s_2}, \mathbf{s_1}, s_2, s_1, s_2, s_1, s_2\}$$

Figure 2.7: Local search phase of the evaluation heuristic.

shorter route is found than the route already known, i.e., we use a first improvement strategy.

Given the solution representation and a defined strategy to evaluate these solutions, the next section defines and details the neighborhood structures to be applied to DVRPMS solutions.

2.2.3 Neighborhood structures

In order to explore the solution space of the DVRPMS, we define four neighborhood structures: Item Swap (IS), Item Ejection Chain (IC), Stack Permutation (SP) and Stack Swap (SS). All these structures are used in our VNS algorithm and are described and detailed as follows.

Item swap

The Item Swap (IS) neighborhood of a solution of the DVRPMS, defined by Chagas et al. (2016), contains the solutions that can be obtained by swapping two items of the containers. The items to be relocated may belong to the same container or to different container, then the size of IS neighborhood is $O\left(\left(\begin{array}{c}|I|\\2\end{array}\right)\right) = O(|I|^2)$. Figure 2.8 shows an example of a solution s and one of its neighbors s'.

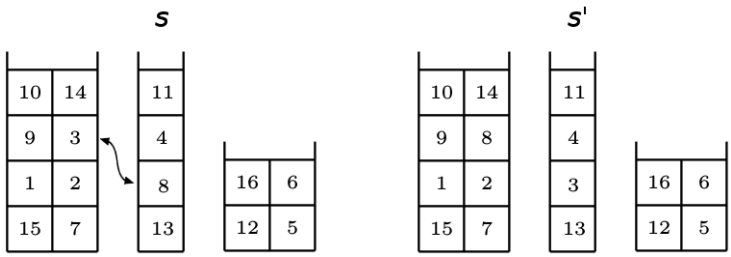


Figure 2.8: An Item Swap (IS) example (Chagas et al., 2016).

Item ejection chain

The Item Ejection Chain (IC) neighborhood of a solution of the DVRPMS contains the solutions that can be obtained by exchanging three items of the containers by way of ejection chain, i.e., the first item is relocated in position of second one, the second item is relocated in position of third one and the third item is relocated in position of the first one. As in the neighborhood structure IS, the items to be relocated may belong to the same container or to different container, so the size of IC neighborhood is $O(|I|^3)$. Figure 2.9 shows a solution s and one of its neighbors s'.

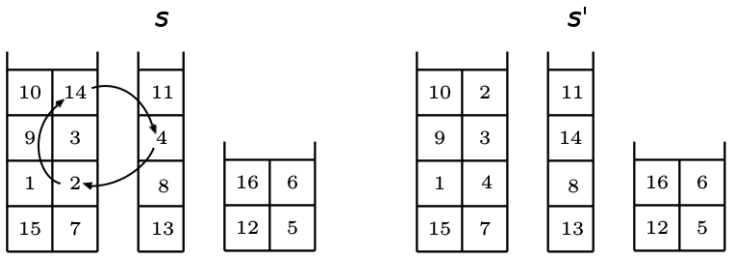


Figure 2.9: An Item Ejection Chain (IC) example.

Stack permutation

The Stack Permutation (SP) neighborhood of a solution of the DVRPMS contains the solutions that can be obtained by rearranging items from the same row (horizontal stack) using a permutation of these items. As the total number of stacks of a solution is $\sum_{k \in K} R_k$ and each stack of size L_k has $L_k!$ permutations, the size of SP neighborhood is $O\left(\sum_{k \in K} L_k!\ R_k\right)$. Figure 2.10 shows an example of a solution s and one of its neighbors s'.

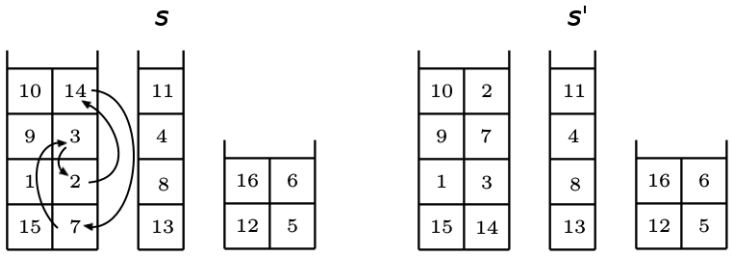


Figure 2.10: A Stack Permutation (SP) example.

Stacks swap

The Stack Swap (SS) neighborhood of a solution of the DVRPMS contains the solutions that can be obtained by swapping two stacks of the different containers. As the stacks to be relocated must belong to different containers, the size of SS neighborhood is given by combination of the total number of stacks grouped in pairs, disregarding the pairs of stacks which belong to the same vehicle, that is, $O\left(\binom{\sum_{k \in K} R_k}{2} - \sum_{k \in K} \binom{R_k}{2}\right)$. Figure 2.11 shows an example of a solution s and one of its neighbors s'.

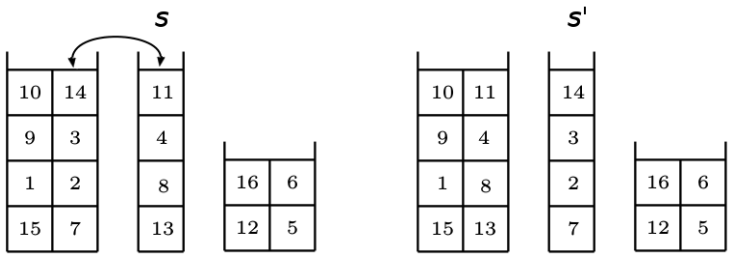


Figure 2.11: A Stack Swap (SS) example.

For two stacks with different heights, the swap is made only between the items of the smallest stack and the items loaded in the first (bottom-up) positions of the largest stack. Figure 2.12 shows an example for this situation.

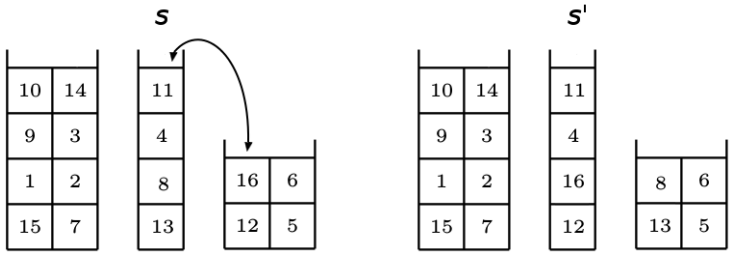


Figure 2.12: A Stack Swap (SS) example, when the heights of the stacks are different.

2.2.4 Initial solution

The initial solution of our VNS algorithm is created randomly. A random subset of customer requests is assigned to each vehicle in such way that the number of requests does not exceed the capacity of each vehicle and that each request is to be served by a single vehicle.

2.2.5 A variable neighborhood search approach

The Variable Neighborhood Search (VNS) metaheuristic, proposed by Mladenović and Hansen (1997) (see e.g. Hansen and Mladenović (2014) for a recent description), is a higher-level procedure widely used to treat a large variety of practical and complex problems Felipe et al. (2009); Tricoire et al. (2011); Wei et al. (2014, 2015); Pinto et al. (2018). In its simplest form, known as basic VNS (Hansen and Mladenović, 2001), the VNS requires a local search procedure and a set of neighborhood structures $\mathcal{N}_k$

($k = 1, 2, \ldots, k_{\max}$) which are used to perform shaking moves in order to avoid getting stuck on local optima solutions found throughout the algorithm.

Algorithm 1 describes our proposed VNS, where $f_{heur}(.)$ and $f_{opt}(.)$ are the functions that run the evaluation heuristics described in Section 2.2.2 for each container and return the total distance traveled by the vehicles. Initially, we define the set $\mathcal{N}$ which consists of the four neighborhood structures previously described. These neighborhood structures are arranged hierarchically according to their perturbation strength, i.e., the first neighborhood structure is the `IS`, followed by the `IC`, `SP` and `SS` ones. The algorithm's initial solution (line 3) is generated randomly according to Section 2.2.4. While the number of iterations without improvement has not reached the established maximum ($iter\_max$), the algorithm chooses a random neighbor s' using the k-th neighborhood structure (initially $k = 1$) from the current solution s and then applies a local search in s', that produces a new solution s''. The local search (see Algorithm 2) consists of applying a descent method using the `IS` neighborhood structure with a first improvement strategy. If the solution s'' is better than s, s'' becomes the current solution and the procedure is repeated using the first neighborhood structure (k is reset to 1), otherwise the procedure is repeated using the next neighborhood structure ($k + 1$). The method stops when the current number of iterations reaches the stopping criteria, noted by $iter\_max$. Then, the best solution found is evaluated by the function $f_{opt}(.)$ and then returned. Note that during the internal part of the algorithm (line 5 to 16) all the solutions are evaluated by function $f_{heur}(.)$. This means that throughout the algorithm, each solution is evaluated heuristically, favouring efficiency, while the last one is evaluated optimally, favouring quality, so that the functioning of the algorithm does not become costly (time consuming).

2.3 Computational experiments

The VNS algorithm has been implemented in C/C++ and has been sequentially (nonparallel) performed on a computer with the same settings as those used in the experiments reported by Chagas et al. (2016), that is, on an Intel Core i5-3570 @ 3.40 GHz x 4 computer with 16GB of RAM running the operating system Ubuntu 14.04 LTS 64 bits. Since for some instances, the ILP formulation requires a large amount of memory, we have run it on an Intel(R) Xeon(R) CPU E5-2660 v2 @ 2.20GHz x 40 computer with 384GB of RAM, running the operating system CentOS Release 6.8.

Algorithm 1: Variable Neighborhood Search (VNS)

```
1  Let 𝒩 = {𝒩_1, 𝒩_2, 𝒩_3, 𝒩_4} = {IS, IC, SP, SS} be the set of neighborhood
     structures
2  k_max ← | 𝒩 |
3  s ← generate a random solution
4  iter ← 0
5  while iter < iter_max do
6  |  k ← 1
7  |  repeat
8  |  |  s' ← pick a random neighbor in the k-th neighborhood structure of s
9  |  |  s'' ← apply local search in s' using the IS neighborhood structure // Alg 2
   |  |   if f_heur(s'') < f_heur(s) then
10 |  |  |  s ← s''
11 |  |  |  k ← 1
12 |  |  |  iter ← 0
13 |  |  else
14 |  |  |  k ← k + 1
15 |  |  |  iter ← iter + 1
16 |  until k > k_max
17 f_opt(s)
18 return s
```

Algorithm 2: Local Search (LS)

```
1  Let s be the solution in which the local search will be applied
2  Let IS(s) be the set of solutions in the Item Swap (IS) neighborhood of
     solution s
3  improv ← true
4  while improv = true do
5  |  improv ← false
6  |  neighbors ← IS(s)
7  |  while neighbors ≠ { } and improv = false do
8  |  |  s' ← pick a random neighbor s' ∈ neighbors
9  |  |  if f_heur(s') < f_heur(s) then
10 |  |  |  s ← s'
11 |  |  |  improv ← true
12 |  |  else
13 |  |  |  neighbors ← neighbors \ {s'}
14 return s
```

Our ILP formulation has been implemented in C/C++ using the Concert Technology Library of CPLEX 12.5 under an academic license, with all CPLEX default settings,

except for the runtime that has been limited to 3 hours. It is worth mentioning that unlike the VNS algorithm, the ILP formulation has been executed on multiple threads, as defined by the CPLEX default settings. Our code is publicly available at https://github.com/jonatasbcchagas/ilp_vns_dvrpms.

2.3.1 Benchmarking instances

In order to assess the quality of our solution approaches, we have used the set of benchmark instances described by Iori and Riera-Ledesma (2015). These authors have defined 24 different types of instances, where each type contains the number of customer requests, the number of vehicles available and the configurations of the vehicle's loading compartments. These 24 types of instances are divided into two sets. The first one, denoted by C, contains 15 types of instances for which the total capacity of the vehicles is equal to the total number of customer requests (number of items), that is, for all these types of instances the containers must be fully loaded in order to serve all customers. The second set, denoted by $\neg C$, contains 9 types of instances for which the total number of customer requests (number of items) is less than the total capacity of the vehicles, so the containers do not need to be completely filled in order to serve customers. Tables 2.1 and 2.2 show, respectively, the specifications of sets C and $\neg C$. Each type is described by the number of customer requests $|I|$, the number of vehicles $|K|$ and the configuration of the containers of the vehicle fleet (column $(R \times L)'s$). The last column of each table shows the total capacity of each type of fleet.

Regarding customer locations, for each type of instances previously defined, Iori and Riera-Ledesma (2015) used the data from 5 DTSPMS benchmark instances (R05, R06, R07, R08, and R09), giving in total 120 instances. Each of these instances reports the customers' locations in the pickup region and in the delivery region. These locations were chosen at random within two different regions of dimensions 100×100, and the depot of each region was fixed in coordinates $(50, 50)$. The distance between any two points of the same region was calculated using the rounded Euclidean distance.

Table 2.1: Instance specifications of group C.

T	$\|I\|$	$\|K\|$	$(R \times L)'s$	$\sum_{k \in K} R_k L_k$
(a)	12	2	(2×3) (2×3)	12
(b)	12	2	(2×2) (2×4)	12
(c)	12	3	(2×2) (2×2) (2×2)	12
(d)	16	2	(2×4) (2×4)	16
(e)	16	3	(2×2) (2×3) (2×3)	16
(f)	16	4	(2×2) (2×2) (2×2) (2×2)	16
(g)	18	2	(2×3) (3×4)	18
(h)	18	3	(2×3) (2×3) (2×3)	18
(i)	18	4	(2×2) (2×2) (2×2) (2×3)	18
(j)	20	2	(2×4) (3×4)	20
(k)	20	3	(2×3) (2×3) (2×4)	20
(l)	20	4	(2×3) (2×3) (2×2) (2×2)	20
(m)	24	2	(3×4) (3×4)	24
(n)	24	3	(2×4) (2×4) (2×4)	24
(o)	24	4	(2×3) (2×3) (2×3) (2×3)	24

Table 2.2: Instance specifications of group $\neg C$.

T	$\|I\|$	$\|K\|$	$(R \times L)'s$	$\sum_{k \in K} R_k L_k$
(p)	18	2	(4×4) (4×4)	32
(q)	18	3	(3×4) (3×4) (3×4)	36
(r)	18	4	(2×4) (2×4) (2×4) (2×4)	32
(s)	20	2	(4×4) (4×4)	32
(t)	20	3	(3×4) (3×4) (3×4)	36
(u)	20	4	(2×4) (2×4) (2×4) (2×4)	32
(v)	24	2	(4×4) (4×4)	32
(w)	24	3	(3×4) (3×4) (3×4)	36
(x)	24	4	(2×4) (2×4) (2×4) (2×4)	32

2.3.2 Parameter tuning

The proposed VNS algorithm has only one parameter (number of iterations without improvement) that is referenced as *iter_max*. It is responsible for stopping the execution of the algorithm (stopping criteria). Naturally, the higher the *iter_max* value, the wider the search and, consequently, the algorithm finds better solutions but spends more time. In order to balance the quality of the solutions and the execution time, for each instance, we have run 30 times the VNS algorithm with different values of *iter_max* and constructed the chart shown in Figure 2.13. The values tested for *iter_max* are arranged on the horizontal axis of the chart, such values have been defined in function of the number of customer requests $|I|$. For each *iter_max* value, we plot the average value of the objective function and also the average execution time.

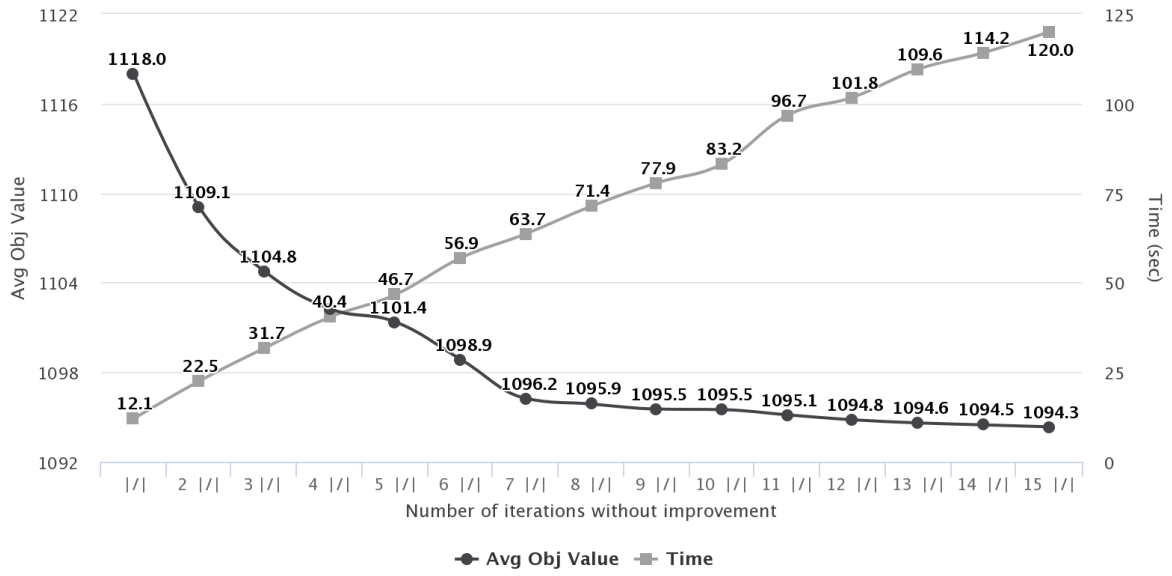


Figure 2.13: Analysis of the stopping criteria of the VNS algorithm.

After analyzing the chart, it is inferred that from $item\_max = 7 \mid I \mid$ the improvement is small in relation to the processing time. Therefore, the final experiments were performed with $item\_max = 7 \mid I \mid$.

2.3.3 Results on test instances

The results obtained by our approaches to solve the DVPRMS is reported in this section. As pointed before, the execution time of the ILP formulation was limited to 3 hours. All exact algorithms proposed by Iori and Riera-Ledesma (2015) were limited to 1 hour. Our VNS algorithm was ran 10 independent times and the average and best value of the objective function obtained in these 10 runs were used in our analysis.

Tables 2.3 and 2.4 report the results on instances of group C, that is, those in which the total capacity of the fleet of vehicles is exactly the same as the number of customer requests. Thus, the containers must be completely filled so that all customers are served. Table 2.3 shows the results for the 30 instances (ID 001 to ID 030), considered by Iori and Riera-Ledesma (2015) as small instances and Table 2.4 shows the results for the 45 large-size instances (ID 031 to ID 075). Table 2.5 reports the results for the 45 instances of group $\neg C$ (ID 076 to ID 120), that is, those in which the total fleet capacity of vehicles is larger than the number of customers and, therefore, the containers will not necessarily be completely filled.

The first three columns of each table describe the instances, being that each instance is identified by a number (column *ID*), a name indicating the pickup and delivery regions (column *R*), and a type of instance (column *T*). The best results obtained by Iori and Riera-Ledesma (2015) are presented in the columns *UB* and *t(s)* that indicate, respectively, the upper-bound and the execution time in seconds. The results obtained by our ILP formulation are described in columns *LB*, *UB*, *Gap %*, *t(s)* and *Opt*. Columns *LB* and *UB*, respectively, indicate the lower-bound and the upper-bound obtained at the end of the execution. Column *Gap %* shows the relative gap between *UB* and *LB* which can be calculated as $100 \times (UB - LB)/UB$. Column *t(s)* informs the total processing time and column *Opt* indicates by an asterisk the instances solved to optimality. The next columns show the best results and the processing time (columns *Best* and *t(s)*) obtained by Silveira et al. (2015). The results obtained by Chagas et al. (2016) and the proposed VNS algorithm are presented in three columns, *Avg*, *Best* and *t(s)* that inform, respectively, the average solution value, the best solution value, and the total processing time consumed by the 10 runs. Note that Silveira et al. (2015) did not consider instances of group $\neg C$ in their experiments, so Table 2.5 does not show these results. Although Chagas et al. (2016) do not show the results of instances of group $\neg C$, we executed their algorithm and reported the results in Table 2.5.

Tables 2.3 and 2.4 show the superiority of the results reported by Iori and Riera-Ledesma (2015) when compared to results reached by the ILP formulation, since for every instance of group *C* the algorithms developed by Iori and Riera-Ledesma (2015) runs faster than our ILP formulation and finds better solutions for most instances. However, although the ILP formulation is not as efficient as the exact methods of Iori and Riera-Ledesma (2015), as it was solved with no tentative of strengthening or specialized techniques, it is of fundamental importance for this work and for future ones that will address the DVRPMS, since through the results of our ILP formulation, it was possible to notice inconsistencies in the results published by Iori and Riera-Ledesma (2015).

All results proven to be inconsistent are highlighted in Table 2.3 and 2.4 by the character †, which is inserted in column UB that relates to the results of Iori and Riera-Ledesma (2015). Those results were considered inconsistencies due to the lower-bound (column LB) of the ILP formulation being larger than the upper-bound (column UB) reported by Iori and Riera-Ledesma (2015). After noticing these inconsistencies, the

Table 2.3: Comparative analysis on small-size instances of group *C*.

Instance			Iori and Riera-Ledesma (2015)		ILP formulation					Silveira et al. (2015)		Chagas et al. (2016)			VNS		
ID	R	T	UB	t(s)	UB	LB	Gap %	t(s)	Opt	Best	t(s)	Avg	Best	t(s)	Avg	Best	t(s)
001	R05	(*a*)	738	2	**738**	738.0	0.00	46	*	746	31	738.0	**738**	13	738.0	**738**	2
002	R06		895	0	**895**	895.0	0.00	18	*	897	30	895.0	**895**	13	895.0	**895**	2
003	R07		761	5	**761**	761.0	0.00	486	*	**761**	31	761.0	**761**	13	761.0	**761**	2
004	R08		848†	0	**851**	851.0	0.00	27	*	**851**	31	851.0	**851**	13	851.0	**851**	2
005	R09		771†	0	**776**	776.0	0.00	20	*	781	31	776.0	**776**	14	776.0	**776**	2
006	R05	(*b*)	716	1	**716**	716.0	0.00	23	*	728	31	716.9	**716**	3	716.0	**716**	1
007	R06		859†	2	**866**	866.0	0.00	18	*	873	31	866.0	**866**	3	877.4	**866**	1
008	R07		733	3	**733**	733.0	0.00	95	*	**733**	31	733.0	**733**	3	733.0	**733**	1
009	R08		858	3	**858**	858.0	0.00	31	*	875	31	858.0	**858**	3	865.5	**858**	1
010	R09		738†	1	**741**	741.0	0.00	16	*	760	31	741.5	**741**	3	752.8	**741**	1
011	R05	(*c*)	855	0	**855**	855.0	0.00	72	*	859	35	855.0	**855**	5	855.0	**855**	1
012	R06		1011	0	**1011**	1011.0	0.00	78	*	**1011**	35	1011.0	**1011**	5	1011.0	**1011**	1
013	R07		894	1	**894**	894.0	0.00	367	*	**894**	35	894.0	**894**	5	894.0	**894**	1
014	R08		990	1	**990**	990.0	0.00	146	*	**990**	35	990.0	**990**	5	990.0	**990**	1
015	R09		852	0	**852**	852.0	0.00	54	*	853	35	852.0	**852**	5	852.0	**852**	1
016	R05	(*d*)	947	329	**947**	919.0	2.96	3 h		976	32	948.1	948	1	950.2	**947**	5
017	R06		1036	43	**1036**	1036.0	0.00	1625	*	1093	32	1040.2	**1036**	1	1045.1	**1036**	6
018	R07		925	46	**925**	925.0	0.00	3303	*	971	32	925.0	**925**	1	926.6	**925**	4
019	R08		1006†	52	**1010**	1010.0	0.00	1632	*	1060	33	1021.5	1020	1	1014.6	**1010**	6
020	R09		907†	97	**921**	921.0	0.00	3195	*	3195	10	925.1	925	1	924.1	**921**	5
021	R05	(*e*)	1050	21	**1050**	1050.0	0.00	4225	*	1087	37	1050.0	**1050**	21	1051.6	**1050**	5
022	R06		1102†	6	**1114**	1114.0	0.00	1778	*	**1114**	37	1114.0	**1114**	20	1114.0	**1114**	4
023	R07		1063	29	1072	958.1	10.63	3 h		1089	37	1063.0	**1063**	20	1066.9	**1063**	4
024	R08		1117†	10	**1126**	1126.0	0.00	2785	*	1142	36	1130.8	**1126**	21	1136.3	**1126**	4
025	R09		1021	9	**1021**	1021.0	0.00	2187	*	1046	36	1021.6	**1021**	21	1026.1	**1021**	5
026	R05	(*f*)	1217	37	**1217**	1054.5	13.35	3 h		1223	42	1217.0	**1217**	10	1217.0	**1217**	2
027	R06		1290	10	**1290**	1155.5	10.43	3 h		**1290**	41	1290.0	**1290**	10	1290.0	**1290**	2
028	R07		1230	44	**1230**	1046.8	14.90	3 h		**1230**	41	1230.0	**1230**	10	1230.0	**1230**	2
029	R08		1261	18	**1261**	1148.3	8.94	3 h		1262	44	1261.0	**1261**	10	1261.0	**1261**	2
030	R09		1127	7	**1134**	996.9	12.09	3 h		**1134**	43	1134.0	**1134**	10	1134.0	**1134**	2
Average			960.6	25.9	963.0	934.0	2.44	3260.9	$\frac{23}{30}$	976.0	33.9	963.7	963.2	8.8	965.2	962.7	2.5

authors were notified and after a careful examination of the details of the solution, we noticed that for some instances the solutions reported by their algorithms were unfeasible (the routes did not respect the stacks' LIFO policy). The authors shared with us their code, but we could not find the reason of the inconsistencies.

Since the results of Iori and Riera-Ledesma (2015) have not been corrected yet, we will focus the comparative analysis of results of group *C* considering only the other methods of resolution. In addition, in order to highlight the solution quality of each method, best results reached for each instance are shown in bold.

For the small instances of group *C*, the ILP formulation was able to find the optimal solution for 23 out of 30 instances, whereas, for the 45 large instances, only 6 of them

Table 2.4: Comparative analysis on large-size instances of group *C*.

Instance			Iori and Riera-Ledesma (2015)		ILP formulation					Silveira et al. (2015)		Chagas et al. (2016)			VNS		
ID	R	T	UB	t(s)	UB	LB	Gap %	t(s)	Opt	Best	t(s)	Avg	Best	t(s)	Avg	Best	t(s)
031	R05	(*g*)	950	9	**950**	950.0	0.00	778	*	1083	31	952.5	**950**	19	964.8	**950**	16
032	R06		1012†	27	**1024**	1024.0	0.00	7224	*	1087	31	1038.8	**1024**	19	1035.6	**1024**	15
033	R07		932	67	**932**	932.0	0.00	8592	*	1054	32	936.8	**932**	21	943.4	**932**	13
034	R08		1011†	50	**1018**	1018.0	0.00	3635	*	1128	31	1040.7	1031	20	1037.8	1024	16
035	R09		909†	48	**919**	919.0	0.00	2860	*	985	31	937.4	**919**	21	931.5	**919**	18
036	R05	(*h*)	1147	60	**1147**	1021.9	10.91	3 h		1198	41	1147.0	**1147**	33	1153.9	**1147**	8
037	R06		1165	8	**1177**	1100.0	6.54	3 h		1190	42	1177.0	**1177**	33	1177.0	**1177**	10
038	R07		1123	32	1140	930.1	18.41	3 h		1136	41	1123.0	**1123**	33	1123.0	**1123**	7
039	R08		1184	40	**1184**	1096.7	7.37	3 h		1214	41	1184.0	**1184**	33	1188.6	**1184**	9
040	R09		1080	50	1097	954.0	13.03	3 h		1111	41	1081.4	**1080**	33	1080.6	**1080**	9
041	R05	(*i*)	1269	319	1291	999.4	22.59	3 h		1292	43	1272.0	**1269**	18	1274.6	**1269**	5
042	R06		1264	38	**1275**	1084.8	14.92	3 h		1282	43	1275.1	**1275**	18	1275.1	**1275**	6
043	R07		1261	193	1281	936.8	26.87	3 h		1284	43	1261.0	**1261**	18	1273.7	**1261**	4
044	R08		1310	504	1312	1109.3	15.45	3 h		1338	44	1310.0	**1310**	18	1317.1	**1310**	6
045	R09		1157	62	**1157**	954.0	17.54	3 h		1196	44	1158.0	**1157**	18	1162.8	**1157**	6
046	R05	(*j*)	1012	115	1013	953.0	5.92	3 h		1126	34	1019.2	**1012**	6	1024.3	**1012**	21
047	R06		1018	24	**1018**	1018.0	0.00	1453	*	1177	33	1043.5	**1018**	6	1050.3	**1018**	21
048	R07		1047	290	1056	941.6	10.83	3 h		1162	34	1065.8	1054	6	1075.5	**1047**	18
049	R08		1040	751	1058	981.7	7.22	3 h		1221	10	1068.8	**1050**	8	1073.4	**1050**	18
050	R09		959	55	**973**	930.4	4.38	3 h		1126	34	985.2	977	7	987.6	974	22
051	R05	(*k*)	1174	860	1231	981.0	20.31	3 h		1212	45	1180.4	**1174**	28	1180.4	**1174**	13
052	R06		1200	66	1234	1064.4	13.74	3 h		1253	44	1209.5	**1200**	27	1227.0	**1200**	11
053	R07		1243	1471	1375	942.6	31.45	3 h		1289	44	1247.4	**1243**	28	1258.0	**1243**	11
054	R08		1196	915	1266	1050.3	17.04	3 h		1275	44	1210.9	**1206**	28	1211.3	**1206**	14
055	R09		1133	268	1178	959.2	18.57	3 h		1196	46	1151.0	**1144**	27	1154.4	**1144**	12
056	R05	(*l*)	1293	1306	1298	961.1	25.96	3 h		1347	49	1294.6	**1293**	30	1293.0	**1293**	11
057	R06		1340	202	1341	1043.6	22.18	3 h		1374	49	1340.4	**1340**	29	1342.4	**1340**	9
058	R07		1373	2843	1405	914.5	34.91	3 h		1418	49	1373.0	**1373**	30	1380.0	**1373**	9
059	R08		1305	460	1345	1049.4	21.98	3 h		1345	49	1311.5	**1305**	29	1323.6	**1305**	9
060	R09		1252	832	1261	896.6	28.90	3 h		1310	49	1259.2	**1258**	30	1262.3	**1258**	9
061	R05	(*m*)	1060	2326	**1069**	924.0	13.56	3 h		1220	35	1089.1	1073	14	1092.4	1071	61
062	R06		1093	70	**1093**	976.9	10.62	3 h		1324	35	1111.1	**1093**	14	1120.1	**1093**	52
063	R07		1096	263	1213	901.4	25.69	3 h		1325	35	1108.8	1103	13	1111.8	**1098**	51
064	R08		1120	1149	1230	941.5	23.46	3 h		1300	34	1156.6	**1133**	18	1159.0	**1133**	67
065	R09		1034	735	1148	863.4	24.79	3 h		1266	34	1074.6	**1047**	20	1074.6	**1047**	60
066	R05	(*n*)	1318	1 h	1407	914.2	35.03	3 h		1387	47	1276.2	**1262**	3	1275.7	**1262**	20
067	R06		1289	1 h	1550	1016.3	34.44	3 h		1438	48	1315.6	**1304**	3	1308.6	**1304**	21
068	R07		1350	1 h	1501	889.7	40.73	3 h		1429	46	1345.5	**1333**	3	1347.3	1338	22
069	R08		1297	1 h	1426	975.9	31.56	3 h		1426	47	1320.5	**1297**	3	1324.2	**1297**	22
070	R09		1239	1 h	1577	817.0	48.19	3 h		1346	47	1264.5	1243	3	1242.5	**1240**	24
071	R05	(*o*)	1518	1 h	1552	912.8	41.19	3 h		1422	53	1373.5	**1367**	60	1368.6	**1367**	25
072	R06		1449	1 h	1749	992.3	43.27	3 h		1516	53	1448.2	**1448**	60	1454.4	**1448**	25
073	R07		1677	1 h	1716	900.8	47.50	3 h		1509	48	1469.2	**1465**	61	1473.2	**1465**	28
074	R08		1463	1 h	1555	975.2	37.29	3 h		1488	52	1448.8	**1444**	61	1446.0	**1444**	25
075	R09		1570	1 h	1488	871.1	41.46	3 h		1413	52	1363.5	**1362**	60	1362.0	**1362**	23
Average			1198.5	1166.8	1249.6	968.7	20.35	9905.4	$\frac{6}{45}$	1271.5	41.3	1196.0	1188.0	23.8	1198.7	1187.5	19.6

were proved to have found the optimal solution. The high relative gap values (column

Table 2.5: Comparative analysis on instances of group ¬C.

Instance			Iori and Riera-Ledesma (2015)		ILP formulation					Chagas et al. (2016)			VNS		
ID	R	T	UB	t(s)	UB	LB	Gap %	t(s)	Opt	Avg	Best	t(s)	Avg	Best	t(s)
076	R05	(*p*)	912†	24	**913**	913.0	0.00	2607	*	938.9	921	34	942.8	921	26
077	R06		945†	138	**948**	948.0	0.00	7141	*	1006.5	989	53	999.2	989	24
078	R07		793†	43	**801**	801.0	0.00	1181	*	900.6	804	124	871.4	804	44
079	R08		902	5	**902**	902.0	0.00	714	*	988.5	909	96	975.7	934	29
080	R09		874	11	**874**	874.0	0.00	2790	*	893.1	889	17	882.7	880	22
081	R05	(*q*)	1031♭	31	**942**	894.6	5.03	3 h		995.0	965	9	960.4	**942**	25
082	R06		1061♭	46	**1010**	952.9	5.65	3 h		1025.2	**1010**	9	1032.4	**1010**	21
083	R07		978♭	40	**931**	816.8	12.27	3 h		933.1	**931**	9	938.9	**931**	17
084	R08		1060♭	74	**995**	964.5	3.07	3 h		1013.4	998	9	1006.1	998	20
085	R09		982♭	36	**889**	867.3	2.44	3 h		908.0	902	10	895.5	**889**	21
086	R05	(*r*)	1200♭	2303	1159	887.8	23.40	3 h		1124.5	1116	5	1116.6	**1113**	12
087	R06		1191♭	302	**1121**	944.2	15.78	3 h		1135.9	1125	5	1131.4	**1121**	9
088	R07		1105♭	191	1141	792.5	30.54	3 h		1103.4	1051	5	1093.1	**1042**	8
089	R08		1191♭	776	1198	971.5	18.91	3 h		1167.4	**1147**	5	1177.2	**1147**	10
090	R09		1100♭	2342	1050	861.1	17.99	3 h		1055.6	1048	5	1046.0	**1033**	10
091	R05	(*s*)	947	4	**950**	914.5	3.74	3 h		1001.3	991	36	989.0	**950**	46
092	R06		1010♭	8	**1005**	1005.0	0.00	4512	*	1021.2	**1005**	55	1021.5	**1005**	38
093	R07		939	8	**952**	881.2	7.44	3 h		1006.9	963	156	1005.3	963	42
094	R08		953	7	**958**	937.8	2.11	3 h		1028.2	1009	98	1009.4	963	41
095	R09		945♭	5	**940**	909.3	3.26	3 h		950.1	942	35	942.7	941	35
096	R05	(*t*)	1101♭	298	1006	891.8	11.36	3 h		1049.7	1000	12	1018.2	**998**	36
097	R06		1059♭	17	**1017**	992.6	2.40	3 h		1105.0	1074	15	1057.3	**1017**	28
098	R07		1101♭	1911	1053	840.6	20.17	3 h		1028.3	**1021**	12	1050.5	**1021**	26
099	R08		1115♭	337	1061	906.5	14.57	3 h		1051.7	1039	16	1050.6	**1025**	31
100	R09		1002♭	15	957	861.0	10.03	3 h		972.9	957	15	970.6	**953**	33
101	R05	(*u*)	1259	1 h	1268	870.6	31.34	3 h		1195.9	1159	6	1165.2	**1149**	15
102	R06		1225♭	258	1197	977.3	18.36	3 h		1185.1	1179	6	1165.0	**1161**	14
103	R07		1325	1 h	1329	825.4	37.90	3 h		1241.2	**1232**	6	1235.2	**1232**	13
104	R08		1245	1 h	1336	951.0	28.82	3 h		1215.6	1207	6	1194.4	**1181**	14
105	R09		1169♭	909	1200	863.0	28.08	3 h		1153.1	1136	6	1132.8	**1118**	15
106	R05	(*v*)	1033	26	1079	904.8	16.15	3 h		1075.3	1062	112	1051.9	**1033**	93
107	R06		1051	34	1084	944.9	12.83	3 h		1085.3	1056	274	1074.4	**1051**	95
108	R07		1058	52	1106	861.5	22.11	3 h		1111.4	1095	75	1086.7	**1080**	93
109	R08		1076	26	1089	923.5	15.20	3 h		1093.4	**1069**	717	1102.9	1080	90
110	R09		1019	6	1030	863.9	16.13	3 h		1039.6	1023	420	1038.5	**1022**	95
111	R05	(*w*)	1142♭	2826	1289	844.7	34.47	3 h		1216.0	1194	14	1153.2	**1073**	59
112	R06		1188♭	331	1271	948.1	25.40	3 h		1212.8	1204	16	1166.3	**1093**	65
113	R07		1186♭	2603	1200	862.5	28.13	3 h		1210.1	1134	19	1189.4	**1098**	52
114	R08		1174♭	666	1198	907.6	24.24	3 h		1249.8	1159	19	1209.0	**1157**	82
115	R09		1098♭	2250	1283	814.0	36.56	3 h		1174.3	1087	24	1142.8	**1058**	64
116	R05	(*x*)	1333	1 h	1392	850.6	38.89	3 h		1348.9	1300	9	1293.3	**1262**	29
117	R06		1377	1 h	1700	937.7	44.84	3 h		1359.7	1320	9	1310.7	**1304**	29
118	R07		1412	1 h	1596	858.8	46.19	3 h		1388.9	**1338**	9	1359.7	**1338**	22
119	R08		1367	1 h	1520	909.3	40.18	3 h		1399.9	1361	10	1357.5	**1297**	27
120	R09		1578	1 h	1448	831.9	42.55	3 h		1314.8	1252	9	1263.1	**1240**	32
Average			1106.9	1061.3	1119.7	895.2	17.74	9781.1	$\frac{6}{45}$	1103.9	1075.0	58.0	1086.1	1058.2	36.6

Gap %) indicate the difficulty of the ILP formulation on finding optimal solutions, mainly, for the larger-size instances.

It is possible to notice that the VNS algorithm overcomes all results obtained by Silveira et al. (2015), finding solutions of the same or higher quality with less computational time. The VNS was also efficient in comparison to the ILP formulation, because for only three instances (ID 034, 050 and 061) the ILP formulation obtained better solutions.

We have also noticed that the average solution (column *Avg*) obtained by the algorithm proposed by Chagas et al. (2016) presents slightly better results than the VNS for most instances of group *C*. However, considering the best solutions obtained by each method (column *Best*), it is possible to notice that in 9 instances (ID 016, 019, 020, 034, 048, 050, 061, 063 and 070) the VNS algorithm has been able to find solutions of higher quality than those found by the algorithm proposed by Chagas et al. (2016). In the other 65 instances, both methods found the same solution and only in one instance (ID 068) the VNS has been worse.

Regarding the computation time, the VNS has been more efficient than the algorithm proposed by Chagas et al. (2016) for most instances. On average (last row of the tables) the VNS was 3.5 times faster than their algorithm in the smaller instances of group *C* and 1.2 faster for the larger-size instances also of group *C*.

Again, for instances of group $\neg C$, some results (instances with ID 076, 077 and 078) obtained by Iori and Riera-Ledesma (2015) were also found to be unfeasible. In addition, we noted that several results which are stated by Iori and Riera-Ledesma (2015) as optimal results are overcome by the ILP formulation and/or by our VNS and/or by the method proposed by Chagas et al. (2016). These results are highlighted by the character $\flat$ which was inserted in column UB that relates to the results of Iori and Riera-Ledesma (2015).

For most instances of group $\neg C$, when comparing the best solution values, the method based on VNS outperformed the method described in Chagas et al. (2016), with two exceptions (ID 079 and ID 109). Regarding computational time, comparing the average execution time of all methods (last row of Table 2.5), the heuristic based on the VNS metaheuristic was the faster one, proving to be a reliable heuristic.

2.4 Conclusions

In this chapter, we have addressed the Double Vehicle Routing Problem with Multiple Stacks (DVRPMS). For solving it, initially, we have introduced a new Integer Linear Programming (ILP) formulation. As the DVRPMS is a complex combinatorial problem, and finding an exact solution is time consuming, a heuristic based on Variable Neighborhood Search (VNS) algorithm has also been proposed to treat large-size instances.

In order to evaluate the quality of our solution approaches, we have performed tests with two sets of instances used in previous works that address the DVRPMS. The first set (named by C) contains instances in which the total capacity of the vehicles is equal to the total number of customer requests, while the second one (named by $\neg C$) contains instances in which the total number of customer requests is less than the total capacity of the vehicles.

Although the ILP formulation has not solved most larger-size instances of the group C and most instances of the group $\neg C$, this exact method was of fundamental importance for the DVRPMS because it highlighted inconsistencies in the values encountered in the literature. Since these inconsistencies have been raised in this work and our ILP formulation has not solved most instances to the optimality, it is hard to do a statistical analysis with all solutions. Though, when considering the average values of the best solutions, the VNS algorithm outperformed all methods and spent less computational time, especially instances of the group $\neg C$.

Chapter 3

The double traveling salesman problem with partial last-in-first-out loading constraints

In this chapter[1], we approach a variant of the DTSPMS (see Chapter 2 for more details) that was suggested in Petersen's Ph.D. thesis (Petersen, 2009) (see page 68, Section 2.5.2.4), which we have named as the Double Traveling Salesman Problem with Multiple Stack and Partial Last-In-First-Out Loading Constraints (DTSPMSPL). The DTSPMSPL, like the DTSPMS, arises in transportation companies responsible for transporting large and fragile items from a pickup area to a delivery area, where these two areas are widely separated. However, the DTSPMSPL is more general as rearrangement operations are allowed as long as they obey a partial LIFO policy, that is, a version of the LIFO policy that may be violated within a given reloading depth. As stated by Petersen (2009), the reason for a reloading depth is that replacing all items stored in the vehicle may be impractical due to the handling cost and the limited space available during reloading, so only a certain number of items may be placed outside the vehicle at any time. Thus, only the first L items from the top of each stack may be relocated at any time. Note that rearrangement operations allow constructing shorter tours than those in which the classical LIFO policy must be respected. Although the base operation cost of the DTSPMSPL is the total routing cost, an additional handling cost should be paid for each item rearranged. Therefore, as stated by Petersen (2009), partial LIFO constraints allow posing the question of what

[1] It has been compiled from paper "The double traveling salesman problem with partial last-in-first-out loading constraints". J. B. C. Chagas, T. A. M. Toffolo, M. J. F. Souza, and M. Iori. International Transactions in Operational Research, (2020). Available at https://doi.org/10.1111/itor.12876

price transportation companies would be willing to pay for the opportunity to move one item.

The objective of the DTSPMSPL is to find a route in each area in such a way that the total cost is a minimum, and there exists a feasible loading/unloading plan following the partial LIFO policy. The total cost involves the routing cost, i.e., the sum of traveled distances in both areas, and the number of reloading operations performed in the loading/unloading plan, which have their cost is given in terms of the routing cost.

To the best of our knowledge, no study to date investigates the DTSPMSPL. Thus, in this work, we start this investigation, addressing a particular case of the DTSPMSPL, where the vehicle has its loading compartment as a single horizontal stack. We have named this new transportation problem as the Double Traveling Salesman Problem with Partial Last-In-First-Out Loading Constraints (DTSPPL).

In order to clarify the characteristics of the DTSPMSPL/DTSPPL, we depict in Figure 3.1 a solution example of an instance with 6 customers, considering a reloading depth equal to 2, that is, $L = 2$. The vehicle starts its pickup route from the pickup depot (gray vertex on the top left part of the figure). It travels to the pickup position of customer 1, storing its item in the stack. Next, it visits the pickup position of customer 5, storing its item on the top of the horizontal stack. Then, it travels to pickup position customer 4. At this point, a rearrangement is performed: the item of customer 5 is removed from the stack, then the item of customer 4 is placed in the stack and finally the item of customer 5 is replaced into the stack. The vehicle continues its pickup route as shown in the figure until all items have been collected and stored in the stack, then the vehicle returns to the pickup depot. Notice that the reloading sequence may be in any order; thus, items do not need to remain in the same relative positions before their rearrangements, as occurs when the vehicle visits the pickup location of customer 6. Upon arrival at the pickup depot, the container is transferred to the depot (gray vertex on the top right part of the figure) located in the delivery area, from where it is again transferred to a vehicle that then executes the delivery operations. In our example, first, the vehicle travels to the delivery position of customer 2 without the requirement of any rearrangement. Next, it travels to the delivery position of customer 5, where items 6 and 3 need to be removed before delivering item 5. The delivery operations continue, as shown in the figure, until all customers are served. In the end, the vehicle returns to the delivery depot. Note that in this example, there were 3 rearrangements in the pickup area (loading plan) and 3 ones in the delivery area (unloading plan), totaling 6 rearrangements.

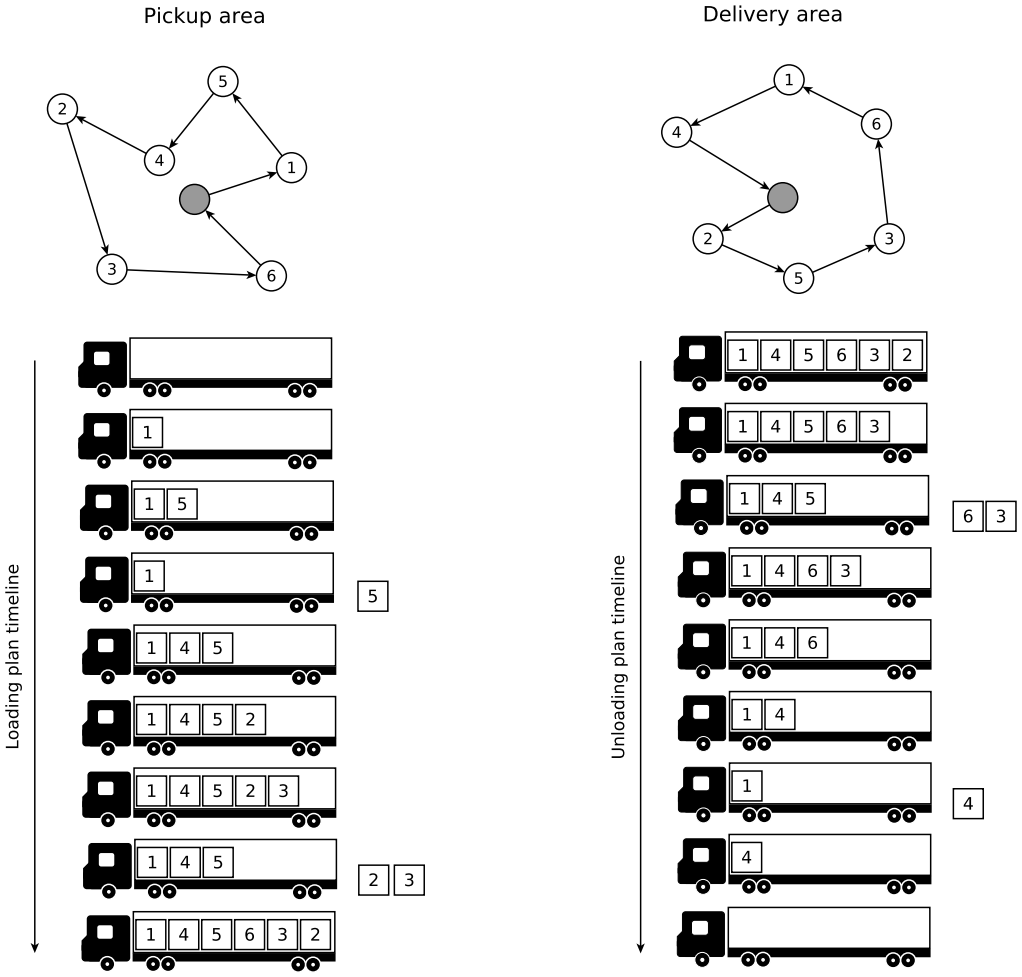


Figure 3.1: Solution of a DTSPPL instance involving 6 customers and reloading depth 2.

Figure 3.2 illustrates in a practical way the loading and unloading plan of the solution described in Figure 3.1. Note that each column in Figure 3.2 indicates the container configuration after each pickup or delivery operation. Note also that it is possible to determine which items have been relocated (highlighted in gray) and how they have been reloaded by analyzing adjacent pairs of container configurations.

No previous work has approached the DTSPMSPL/DTSPPL. Nonetheless, Ladany and Mehrez (1984) addressed a similar problem in which the reshuffling of all goods inside a container is allowed and causes costs and time losses. They investigated a real-world scenario in which identically sized crates should be transported from metropolitan area A to metropolitan area B using a single vehicle. The authors have solved small-size instances exactly by using an enumeration procedure. In addition, Veenstra et al. (2017) also approached a similar problem, named Pickup and Delivery

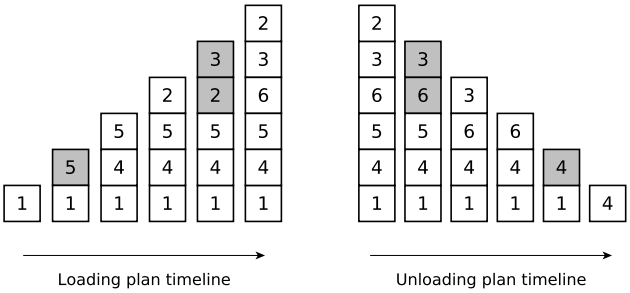


Figure 3.2: A practical representation of loading and unloading plans of the solution shown in Figure 3.1.

Traveling Salesman Problem with Handling Costs (PDTSPH). It is a variant of the PDTSPL where rearrangement operations are allowed only at delivery locations, and, as in the problem studied by Ladany and Mehrez (1984), there is no maximum depth for reloading, i.e., at any delivery location, all items stored in the container may be relocated. The authors proposed a binary integer program for the PDTSPH, considering that the reloading sequence is the inverse of the unloading sequence, i.e., the items remain in the same relative positions before their rearrangements. They have also developed a Large Neighborhood Search (LNS) heuristic, which considers the reloading policy adopted in the binary integer program and another one where the reloaded items are positioned in the sequence in which they will be delivered. Their results show that this last reloading policy reduces the number of rearrangement operations.

It is important to stress that by allowing rearrangement operations in both regions, a smaller reloading depth may be needed to rearrange items according to a given pair of tours π^P and π^D (pickup and delivery tours, respectively). To illustrate this, consider the two different pairs of tours in Figure 3.3. In the first pair (scenario #1), we exemplify what might happen when rearrangements are allowed only in the pickup region. Note that the item of the n-th customer is the last to be collected and also the last to be delivered. Therefore, a reloading depth equal to $n-1$ is needed to allow unloading all $n-1$ items before collecting the n-th item in the pickup region, and then store it in the first position of the container, thus preparing the container for all deliveries without any rearrangement operation. In turn, if rearrangements are allowed only in the delivery region, a reloading depth equal to $n-1$ to construct feasible loading and unloading plans from π^P and π^D is also needed in the second pair of tours (scenario #2). This reloading depth is needed since the first item to be delivered is stored in the first position of the container during collection.

$$\underbrace{\begin{array}{l}\pi^P = \langle 0 \to 1 \to 2 \to ... \to n-1 \to n \to 0 \rangle \\ \pi^D = \langle 0 \to n-1 \to n-2 \to ... \to n \to 0 \rangle\end{array}}_{\text{scenario \#1}} \qquad \underbrace{\begin{array}{l}\pi^P = \langle 0 \to 1 \to 2 \to ... \to n-1 \to n \to 0 \rangle \\ \pi^D = \langle 0 \to 1 \to n \to n-1 \to ... \to 2 \to 0 \rangle\end{array}}_{\text{scenario \#2}}$$

Figure 3.3: Scenarios that justify the importance of allowing rearrangement operations in both regions.

In both scenarios showed in Figure 3.3, when rearrangement operations are allowed in the pickup and delivery regions, a reloading depth equal to 1 is enough to construct feasible loading and unloading plans. Therefore, in the context of the DTSPMSPL/DTSPPL, where rearrangement many items may be impractical due to the handling cost and the limited space available during reloading, it is crucial to allow rearrangement operations in both regions. Note that this can be done without any additional resources regarding those already available in the DTSPMS context. The same equipment used to load/unload an item can be used to unload and reload other items at each pickup and delivery point.

As is commonly addressed in the literature, we do not allow rearrangement operations at the depot. However, we can formulate scenarios in which rearrangements are also allowed at the depot by considering a fictitious item localized at the depot. If rearrangements at the depot are interesting, the fictitious item will be used to do them. It must be stressed that, in this approach, the reloading depth at the depot is limited to the same one used on the routes.

In the remainder of this chapter, we present our contributions. In Section 3.1, we formally describe the DTSPPL via two Integer Linear Programming (ILP) formulations. Section 3.2 describes a heuristic algorithm based on the concept of the Biased Random-Key Genetic Algorithm (BRKGA), which is able to find high-quality solutions for the DTSPPL in shorter computational time. Section 3.3 reports the experiments and analyzes the performance of the proposed solution approaches. Finally, in Section 5.4, we present the conclusions and give suggestions for further investigations.

3.1 Problem description and mathematical formulations

In this section, we present the necessary notation to mathematically describe the DTSPPL and then propose two compact ILP formulations. Our mathematical formulations differ from each other in the way that route constraints are imposed. In both

formulations, it is needed to provide for the other constraints of the models the visiting order of the customers in order to construct the loading and unloading plans.

An alternative formulation to those described as follows in this section could be designed based on infeasible path constraints. In this formulation, the problem should be decomposed into its routing and loading/unloading components. The routing component goal would be to construct a tour for each region, while loading/unloading component would aim at solving a packing problem from the fixed pair of tours found by the routing component. The packing goal would be to construct a complete feasible solution for the problem or identify cuts that eliminate tours that do not allow construction feasible loading and unloading plans. This strategy was used, e.g., for solving the DTSPMS in a branch-and-cut algorithm proposed by Alba Martínez et al. (2013). As stated by Alba Martínez et al. (2013), in the DTSPMS context, from a given pair of tours, it is possible to construct a precedence graph, and then from it determining whether the tours are compatible with the LIFO constraints. Their separation algorithms have been developed from this property in order to find cuts for their branch-and-cut method. Note that our packing problem is much more complex. Due to partial LIFO constraints, we may not construct a precedence graph and work on it once items may be rearranged. Therefore, we cannot efficiently solve the DTSPPL by using this strategy.

3.1.1 Problem description

The DTSPPL can be formally described as follows. Let $C = \{1, 2, ..., n\}$ be the set of n customer requests, $V_c^P = \{1^P, 2^P, ..., n^P\}$ the set of pickup locations and $V_c^D = \{1^D, 2^D, ..., n^D\}$ the set of delivery locations. For each customer request $i \in C$, an item has to be transported from the pickup location i^P to the delivery location i^D.

The DTSPPL is defined on two complete directed graphs, $G^P = (V^P, A^P)$ and $G^D = (V^D, A^D)$, which represent the pickup and delivery areas, respectively. The sets $V^P = \{0^P\} \cup V_c^P$ and $V^D = \{0^D\} \cup V_c^D$ represent the vertices in each area, with 0^P and 0^D denoting the depots of the pickup and delivery areas, respectively. The sets of arcs in the pickup and delivery areas are defined by $A^P = \{(i^P, j^P, c_{ij}^P) \ \forall i^P \in V^P, \forall j^P \in V^P \mid j^P \neq i^P\}$ and $A^D = \{(i^D, j^D, c_{ij}^D) \ \forall i^D \in V^D, \forall j^D \in V^D \mid j^D \neq i^D\}$, where c_{ij}^P and c_{ij}^D correspond to the travel distances associated with arcs (i^P, j^P) and (i^D, j^D), respectively. For convenience of notation, when no confusion arises we also

refer to sets V_c^P and V_c^D as the set of requests C, and we use i to denote both i^P and i^D and (i, j) to denote both (i^P, j^P) and (i^D, j^D).

A feasible solution s for the DTSPPL consists of a Hamiltonian cycle on a graph G^P that starts at the pickup depot 0^P, another Hamiltonian cycle on graph G^D that begins at the delivery depot 0^D, and a loading/unloading plan. Besides, the two Hamiltonian cycles and the loading and unloading plan must obey the partial LIFO policy, which is defined by the maximum reloading depth L.

Let us denote by $\mathcal{F}$ the set of all feasible solutions for a DTSPPL instance. Each solution $s \in \mathcal{F}$ has a cost c_s that involves the travel distance on the two Hamiltonian cycles and the number of rearrangements performed on the loading and unloading plan, with a cost h associated with a single item rearrangement. The objective of the DTSPPL is to find a solution $s^* \in \mathcal{F}$, so that $c_{s^*} = \min_{s \in \mathcal{F}} c_s$.

3.1.2 Integer linear programming formulation 1

To better explain the first proposed ILP formulation (ILP1) for the DTSPPL, we categorize the constraints under three groups: (*i*) routes structuring constraints, (*ii*) loading/unloading plan and partial LIFO constraints, and (*iii*) reloading control constraints. Throughout mathematical modeling, we also use the notation $[a, b]$ to denote the set $\{a, a+1, ..., b-1, b\}$. Note that for any $a > b$, $[a, b]$ is an empty set. After describing all constraints, we present the objective function of the DTSPPL.

Route structuring constraints

In order to characterize the pickup and delivery routes, we define constraints (3.1)-(3.7), which use a binary decision variable x_{ij}^{kr}, $\forall r \in \{P, D\}$, $k \in [1, n+1]$, $(i, j) \in A^r$, that assumes value 1 if arc (i, j) is the k-th traveled arc by the vehicle in the area r, and value 0 otherwise.

$$\sum_{j\,:\,(0,\,j)\,\in\,A^r} x_{0j}^{1r} = 1 \qquad r \in \{P, D\} \tag{3.1}$$

$$\sum_{i\,:\,(i,\,0)\,\in\,A^r} x_{i0}^{n+1,r} = 1 \qquad r \in \{P, D\} \tag{3.2}$$

$$\sum_{(i,j) \in A^r} x_{ij}^{kr} = 1 \qquad r \in \{P, D\},\ k \in [2, n] \tag{3.3}$$

$$\sum_{k \in [1,n]} \sum_{i\,:\,(i,j) \in A^r} x_{ij}^{kr} = 1 \qquad r \in \{P, D\},\ j \in V_c^r \tag{3.4}$$

$$x_{ij}^{kr} \leq \sum_{i'\,:\,(i',i) \in A^r} x_{i'i}^{k-1,r} \qquad r \in \{P, D\},\ k \in [2, n+1],\ (i,j) \in A^r \tag{3.5}$$

$$x_{ij}^{kr} \in \{0, 1\} \qquad r \in \{P, D\},\ k \in [1, n+1],\ (i,j) \in A^r \tag{3.6}$$

Constraints (3.1) and (3.2) force the vehicle to leave from the depot and return to it using, respectively, the first and the last arc in each area. Constraints (3.3) guarantee that only a single arc may be the k-th one of each route. Constraints (3.4) guarantee that every customer is served. Constraints (3.5) establish the flow conservation for each vertex, and constraints (3.6) define the domain of the decision variables used to represent the routes.

Loading/unloading plan and partial LIFO constraints

For the purpose of representing the loading plan, we define a binary decision variable $y_{j\ell}^{kP}$, $\forall k \in [1, n]$, $\ell \in [1, k]$, $j \in C$, that assumes value 1 if the item referring to the customer request j is stored in position l on the k-th container configuration in the pickup area, and value 0 otherwise. We also define other binary decision variables $y_{j\ell}^{kD}$, $\forall k \in [1, n]$, $\ell \in [1, n-k+1]$, $j \in C$, to represent the unloading plan, which has the same meaning as the previous variable but considers the delivery area. With these variables, we can ensure the feasibility of the loading and unloading plans throughout constraints (3.7)-(3.15), which are next explained in detail.

$$\sum_{j \in C} y_{j\ell}^{kP} = 1 \qquad k \in [1, n],\ \ell \in [1, k] \tag{3.7}$$

$$\sum_{j \in C} y_{j\ell}^{kD} = 1 \qquad k \in [1, n],\ \ell \in [1, n-k+1] \tag{3.8}$$

$$\sum_{l \in [1,\, k]} y_{j\ell}^{kP} = \sum_{k' \in [1,\, k]} \sum_{i\,:\,(i,\,j) \in A^P} x_{ij}^{k'P} \qquad k \in [1,n],\, j \in C \tag{3.9}$$

$$\sum_{l \in [1,\, n-k+1]} y_{j\ell}^{kD} = \sum_{k' \in [k,\, n]} \sum_{i\,:\,(i,\,j) \in A^D} x_{ij}^{k'D} \qquad k \in [1,n],\, j \in C \tag{3.10}$$

Constraints (3.7) and (3.8) ensure that only one item must occupy each container position in each of its configurations. Constraints (3.9) establish that in the pickup area the k-th container configuration has to contain all items collected at the vertices $v_i,\ \forall i \leq k$. In turn, constraints (3.10) guarantee that in the delivery area the k-th container configuration has to contain all items that have not yet been delivered at the vertices $v_i,\ \forall i \geq k$. Figure 3.4 depicts the operation of constraints (3.9) and (3.10) for an instance with 6 customer requests.

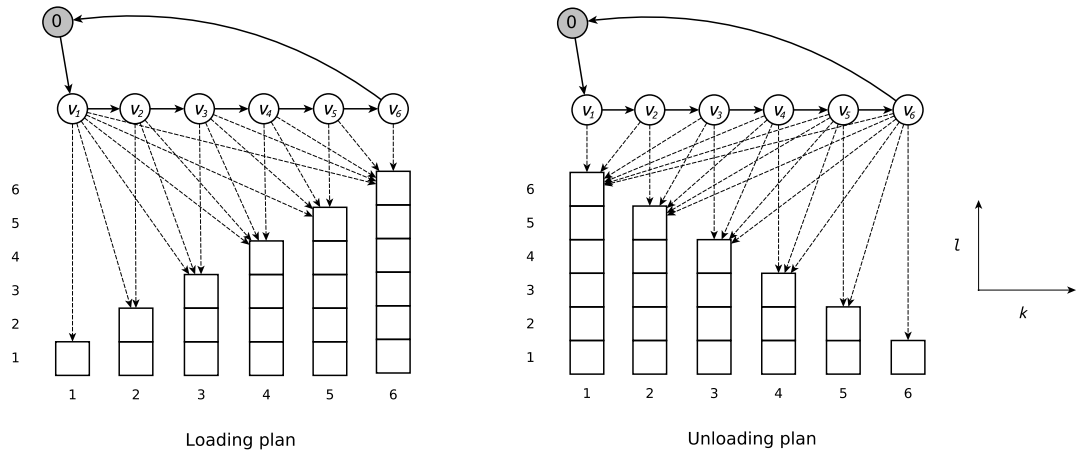


Figure 3.4: Graphical representation of constraints (3.9) and (3.10). Dashed arrows indicate which items must be in each container configuration according to the pickup and delivery tours, which are represented by the continuous lines that connect the vertices.

Constraints (3.11) certify that the first container configuration in the delivery area must be the same as the last container configuration in the pickup area. In other words, these constraints ensure that there is no rearrangement of items between the transfer from the pickup depot to the delivery depot.

$$y_{j\ell}^{1D} = y_{j\ell}^{nP} \qquad \ell \in [1,n], j \in C \tag{3.11}$$

Figure 3.5 illustrates the operation of constraints (3.11) for an instance with 6 customer requests. It also illustrates the operation of constraints (3.12) and (3.13) that ensure the loading plan obeys the partial LIFO policy, taking as example $L = 2$. Note that constraints (3.12) and (3.13) establish which items (represented by different geometric shapes) have to remain in their previous container positions in order not to violate the partial LIFO policy.

$$y_{j\ell}^{kP} = y_{j\ell}^{nP} \qquad k \in [1, n-1],\ \ell \in [1, k-L], j \in C \tag{3.12}$$

$$y_{j\ell}^{kD} = y_{j\ell}^{1D} \qquad k \in [2, n],\ \ell \in [1, n-k-L+1],\ j \in C \tag{3.13}$$

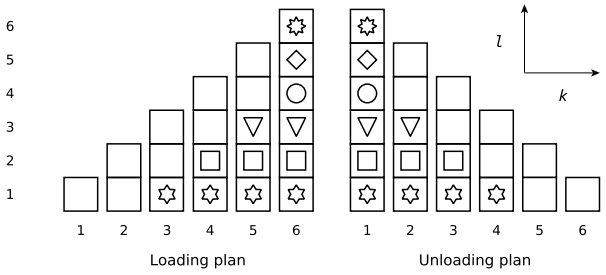


Figure 3.5: Graphical representation of constraints (3.11)-(3.13).

Finally, constraints (3.14) and (3.15) define the domain of the decision variables used to represent the loading and unloading plan.

$$y_{j\ell}^{kP} \in \{0, 1\} \qquad k \in [1, n],\ \ell \in [1, k],\ j \in C \tag{3.14}$$

$$y_{j\ell}^{kD} \in \{0, 1\} \qquad k \in [1, n],\ \ell \in [1, n-k+1],\ j \in C \tag{3.15}$$

Reloading control constraints

To determine how many rearrangements are performed from the loading and unloading plans, we define decision variables $z^{kr},\ \forall r \in \{P, D\}, k \in [1, n-1]$, that indicate the number of rearrangements made in the k-th container configuration in area r. We also define constraints (3.16) and (3.17), which are responsible for determining the number of rearrangements in the loading and unloading plans, respectively. Constraints (3.16)

analyze every pair of adjacent container configurations (k and k+1-th) of the loading plan to determine the number of rearrangements made in the k-th container configuration according to the k+1-th configuration. Figure 3.6 illustrates the operation of constraints (3.16) by exemplifying the calculation of the number of rearrangements made in the 3-th container configuration. In this example, the first ($\ell = 1$) item is not rearranged, as shown in the 4-th container configuration; while the second ($\ell = 2$) and third ($\ell = 3$) items are. Note that the number of rearrangements is given from the deeper change in the k-th container configuration. Constraints (3.17) are similar to constraints (3.16), but they count the rearrangements made in the delivery area. Finally, constraints (3.18) define the domain of these decision variables.

$$z^{kP} \geq \left(y_{j\ell}^{kP} - y_{j\ell}^{k+1,P} \right) \cdot \left(k - \ell + 1 \right) \qquad k \in [1, n-1],\ \ell \in [1, k-1],\ j \in C \tag{3.16}$$

$$z^{kD} \geq \left(y_{j\ell}^{k+1,D} - y_{j\ell}^{kD} \right) \cdot \left(n - k - \ell + 1 \right) \qquad k \in [1, n-1],\ \ell \in [1, n-k],\ j \in C \tag{3.17}$$

$$z^{kr} \in \mathbb{Z}^{+} \qquad r \in \{P, D\},\ k \in [1, n-1] \tag{3.18}$$

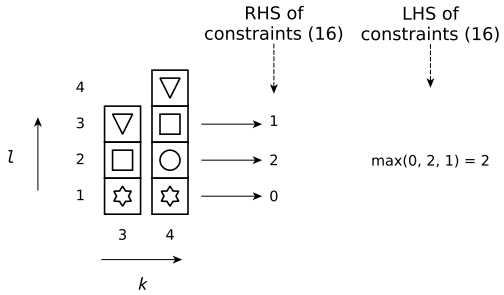


Figure 3.6: Graphical representation of constraints (3.16).

Objective function

Constraints (3.1)-(3.18) are enough to represent all feasible solutions of the DTSPPL. Therefore, to complete the first mathematical model, we define the objective function

(3.19), which minimizes the total cost. It involves the distance traveled in both areas, as well as the cost of all rearrangements performed.

$$\min \sum_{r \in \{P,D\}} \sum_{(i,j) \in A^r} c_{ij}^r \cdot \sum_{k \in [1,n+1]} x_{ij}^{kr} + h \cdot \sum_{r \in \{P,D\}} \sum_{k \in [1,n-1]} z^{kr} \tag{3.19}$$

3.1.3 Integer linear programming formulation 2

Our second ILP formulation (ILP2) uses binary decision variables χ_{ij}^r, $\forall r \in \{P,D\}$, $(i,j) \in A^r$, to describe the vehicle route in each area. More specifically, each variable χ_{ij}^r assumes value 1 if arc (i,j) is traveled by the vehicle in area r, and value 0 otherwise. Moreover, we use an integer variable u_j^r, $\forall r \in \{P,D\}$, $j \in [0,n]$, that gives the position of vertex j in the route of area r. With these new decision variables, we can use constraints (3.20)-(3.25), instead of (3.1)-(3.6) adopted for ILP1, to define the routes of the vehicle.

$$\sum_{j\,:\,(i,j) \in A^r} \chi_{ij}^r = 1 \qquad r \in \{P,D\},\ i \in V^r \tag{3.20}$$

$$\sum_{i\,:\,(i,j) \in A^r} \chi_{ij}^r = 1 \qquad r \in \{P,D\},\ j \in V^r \tag{3.21}$$

$$u_j^r \geq u_i^r + 1 - n \cdot \left(1 - \chi_{ij}^r\right) \qquad r \in \{P,D\},\ (i,j) \in A^r : j \neq 0 \tag{3.22}$$

$$\chi_{ij}^r \in \{0,1\} \qquad r \in \{P,D\},\ (i,j) \in A^r \tag{3.23}$$

$$u_0^r = 0 \qquad r \in \{P,D\} \tag{3.24}$$

$$u_j^r \in \{a \in \mathbb{Z}^+ : a \leq n\} \qquad r \in \{P,D\},\ j \in C \tag{3.25}$$

Constraints (3.20), (3.21) and (3.23) ensure that each pickup and delivery location is visited exactly once, whilst constraints (3.22), (3.24) and (3.25) impose the subcycle elimination.

To complete ILP2, we define constraints (3.26) and (3.27), and the objective function (3.28). Moreover, we also include constraints (3.7), (3.8) and (3.11)-(3.18), which have been previously defined for ILP1.

$$1 \geq \sum_{l \in [1,\, k]} y_{j\ell}^{kP} \geq \frac{k - u_j^P + 1}{k} \qquad k \in [1, n],\ j \in C \tag{3.26}$$

$$1 \geq \sum_{l \in [1,\, n-k+1]} y_{j\ell}^{kD} \geq \frac{u_j^D - k + 1}{n - k + 1} \qquad k \in [1, n],\ j \in C \tag{3.27}$$

Note that constraints (3.26) and (3.27) have the same aim as constraints (3.9) and (3.10), which ensure the correct assignment of items to the loading compartment throughout the pickup and delivery routes. Note also that equation (3.28), similarly to (3.19), describes the total DTSPPL cost to be minimized.

$$\min \sum_{r \in \{P,\, D\}} \sum_{(i,j) \in A^r} c_{ij}^r \cdot \chi_{ij}^r \;+\; h \cdot \sum_{r \in \{P,\, D\}} \sum_{k \in [1,\, n-1]} z^{kr} \tag{3.28}$$

3.1.4 Providing the ILP models with a feasible initial solution

We have solved models ILP1 and ILP2 using Gurobi Optimizer, which is currently one of the best ILP optimization solvers. However, we have noticed that, even for small instances, Gurobi had difficulties in solving both models within a reasonable time. Therefore, in order to help the optimization process, we compute a feasible DTSPPL solution and initialize both models with it. For the initial solution, we consider the case where rearrangements are not allowed (i.e., $L = 0$). Thus, as every loading/unloading operation must verify the classic LIFO principle, the pickup and delivery routes must be exactly opposite each other, since the vehicle has its loading compartment as a single stack. Therefore, we can solve the initial solution by solving a TSP instance on a graph where each arc (i, j) is associated a cost $c_{ij} = c_{ij}^P + c_{ji}^D$. This strategy was also used by Felipe et al. (2009) to compute an initial solution for the DTSPMS. In this work, we solve a TSP instance via the classical two-index model for the TSP (see. e.g., Gutin and Punnen (2006)) by adding subtour elimination constraints iteratively until the incumbent solution does not contain subtours.

3.2 A biased random-key genetic algorithm

In this section, we describe a heuristic algorithm based on the Biased Random-Key Genetic Algorithm (BRKGA) (Gonçalves and Resende, 2011). Although metaheuristic algorithms based on local search such as Tabu Search, Variable Neighborhood Search and Iterated Local Search, among others (see, e.g., (Talbi, 2009)) are more often applied to address vehicle routing problems, we have chosen an evolutionary algorithm that works with an indirect representation of its individuals (where each individual represents a problem solution). The justification for this choice is that the DTSPPL has a high dependency on routes and loading/unloading plans. So, working with direct solutions may not be practicable, since defining efficient moves that can navigate between feasible solutions, and, especially, escape from infeasible solution space is extremely hard. In turn, an indirect representation of DTSPPL solutions allows us to navigate in the feasible solution space through simple genetic operators, quickly producing a high number of feasible solutions. This representation strategy has been successfully applied to several complex optimization problems in recent years (Gonçalves and Resende, 2012; Resende, 2012; Gonçalves and Resende, 2013; Lalla-Ruiz et al., 2014; Gonçalves and Resende, 2015; Santos and Chagas, 2018).

In the remainder of this section, we present the main components of the proposed BRKGA (Sections 3.2.1-3.2.6) and then describe how these components are combined together (Section 3.2.7).

3.2.1 Encoding structure

BRKGAs, as well as classic Genetic Algorithms (GAs) (see Mitchell (1998) for a reference), are evolutionary metaheuristics that mimic the processes of Darwinian Evolution. Basically, a GA maintains a population of individuals, each encoding a solution to the problem at hand. Through the use of stochastic evolutionary processes (selection, recombination, and diversification) over the population, individuals with higher fitness tend to survive, thus guiding the algorithm to explore more promising regions of the solutions space.

Each individual in BRKGAs is represented by a vector of random-keys, i.e., a vector of real numbers that assume values in the continuous interval [0, 1]. This representation is generic because it is independent of the problem addressed. Therefore, a

deterministic procedure (to be presented later) is necessary to decode each individual (a vector of random-keys) to a feasible solution of the problem at hand (the DTSPPL in our case).

In Figure 3.7, we show the structure defined to represent each BRKGA individual for the DTSPPL. We divide this structure into three partitions: pickup route, loading plan, and unloading plan and delivery route. The first one consists of n random-keys, which are responsible for determining the pickup route. The next $\sum_{k=1}^{n} \min(k,\ L+1)$ random-keys determine the loading plan carried out along the pickup route. Finally, the last $\sum_{k=1}^{n} \min(n-k+1, L+1)$ random-keys define the entire unloading plan and also the delivery route.

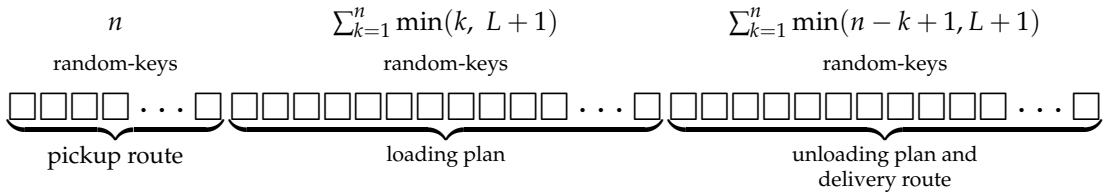


Figure 3.7: Chromosome structure.

3.2.2 Decoding procedure

As stated before, each individual $p \in \mathcal{P}$ has a generic representation in a BRKGA. Therefore, to determine the fitness of p, we developed a procedure that generates a feasible solution s from p. The fitness of p is then defined proportionally to the quality of s.

Our decoding procedure consists of three stages, which must be sequentially performed due to the dependency among them. Figure 3.8 depicts how these stages are performed in order to decode the solution shown in Figure 3.1 from a vector of random-keys. Initially, the n pickup locations are mapped on the first n random-keys. Then, we sort the pickup locations according to the values of the mapped random-keys. The sorted pickup locations define the pickup route π^P performed by the vehicle. In Figure 3.1, the first n random-keys produce the pickup route $\pi^P = \langle 0, 1, 5, 4, 2, 3, 6, 0 \rangle$.

We characterize the loading plan from π^P and the next $\sum_{k=1}^{n} \min(k,\ L+1)$ random-keys. The loading plan is created iteratively in n steps, where each of them depends on

the previous one. Iteratively, for each $k \in \{1, 2, ..., n\}$, we map the $\min(k,\ L+1)$ items that may be relocated without violating the partial LIFO constraints to $\min(k,\ L+1)$ random-keys. Next, we sort the items in non-decreasing order according to the values of the mapped random-keys to define the k-th container configuration in the pickup area. For the sake of clarity, consider the steps shown in Figure 3.8. At first, the container is empty, so we just store in it the item (highlighted in gray) of the first customer visited according to π^P. Notice that, though unnecessary, we kept a random-key (0.48 in this example) to decode the first operation of the loading plan. We have decided to keep it for simplicity, to follow the same pattern used for the other loading operations. To define the second container configuration, we map item 1 (it may be relocated from the previous container configuration) to the random-key 0.59 and item 5 (the second customer visited according to π^P) to the random-key 0.61. After sorting these items according to the values of the mapped random-keys, items 1 and 5 are stored in the container following the sorted order. The decoding process continues by mapping the items that may be relocated at each time to the random-keys, and then sort them to define each remaining pickup container configuration.

Finally, the unloading plan and the delivery route are defined from the loading plan and the last random-keys. Similarly as before, it is created iteratively in n steps, where each of one is dependent on the previous one. For each $k \in \{1, 2, ..., n\}$ we map the $\min(n-k+1, L+1)$ items that may be delivered without violating the partial LIFO constraints to $\min(n-k+1, L+1)$ random-keys. Then, we sort the items in non-decreasing order according to the values of the mapped random-keys. The $\min(n-k+1, L+1)-1$ first items in the order to define the k-th container configuration in the delivery area and the last item is then delivered, iteratively making up the delivery route. In Figure 3.8, take for example the first step ($k=1$), where we map items 6, 3 and 2 (only these items may be relocated because in this case, the reloading depth is 2) to the random keys 0.68, 0.94 and 0.95, respectively. After sorting these random-keys, item 2 (highlighted in gray), which is mapped to the greatest random-key (0.95 in this example) is delivered. The other items are stored in the container following the order of their random-keys. This process is repeated until the whole unloading plan has been completed.

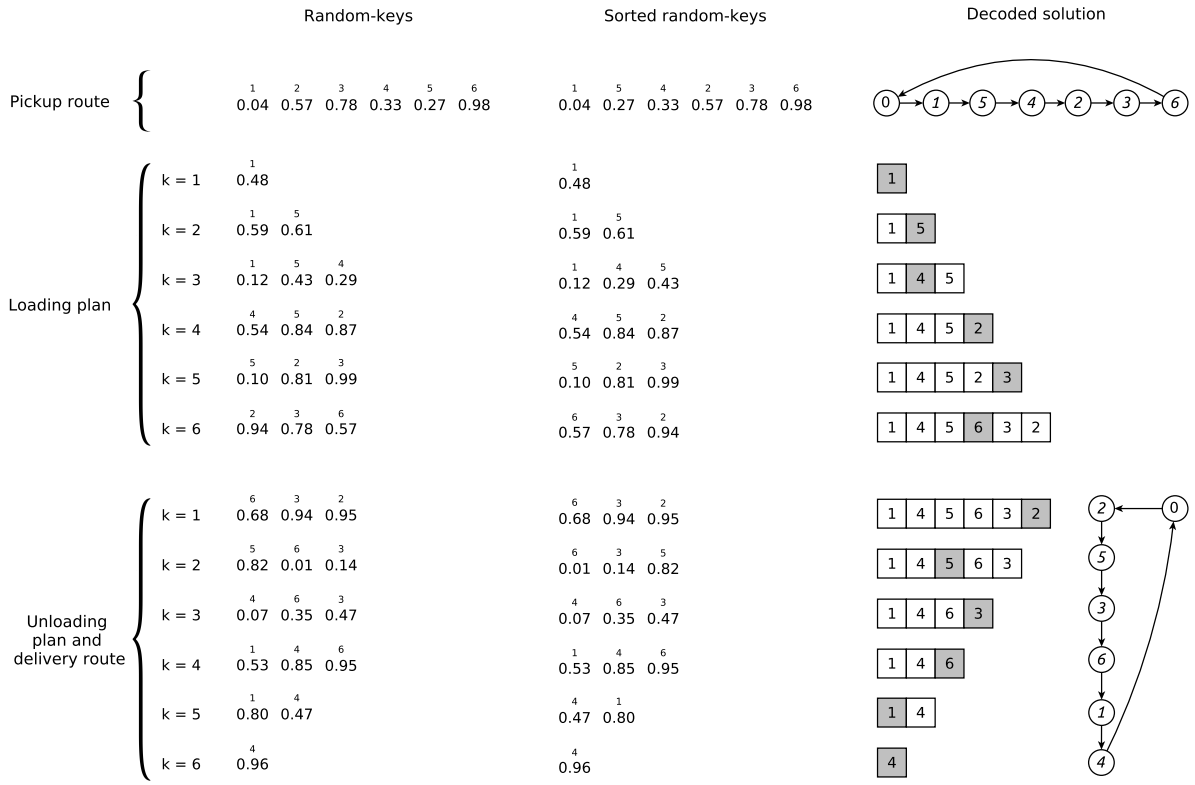


Figure 3.8: Decoding of the vector of random-keys $\langle$ 0.04, 0.57, 0.78, 0.33, 0.27, 0.98, 0.48, 0.59, 0.61, 0.12, 0.43, 0.29, 0.54, 0.84, 0.87, 0.10, 0.81, 0.99, 0.94, 0.78, 0.57, 0.68, 0.94, 0.78, 0.57, 0.68, 0.94, 0.78, 0.57, 0.68, 0.94, 0.95, 0.82, 0.01, 0.14, 0.07, 0.35, 0.47, 0.53, 0.85, 0.95, 0.80, 0.47, 0.96 $\rangle$ to the solution shown in Figure 3.1.

3.2.3 Initial population

Our BRKGA maintains a population $\mathcal{P}$ of N individuals throughout the evolutionary process. To make the initial population, we create $N - 1$ random individuals, where each random-key of each individual is generated independently at random in the real interval $[0, 1]$. Furthermore, to introduce an orientation (hopefully a good one) to the search procedure of the BRKGA, we create and insert into the initial population an individual that represents the initial solution computed to initialize the mathematical models, i.e., the optimal DTSPPL solution in which no rearrangements of items are performed. This last individual is created following the reverse process of the decoding procedure previously described.

3.2.4 Biased crossover

In order to combine the genetic information from the parents and generate new offsprings, a BRKGA uses a biased crossover operator. This crossover always involves two parents, where one is randomly selected from the elite group $\mathcal{P}_e$, and the other is randomly chosen from the non-elite group $\mathcal{P}_{\bar{e}}$. The groups $\mathcal{P}_e$ and $\mathcal{P}_{\bar{e}}$ are formed at each generation of the algorithm after all individuals of population $\mathcal{P}$ have been decoded. Group $\mathcal{P}_e$ is formed by the individuals with greater fitness, while $\mathcal{P}_{\bar{e}}$ is formed by the other individuals, i.e., $\mathcal{P}_{\bar{e}} = \mathcal{P} \setminus \mathcal{P}_e$. Moreover, the biased crossover operator has a parameter ρ_e that defines the probability of each random-key of the elite parent to be inherited by the offspring individual. More precisely, from an elite parent a and a non-elite parent b, we can generate an offspring c according to the biased crossover as follows:

$$c_i \leftarrow \begin{cases} a_i & \text{if } \mathit{random}\,(0,\,1) \;\leq\; \rho_e \\ b_i & \text{otherwise} \end{cases} \qquad \forall i \in \{1, 2, ..., M\}$$

where M is the number of random-keys of the individuals and a_i, b_i and c_i are, respectively, the i-th random-key of individuals a, b and c.

3.2.5 Mutant individuals

Unlike most GAs, BRKGAs do not contain mutation operators. Instead, to maintain population diversity, they use mutant individuals, which are merely new individuals generated by choosing for each random-key a real number between 0 and 1.

3.2.6 Next generation

In the BRKGA scheme, from any generation k, a new population is formed based on the current population $\mathcal{P}$. First, all elite individuals of generation k in $\mathcal{P}_e$ are copied into the new population (generation $k + 1$) without any modification. Next, some mutant individuals are added to the new population to maintain high population diversity. Finally, to complete the new population, new individuals are added by using the biased crossover operator.

3.2.7 Overall BRKGA

The previous components are organized as described in Algorithm 3. Initially (line 1), the best solution found by the algorithm is initialized as an empty solution. Then, at line 2, the initial population is generated. While the stopping criterion is not achieved, the algorithm performs its evolutionary cycle (lines 3 to 11). At lines 4 to 6, all individuals in the current population are decoded and the best solution found is possibly updated. After selecting the individual elites (line 7) and generating the mutant individuals (line 8) as well as the offspring individuals (line 9), the algorithm updates the population of individuals (line 10). At the end of the algorithm (line 12), the best solution found is returned.

Algorithm 3: Biased Random-Key Genetic Algorithm (BRKGA).

```
1  s^best ← ∅
2  P ← initial population with N individuals
3  repeat
4  |  foreach p ∈ P do
5  |  |  s ← individual p decoded
6  |  |  if s is better than s^best then s^best ← s end
7  |  P_e ← set of the N_e best individuals (elite) from P
8  |  P_m ← set of N_m mutant individuals
9  |  P_o ← set of N − N_e − N_m offspring individuals
10 |  P ← P_e ∪ P_m ∪ P_o
11 until time limit is reached
12 return s^best
```

3.3 Computational experiments

In this section, we present the computational experiments performed to study the performance of the proposed solution approaches. As there is no previous work on the DTSPPL, we compare the proposed approaches. Here, we graphically present the results obtained, whereas all numerical results for each instance can be found at https://github.com/jonatasbcchagas/ilps_brkga_dtsppl, where is also available our code, as well as all best solutions (tours and loading/unloading plans) found by each solution approach.

Our solution approaches have been coded in C/C++ language. Mathematical models have been solved using Gurobi solver version 9.0.1. The proposed BRKGA has been coded from the framework developed by Toso and Resende (2015). All experiments have been sequentially (nonparallel) performed on an Intel(R) Xeon(R) E5-2660 (2.20GHz), running under CentOS Linux 7 (Core).

3.3.1 Benchmarking instances

To assess the quality of the proposed solution methods, we have defined a comprehensive set of 1080 DTSPPL instances. The instances are divided into 216 different types in order to analyze the solution methods on different instance characteristics. Each type is described by the number of customers n, the reloading depth L, and the cost of each rearrangement h. The number of customers n varies in {6, 8, 10, 12, 15, 20}, while the reloading depth L and the cost h vary in {1, 2, 3, 4, 5, n} and {0, 1, 2, 5, 10, 20}, respectively. Note that for $h = 0$, we allow rearrangements according to the reloading depth L without any reloading cost. In turn, for $L = n$, we allow any rearrangement of the items, regardless of the number of items introduced in the container. Although this last configuration seems very unpractical in real-world applications, we have considered it in our computational experiments so as to have an interesting comparison term. For each type of instances, we have used the areas R05, R06, R07, R08, and R09 defined by Petersen and Madsen (2009) for the DTSPMS. Each of these areas consists of two sets, where one defines the pickup locations (pickup region), and the other defines the delivery locations (delivery region). The locations of each region have been generated randomly in a 100×100 square. The distance between any two points of each region is the Euclidean distance rounded to the nearest integer, following the conventions from the TSPLIB. The first point of each region, which is fixed in coordinates $(50, 50)$, corresponds to the depot, while the next n points define the n customer locations.

3.3.2 Parameter settings

Our mathematical models ILP1 and ILP2 have been solved by using Gurobi solver with all its default settings. The exceptions are the runtime, which has been limited to one hour, and the optimization process, which has been limited to use only a single processor core. Regarding the parameters of the BRKGA, we have used as stopping criterion the same execution time that both mathematical models have been limited, i.e.,

one hour. For the other BRKGA parameters, we have used the automatic configuration method I/F-Race (Birattari et al., 2010) to find the most suitable configuration. We have used the implementation of I/F-Race provided by the Irace package (López-Ibáñez et al., 2016b), which is implemented in the R language and is based on the iterated racing procedure. In our tuning experiments, we have used all Irace default settings, except for the parameter *maxExperiments*, which has been set to 5000. This parameter defines the stopping criterion of the tuning process. We refer the readers to López-Ibáñez et al. (2016a) for a complete user guide of the Irace package.

In Table 3.1, we describe the BRKGA parameters as well as the tested values for them. Note that the population size N is given in terms of the size of each individual. Moreover, the elite population size N_e and mutant population size N_m are granted in terms of N. After a vast experiment that used a sample of 10% of all 1080 instances, Irace pointed out the parameters highlighted in bold in Table 3.1 as the best ones.

Table 3.1: BRKGA parameters.

Parameter	Description	Tested values
N	population size	$1M, 2M, 5M, 10M, 20M, 50M, 100M, \mathbf{200M}, 500M$
N_e	elite population size	$0.05N, \mathbf{0.10N}, 0.15N, 0.20N, 0.25N, 0.30N$
N_m	mutant population size	$0.05N, 0.10N, 0.15N, 0.20N, \mathbf{0.25N}, 0.30N$
ρ_e	elite allele inheritance probability	$0.55, 0.60, 0.65, \mathbf{0.70}, 0.75, 0.80, 0.85, 0.90$

M is the number of random-keys of each individual.

3.3.3 ILP1 *vs.* ILP2

Our first analysis of results contrasts the mathematical models ILP1 and ILP2. We begin by comparing the performance of each model regarding the optimal solutions found. The results that we obtained are shown in Figure 3.9, where each cell represents a single test instance (horizontal axis informs L and h, and vertical axis informs *area* and n). We indicate with markers which of the instances have been solved to proven optimality by each model. Empty cells indicate for those instances that no model has been able to prove optimality within one hour of processing time, while markers $\diamond$ and $\circ$ evince those solved by models ILP1 and ILP2, respectively. We point out with $\star$ when both models have been solved to proven optimality.

It can be noticed from Figure 3.9 that the proposed mathematical models have been able to solve all instances with 6 customers, and almost all instances with 8

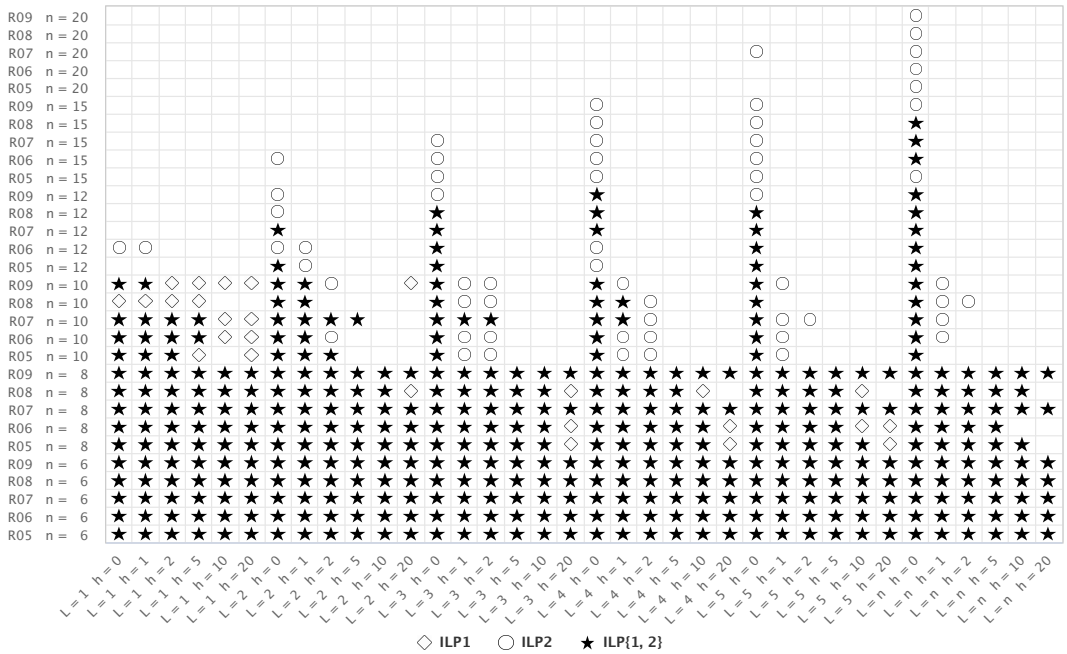


Figure 3.9: Instances solved to proven optimality.

customers. Both models have presented difficulties in solving instances with 10 or more customers, especially with the increase of the reloading depth (L) and the cost of each rearrangement (h). However, we can observe that for instances when there are no rearrangement costs, i.e., $h = 0$, both models perform better with the increase of the reloading depth. Indeed, note that the optimal solutions for instances with $h = 0$ tend to approximate the solution formed by the pickup and delivery optimal tours as the reloading depth increases. This trend results in an easier combinatorial problem, which has been verified in the performance of our models.

To better investigate the behavior of the mathematical models ILP1 and ILP2, we measure the percentage variation of their lower bounds (`ILP1_LB` and `ILP2_LB`, respectively) reached at the end of computation by calculating $($`ILP1_LB - ILP2_LB`$)$ `/ max(ILP1_LB, ILP2_LB)` $\times$ `100%`. The percentage variations obtained between the lower bounds are graphically shown in Figure 3.10, where we have used a heatmap visualization to emphasize larger variations. Note that positive variation values (highlighted in shades of red) indicate that model ILP1 has reached higher lower bounds than those reached by the model ILP2. In contrast, negative variation values (highlighted in shades of blue) indicate the opposite behavior. Besides, the higher

the absolute value (more intense color), the higher the difference between the lower bounds.

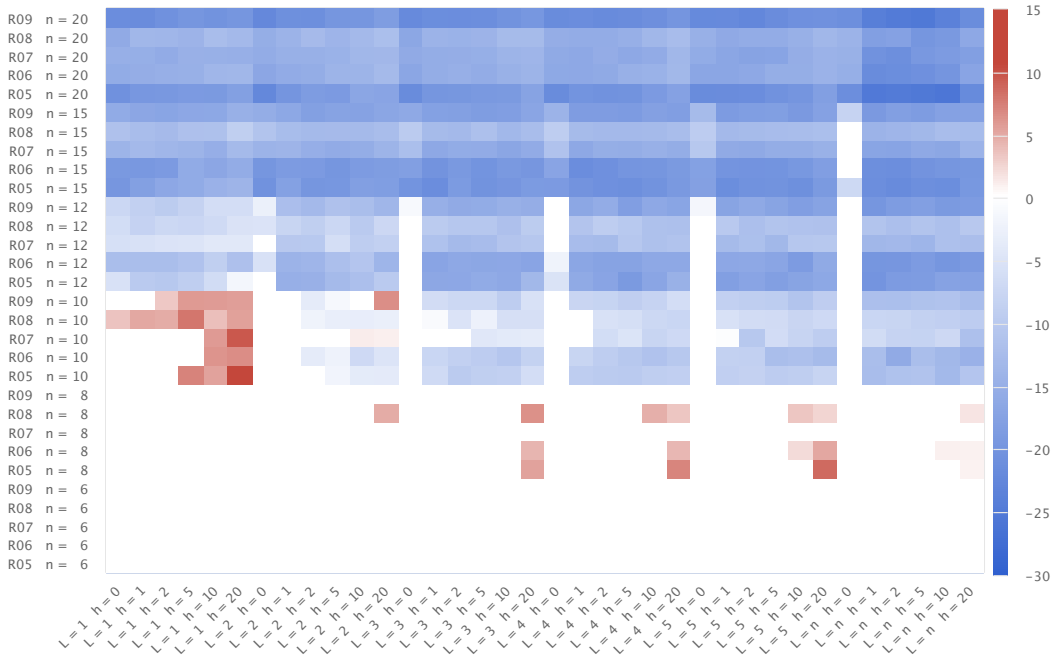


Figure 3.10: Percentage variation between the lower bounds `ILP1_LB` and `ILP2_LB`.

From the results reported in Figure 3.10, we can observe that model ILP2 is more effective than ILP1 for most of the instances regarding the lower bounds achieved. It must be stressed that the most instances where the model ILP1 has found tighter lower bounds are those in which have been solved to proven optimality only by that model, as shown previously in Figure 3.9. Besides, we can see that the absolute value of the negative variation is higher than the positive variation, thus indicating, in general, a better performance of model ILP2 concerning the lower bounds. Interestingly, we have noted in our experiments that for many larger instances even the linear relaxation of model ILP2 is tighter than the lower bound reached at the end of one hour of processing time of the model ILP1. We recall the readers interested in the detailed numerical results that we made them available at https://github.com/jonatasbcchagas/ilps_brkga_dtsppl along with our code and solutions.

We compare now the models ILP1 and ILP2 regarding their upper bounds (`ILP1_UB` and `ILP2_UB`, respectively) reached at the end of computing. Figure 3.11 reports a similar heatmap visualization to the one shown in Figure 3.10. However, each cell now

represents the calculated value as (ILP2_UB - ILP1_UB) / min(ILP1_UB, ILP2_UB) × 100%. Note that cells with more intense colors indicate a higher difference between the upper bounds. While positive values (highlighted in shades of red) indicate that model ILP2 has found better solutions than those found by the model ILP1, negative values (highlighted in shades of blue) indicate the opposite behavior.

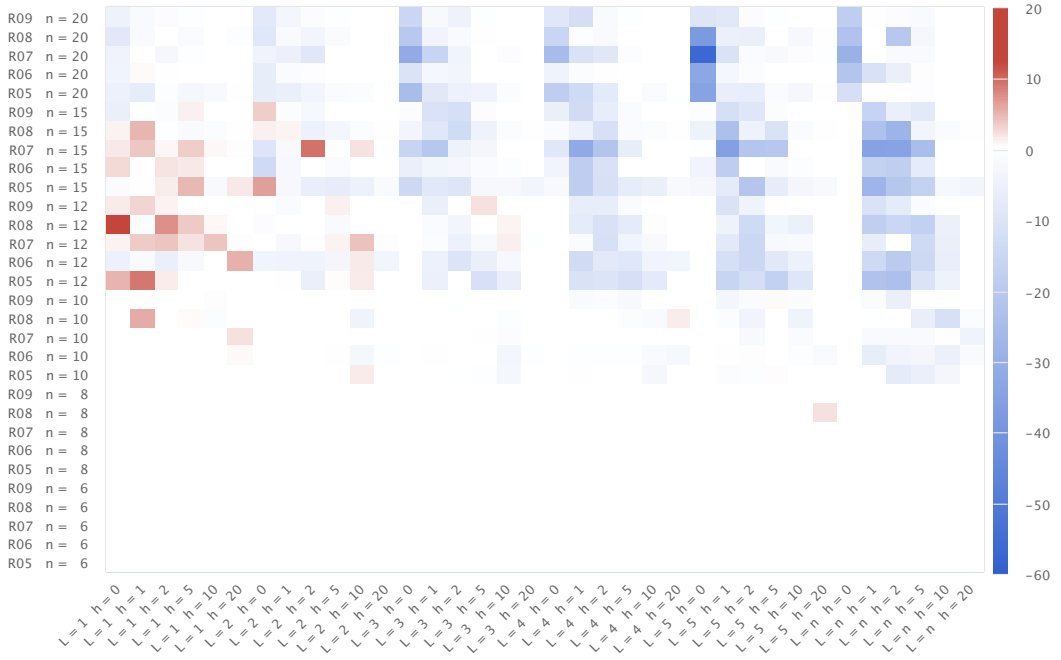


Figure 3.11: Percentage variation between the upper bounds ILP1_UB and ILP2_UB.

Although the models have shown substantial differences concerning their lower bounds, the results reported in Figure 3.11 show that both models have obtained solutions with the same and similar quality (white and light blue/red cells), i.e., same and similar upper bounds, for several instances. Disregarding the smaller instances where both models have found optimal solutions, this fact has befallen on many instances that involve large rearrangement costs because the models have not even been able to improve their initial incumbent solution (which we recall is the optimal solution obtained when reloading operations are not allowed). It indicates that in these instances the customers are located in such a way that no rearrangement is attractive or that both models have difficulty finding a way to accomplish them.

For the cases where the models showed significant differences between their upper bounds, we can observe that model ILP1 has performed better for those instances with smaller rearrangement depth. On the other hand, model ILP2 has achieved better

solutions for more cases. Furthermore, it has also reached higher absolute differences between the upper bounds found by the model ILP1.

We conclude the analysis of our models by examining the gap, which is computed as `(UB - LB) / UB × 100%`, between the best lower and upper bounds (`LB` and `UB`, respectively) reached by both models. To be clear, the `LB` and `UB` values are calculated as `max(ILP1_LB, ILP2_LB)` and `min(ILP1_UB, ILP2_UB)`, respectively. In Figure 3.12, we compare these values for all types of instances, except for those types that involve 6 customers. For these cases, both models have been able to solve all instances to proven optimality, and, consequently, gap values are zeros. The results reported in the figure consist of the average for all five instances of each type, i.e., average over five instances (R05, R06, ..., R09). better analyze the results, we plot them into six different lines, wherein each of them contains all instances with the same rearrangement cost.

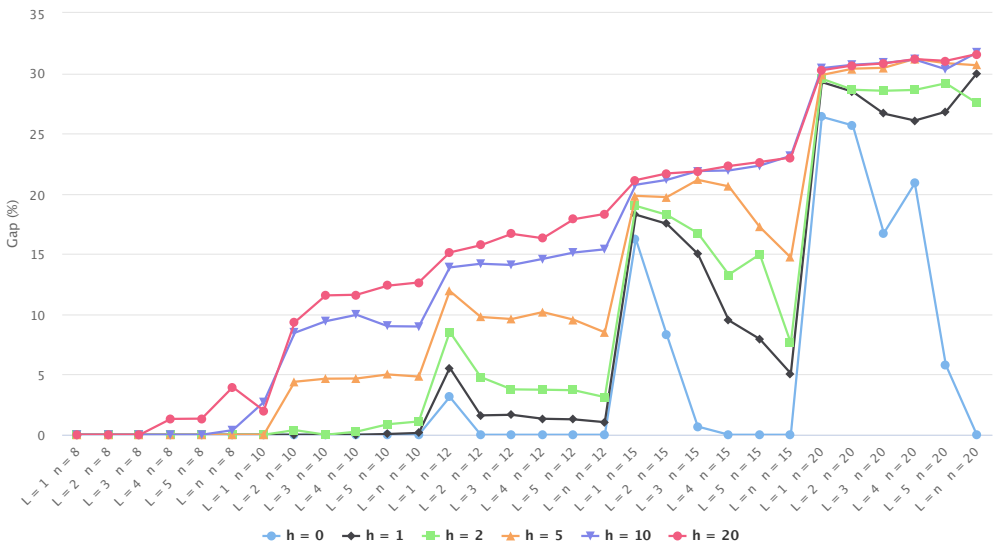


Figure 3.12: Relative gap between the best lower and upper bounds.

As can be seen from the gaps reported in Figure 3.12, the models' performance is strongly influenced by the rearrangement cost. Note that the higher the rearrangement cost, the larger the gap is, with few exceptions. For larger instances, the gap exceeds the value of 30%, thus indicating the models' difficulty to solve these instances. A closer look reveals, in general, that for rearrangement cost up to 5, that gap decreases as the reloading depth increases. This befalls because rearrangement operations are more attractive in these cases to minimize the total operation costs, reducing the difference between lower and upper bounds.

3.3.4 ILPs *vs.* BRKGA

We focus now on the computational analysis of the proposed heuristic algorithm. For this purpose, we have run our BRKGA 10 independent times on each instance, and then used the average value ($\texttt{BRKGA}_{\texttt{avg}}$) of the objective function in these runs in our analysis. To assess the quality of our BRKGA, we compare the $\texttt{BRKGA}_{\texttt{avg}}$ values with the best upper bound (UB) found by the mathematical models by means of a heatmap schema to emphasize the difference between the quality of the solutions obtained.

In Figure 3.10, a heatmap is reported, where each one of its cells reports the percentage difference between $\texttt{BRKGA}_{\texttt{avg}}$ and UB of a specific instance, which is identified as in the heatmaps previously presented. The value of each cell is calculated as $(\texttt{BRKGA}_{\texttt{avg}} - \texttt{UB}) \;/\; \texttt{min}(\texttt{BRKGA}_{\texttt{avg}}, \texttt{UB}) \times 100\%$. Now, cells with negative values (highlighted in shades of blue) indicate that BRKGA has found, on average, better solutions than those found by mathematical models. In turn, cells with positive values (highlighted in shades of red) indicate a better performance of the models. The higher the absolute value (more intense color) the higher the difference between the quality of the solutions.

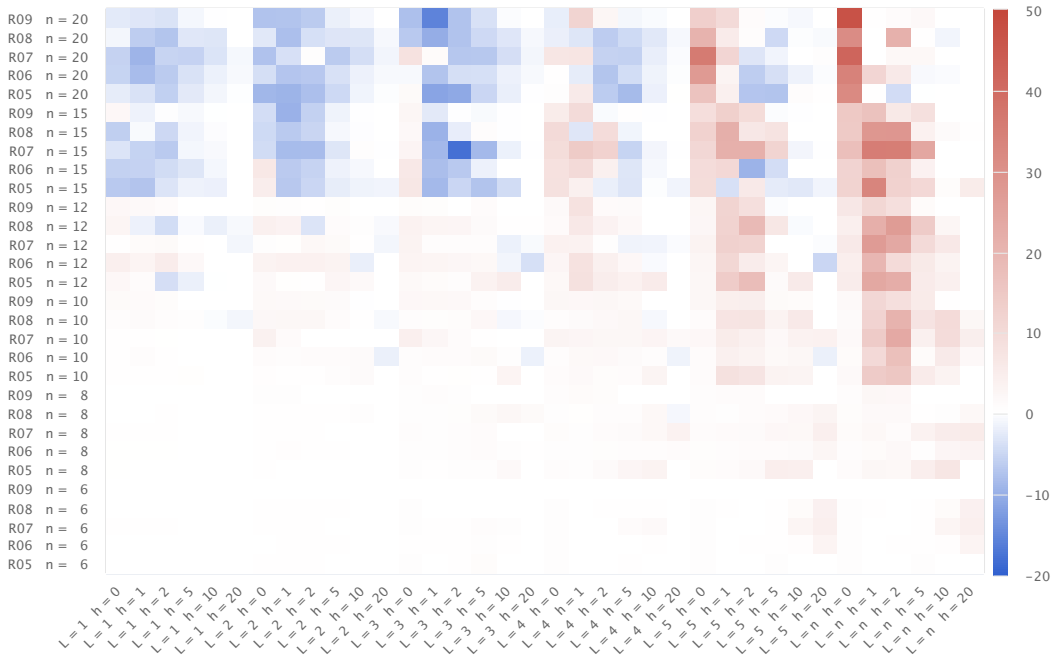


Figure 3.13: Percentage variation between $\texttt{BRKGA}_{\texttt{avg}}$ and UB.

It can be noticed from Figure 3.13 that our BRKGA has been able to find better solutions than those found by the mathematical models for almost all types of instances that involve reloading depth up to 3. On the other hand, the BRKGA has difficulty dealing with larger reloading depths. For these cases, the models have performed significantly better than our proposed heuristic, although there are some exceptions (see e.g. instances with $L = 5$, $h = \{2, 5, 10\}$, and $n = 20$). Note that the models have performed better for those instances with no limited reloading depth $(L = n)$ and smallest rearrangement costs $(h = \{0, 1, 2\})$. As stated before, for these instances, we have an easier combinatorial problem. Our models, especially the ILP2, take advantage of this behavior, while the BRKGA does not.

Now, we compare the quality of solutions reported in Figure 3.13 according to the time spent from reaching them. To this end, Figure 3.14 shows the average time spent over the five instances of each type. To better analyze, we plot these results into six separate charts, wherein each of them contains all instances with the same reloading depth.

From Figure 3.14, we can note that BRKGA has had a faster convergence for most of the instances when compared with the models, which has had a positive behavior for instances with reloading depth up to 3 where it has been able to find several better solutions. In general, we can affirm that all our solution approaches have faster convergence behavior as the rearrangement cost increases, which was expected as few or no rearrangements would be interesting for these cases.

Finally, we analyze the percentage improvement of the best solution achieved concerning the optimal solution that does not perform any rearrangement, i.e., when the classic LIFO policy is met. In Figure 3.15, we show for each type of instance (a combination of L, n, and h) the percentage improvement in terms of costs obtained by allowing rearrangement items within a reloading depth L and paying a cost h for each item rearranged. Note that all types of instances with the same rearrangement cost are described together in a single line (same color). For example, note that for the type of instance with $L = 2$, $n = 8$, and $h = 5$, we have obtained around 10% of improvement concerning the optimal solution without rearrangements. As expected, the lower the cost of rearrangement, the higher the percentage of improvement is. Besides, a closer look at the improvement rates shows that when rearrangements are less costly, a deeper reloading limit also implies more significant gains since a higher number of items may be rearranged. Note also that, even when rearrangements are more

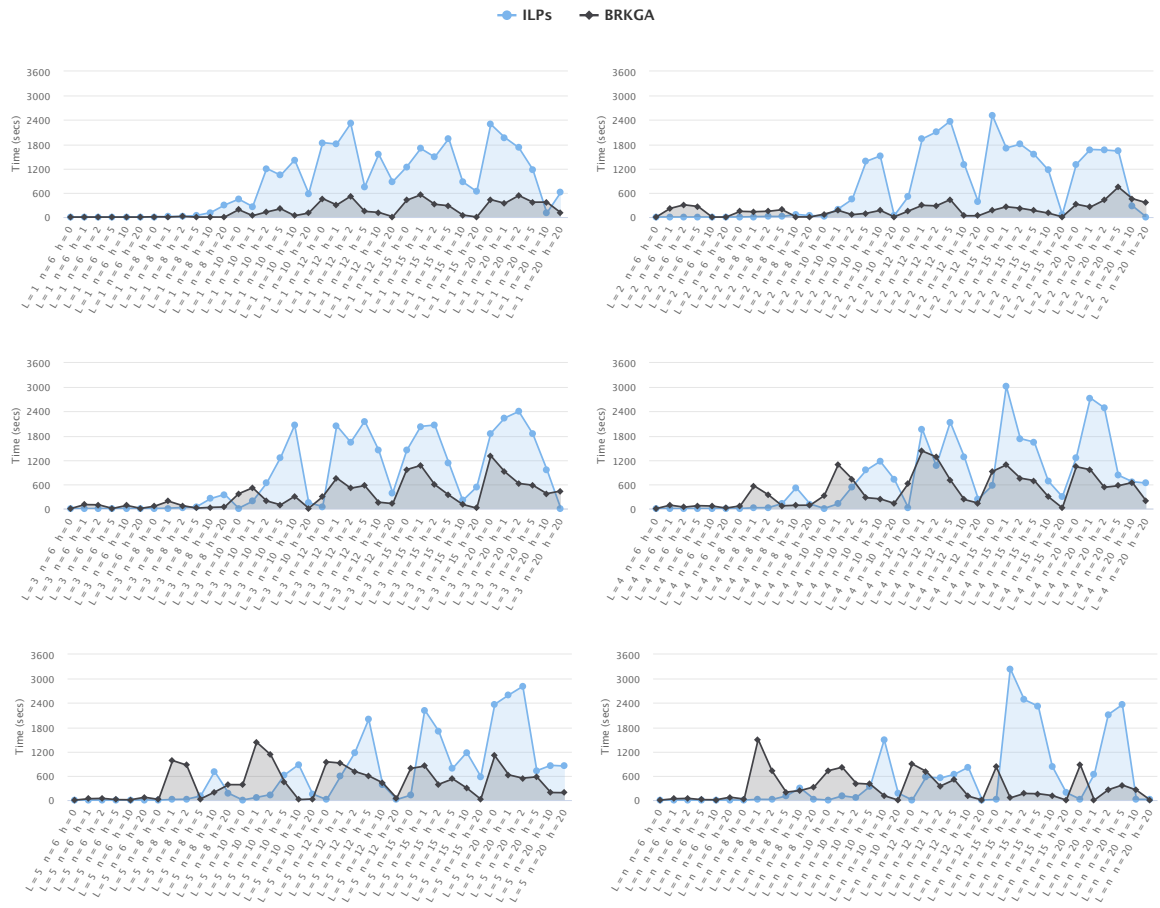


Figure 3.14: Computational time to reach $\texttt{BRKGA}_{\texttt{avg}}$ and UB for their solution approaches.

expensive, some improvements have been reached, especially for smaller instances where our approaches have been able to find solutions with better-proven quality.

3.4 Conclusions and open perspectives

We have approached the Double Traveling Salesman Problem with Partial Last-In-First-Out Loading Constraints (DTSPPL), a pickup-and-delivery single-vehicle routing problem. In this problem, the vehicle has its loading compartment as a single stack, and all pickup and delivery operations must obey a version of the LIFO policy that may be violated within a given reloading depth. We have presented two Integer Linear Programming (ILP) formulations, and we have developed a heuristic algorithm based on the Biased Random-Key Genetic Algorithm (BRKGA) metaheuristic.

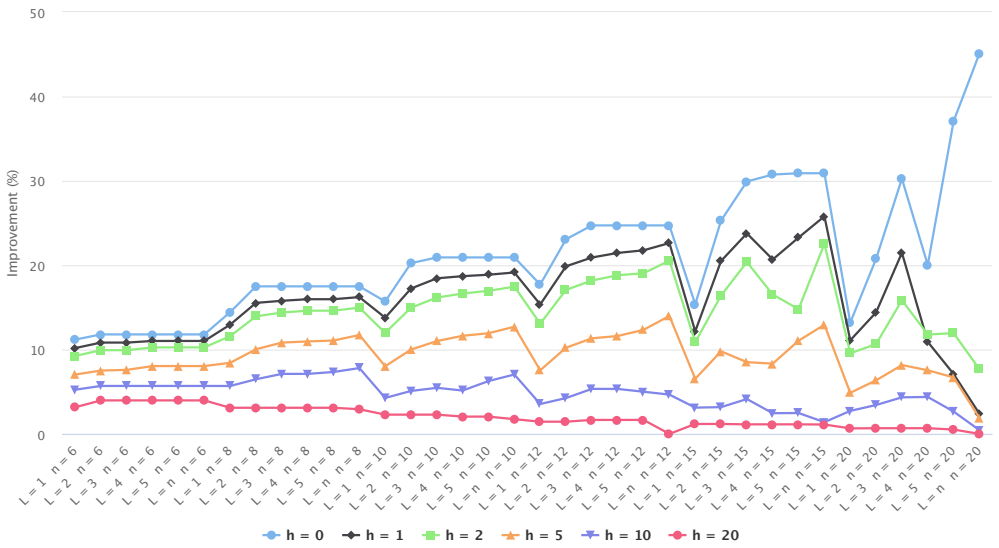


Figure 3.15: Percentage improvement with partial LIFO loading.

The performance of the ILP formulations and the BRKGA has been studied on a comprehensive set of instances built from the DTSPMS benchmark instances. Both ILP formulations have been able to solve to proven optimality only the smaller instances within one hour of processing time. One of them has had tighter lower bounds for almost all instances, although both formulations have shown similar performance concerning their upper bounds found at the end of the computation. The BRKGA found good quality solutions for all instances, requiring on average short computing times.

There are many possibilities for extending this work. Maybe the most relevant one would be to model and solve the DTSPMSPL, the version of the DTSPPL where the loading compartment of the vehicle is divided into multiple stacks instead of a single one. Because of the increased computational complexity, we expect metaheuristic techniques to be the best option to find good solutions for moderate size instances. Similarly as proposed by Chagas et al. (2020a) for the DVRPMS, it is possible to represent a DTSPPL/DTSPMSPL solution from only the container configuration with all items stored in it. Then, we can determine the optimal pickup and delivery routes from a dynamic programming algorithm. Thus, the search space of a heuristic algorithm would be just the container configurations, while the dynamic programming would find, for each container configuration, the pair of tours, and, consequently, how to rearrange items throughout the tours if it is needed. Another important extension of the DTSPPL/DTSPMSPL would be to consider a fleet of vehicles to serve the cus-

tomers. A larger number of vehicles would cause an increase in the flexibility of the loading/unloading operations, which could lead to a reduction in operating costs. In order to formulate more general problems, would also be interesting to consider partial LIFO loading in situations where backhaul deliveries are dropped, i.e., situations when deliveries are allow without the need to perform all pickup operations before, as in (Côté et al., 2009; Cordeau et al., 2010). Finally, as multi-objective formulations provide a more powerful optimization tools for decision making, it would be opportune to formulate pickup-and-delivery problems with partial LIFO as bi-objective problems by minimizing the total routing cost and the number of rearrangement operations.

Chapter 4

The bi-objective traveling thief problem

In this chapter[1], we address the Traveling Thief Problem (TTP) (Bonyadi et al., 2013), an academic multi-component problem. It is particularly important because it combines the classical traveling salesperson problem (TSP) and the knapsack problem (KP) – both of which are very well studied in isolation – and because of the interaction of both components can be adjusted. In brief, the TTP comprises a thief stealing items with weights and profits from a number of cities. The thief has to visit all cities once and collect items such that the overall profit is maximized. The thief uses a knapsack of limited capacity and pays rent for it proportional to the overall travel duration. To make the two components (TSP and KP) interdependent, the speed of the thief is made non-linearly dependent on the weight of the items picked so far. The interactions of the TSP and the KP in the TTP result in a complex problem that is hard to solve by tackling the components separately.

Basically, the TTP seeks to optimize the overall traveling time and the profit made through stealing items. Most of the research focused on the single-objective problem, where the objectives are composed by using a weighted sum. To be more precise, the profit is reduced by the costs due to renting the knapsack, which is calculated by multiplying the overall traveling time by a renting rate.

The TTP has been gaining fast attention due to its challenging interconnected multi-components structure, and also propelled by several competitions[2] organized

[1] It has been compiled from paper "A weighted-sum method for solving the bi-objective traveling thief problem". J. B. C. Chagas, M. Wagner. Submitted to a journal.

[2] https://cs.adelaide.edu.au/~optlog/research/combinatorial.php

to solve it, which have led to significant progress in improving the performance of solvers. Among these, are iterative and local search heuristics (Polyakovskiy et al., 2014; Faulkner et al., 2015; Maity and Das, 2020), solution approaches based on co-evolutionary strategies (Bonyadi et al., 2014; El Yafrani and Ahiod, 2015; Namazi et al., 2019), memetic algorithms (Mei et al., 2014; El Yafrani and Ahiod, 2016), swarm-intelligence based approaches (Wagner, 2016; Zouari et al., 2019), simulated annealing algorithm (El Yafrani and Ahiod, 2018) and evolutionary strategy with probabilistic distribution model to construct high-valued solution from low-level heuristics (Martins et al., 2017). Exact approaches have also been considered, however they are limited to address very small instances (Wu et al., 2017).

As the TTP's components are interlinked, multi-objective considerations that investigate the interactions via the idea of "trade-off"-relationships have been becoming increasingly popular. For example, Yafrani et al. (2017) created a fully-heuristic approach that generates diverse sets of solutions, while being competitive with the state-of-the-art single-objective algorithms. Wu et al. (2018) considered a bi-objective version of the TTP, which used dynamic programming as an optimal subsolver, where the objectives were the total weight and the TTP objective score. At two recent competitions[3,4], a bi-objective TTP (BITTP) variant has been used that trades off the total profit of the items and the travel time. The same BITTP variant was investigated by Blank et al. (2017), who proposed a simple algorithm for solving the problem. More recently, Chagas et al. (2020b) proposed a customized NSGA-II with biased random-key encoding. The authors have evaluated their algorithm on 9 instances, the same ones used in the aforementioned BITTP competitions. Their algorithm has shown to be effective according to the competition results.

In this work, we also address the BITTP variant used in competitions with a simple and effective heuristic approach. Specifically, our contributions with this work are:

1. We have realized that we can decompose the multi-objective problem into a number of single-objective ones using a simple weighted-sum method (Zadeh, 1963), which is one of the oldest strategies for addressing multi-objective optimization problems (Ramanathan, 2006; Marler and Arora, 2010; Stanimirovic et al., 2011; Galand and Spanjaard, 2012).

[3] *EMO-2019* https://www.egr.msu.edu/coinlab/blankjul/emo19-thief/
[4] *GECCO-2019* https://www.egr.msu.edu/coinlab/blankjul/gecco19-thief/

2. We tackle each single-objective problem through a two-stage heuristic by constructing a tour for the thief and then from it, we determine the packing plan with the stolen items. We use well-known efficient strategies for finding good tours and a problem-specific packing heuristic, which is a randomized version of a popular deterministic heuristic for the single-objective TTP, for determining the items stolen by the thief.

3. We incorporate into our algorithm the concepts of exploration and exploitation, which are aspects of effective search procedures (Črepinšek et al., 2013; Qi et al., 2015) by combining with our two-stages strategy, efficient local search operators already used in the single-objective TTP.

4. To investigate the contributions that our algorithmic components have, we tune our solution method on 96 groups of instances and characterize the resulting configurations.

5. We compare our approach with the tuned variant of Chagas et al. (2020b), with the competition entries of the two competitions, and with single-objective TTP solvers.

In the remainder of this chapter, we first define the BITTP in Section 4.1. Then, in Section 4.2, we describe our weighted-sum method, where the decomposition is based on the respective influence of the two interacting components. There, we also introduce a randomized version of a popular packing strategy. Section 5.3 contains the computational evaluation: the tuning of configurations and their characterisation, and the comparison with a range of (tuned) approaches from the literature, with the entries for two recent BITTP competitions, and with single-objective TTP solvers.

4.1 Problem definition

The Bi-objective Traveling Thief Problem (BITTP) can be formally described as follows. There is a set of m items $\{1, 2, \ldots, m\}$ distributed among a set of n cities $\{1, 2, \ldots, n\}$. For any pair of cities $i, j \in \{1, 2, \ldots, n\}$, the distance $d(i, j)$ between them is known. Every city, except the first one, contains a subset of the m items. Each item $j \in \{1, 2, \ldots, m\}$ has a profit p_j and a weight w_j associated. There is a single thief who has to visit all cities exactly once starting from the first city and returning back to it in the end (TSP component). The thief can make a profit by stealing items and storing

them in a knapsack with a limited capacity W (KP component). As stolen items are stored in the knapsack, it becomes heavier, and the thief travels more slowly, with a velocity inversely proportional to the knapsack weight. Specifically, the thief can move with a speed $v = v_{max} - w \times (v_{max} - v_{min}) \,/\, W$, where w is the current weight of their knapsack. Consequently, when the knapsack is empty, the thief can move with the maximum speed v_{max}; when the knapsack is full, the thief moves with the minimum speed v_{min}.

Any feasible solution for the BITTP can be represented through a pair $\langle \pi, z \rangle$, where $\pi = \langle \pi_1, \pi_2, \ldots, \pi_n \rangle$ is a permutation of all cities in the order they are visited by the thief, and $z = \langle z_1, z_2, \ldots, z_m \rangle$ is a binary vector representing the packing plan ($z_j = 1$ if item j is collected, and 0 otherwise) adopted by the thief throughout their robbery journey. We can formally express the space of feasible solutions for the BITTP by constraints (4.1) to (4.5).

$$\pi_i \neq \pi_j \qquad i \in \{1, 2, \ldots, n\}, j \in \{i+1, i+2, \ldots, n\} \tag{4.1}$$

$$\pi_i \in \{1, 2, \ldots, n\} \qquad i \in \{1, 2, \ldots, n\} \tag{4.2}$$

$$\pi_1 = 1 \tag{4.3}$$

$$\sum_{j=1}^{m} z_j \cdot w_j \leq W \tag{4.4}$$

$$z_j \in \{0, 1\} \qquad j \in \{1, 2, \ldots, m\} \tag{4.5}$$

Constraints (4.1) and (4.2) ensure that each city is visited exactly once, while constraint (4.3) establishes that the thief must start their journey from city 1. Constraints (4.4) and (4.5) ensure, respectively, that the knapsack capacity is not exceeded, and that each item may be collected only once.

The objectives of the BITTP are to maximize the profit of the collected items and to minimize the total traveling time spent by the thief to conclude their journey. These objectives are mathematically defined according to Equations (4.6) and (4.7), respectively.

$$\max\ g(z) = \sum_{j=1}^{m} p_j \cdot z_j \tag{4.6}$$

$$\min\ h(\pi, z) = \sum_{i=1}^{n-1} \frac{d(\pi_i, \pi_{i+1})}{v_{max} - v \cdot \omega(i, \pi, z)} + \frac{d(\pi_n, \pi_1)}{v_{max} - v \cdot \omega(n, \pi, z)} \tag{4.7}$$

Note that while the objective (4.6) is calculated directly from the packing plan z, the calculation of the objective (4.7) is more complex. Since the speed of the thief depends on the current weight of their knapsack, it may change after visiting each city. Therefore, it is necessary to know the traveling speed between each pair of cities in order to calculate the total traveling time. For this purpose, it is necessary to determine the total weight of the knapsack after visiting each city i according to the tour π and the packing plan z, which is denoted by $\omega(i, \pi, z)$ and is calculated as described in Equation (4.8). Hence, all speeds of the thief throughout their journey, and, consequently, the total traveling time can be computed.

$$\omega(i, \pi, z) = \sum_{k=1}^{i} \sum_{j=1}^{m} w_j \cdot z_j \cdot \begin{cases} 1 & \text{if item } j \text{ is localized in city } \pi_k \\ 0 & \text{otherwise} \end{cases} \tag{4.8}$$

It is important to note that the objectives of the BITTP are conflicting with each other, as optimizing each one of them independently does not necessarily produce a good solution in terms of the other objective. Indeed, for faster tours, the thief should not collect items or collect a few items with small weights. On the other hand, for collecting sets of items with high profit, the thief travels slowly due to the weight of the collected items. Therefore, there is no single solution that simultaneously optimizes both objectives, but a set of solutions, called Pareto-optimal solutions, in which each solution is non-dominated in terms of their objective values by any other solution.

4.2 Problem-solving methodology

Throughout this section, we discuss the methodology we have adopted in order to find high-quality non-dominated solutions for the BITTP. We describe in detail all

components of our proposed algorithm as well as all the decisions made during its design development.

4.2.1 The overall algorithm

Our proposed algorithm is based on the weighted-sum method (WSM) (Zadeh, 1963), a well-known strategy for addressing multi-objective optimization problems (Marler and Arora, 2010). Basically, its core idea consists of converting the multi-objective problem at hand into several single-objective problems by using different convex combinations of the original objectives. Then, each one of the created single-objective problems is solved in order to generate non-dominated solutions for the multi-objective problem (Das and Dennis, 1997). Note that the optimal solution for each single-objective problem is a Pareto-optimal solution for the multi-objective problem, because, if this were not the case, then there must exist another feasible solution with an improvement on at least one of the objectives without worsening the others. Hence, that solution would have a better value according to the weighted-sum objective function.

According to Marler and Arora (2010), the WSM is often used for addressing real-world applications, especially for those with just two objective functions, not only to provide multiple solutions widely spread across the space of the objectives, but also to provide a single solution that reflects preferences presumably incorporated in the selection of a single set of weights for the objectives. WSM has also given rise to very popular multi-objective decomposition-based optimization algorithms like MOEA/D (Zhang and Li, 2007).

Limitations of WSMs include their inability to capture Pareto-optimal solutions that lie in non-convex portions of the Pareto-optimal curve, and also that they do not necessarily generate a dispersed distribution of solutions in the Pareto-optimal set, even with a consistent change in weights attributed to the objectives. Throughout the chapter, we point out why these limitations do not affect our algorithm.

For the BITTP, our proposed WSM converts the objective functions (4.6) and (4.7) into the weighted-sum objective function (4.9) by including a scalar value α that may assume any real number between 0 and 1. In addition, we have included in weighted-sum objective function the renting rate R defined by Polyakovskiy et al. (2014) for the set of TTP instances, which is widely used as benchmarking in TTP related researches.

As stated by Polyakovskiy et al. (2014) the renting value has been tailored to each TTP instance, and its value establishes the connection between both TTP components. It is important to emphasize that the renting values vary widely among the benchmarking TTP instances. Thus, by varying the α values, we will be creating new TTP instances with different weights/importance for their components, but they will still have the tailored influence of the renting rate.

$$\max \quad f(\pi, z, \alpha) = \alpha \cdot g(z) - (1 - \alpha) \cdot R \cdot h(\pi, z) \tag{4.9}$$

Although exact algorithms exist for the TTP, they are limited to solving very small instances within a reasonable computational time (Wu et al., 2017). In fact, unless $\mathcal{P} = \mathcal{NP}$, it is not possible to develop an exact strategy able to solve general TTP instances in polynomial time. Therefore, we solve each new TTP instance by using concepts of effective heuristic approaches proposed for the TTP over the years. Consequently, there is no guarantee that our WSM finds Pareto-optimal solutions. On the other hand, it is able to find solutions possibly located in non-convex portions of the Pareto-optimal curve. Indeed, there is no convex combination of the two objectives whose global optimal value corresponds a solution located in non-convex portions. However, since each single-objective problem is approached with a heuristic strategy, these solutions can be achieved when the heuristic fails to find the global optimal value.

As the TTP has gained increasing attention since its proposition, several approaches have emerged to solve it. Some of them use techniques that require higher computational effort, whereas others bet on low-level search operators, which can also produce high-quality solutions with shorter computation time (Polyakovskiy et al., 2014; Faulkner et al., 2015; Wagner et al., 2018). As the BITTP demands a set of non-dominated solutions instead of a single solution, a higher computational effort is required to find high-quality solutions. Thus, we have designed our solution strategy with low-level search operators in mind with the purpose of develop an efficient and scalable solution approach that balances the concepts of exploration and exploitation in order to find high-quality and high-diversity non-dominated BITTP solutions.

In Algorithm 4, we present in detail the steps performed in our WSM for solving the BITTP. It starts (Line 1) by initializing the set that stores all non-dominated solutions found throughout the algorithm. By non-dominated solutions, we refer to solutions $\mathcal{S} \subseteq \mathcal{S}'$, from the set of solutions $\mathcal{S}'$ that our algorithm found, where

none of the solutions from $\mathcal{S}' \setminus \{\mathcal{S}\}$ dominates the solutions from $\mathcal{S}$. Our algorithm performs iterative cycles (Lines 2 to 23) while its stopping criterion is not achieved. At each iteration, we carry out exploration and exploitation mechanisms. During the exploration phase (Lines 3 to 7), our algorithm generates η feasible solutions for the BITTP as follows. Initially (Line 3), a tour π is generated by using the well-known Chained-Lin-Kernighan heuristic (Applegate et al., 2003).

Afterwards, we construct a feasible packing plan z at a time (Line 6) by using a randomized packing heuristic we have developed. Then, each packing plan z is combined with tour π in order to compose a feasible solution $\langle \pi, z \rangle$ for the BITTP, which is used to update the set of non-dominated solutions $\mathcal{S}$ (Line 7). The update of $\mathcal{S}$ is done in order to keep only non-dominated solutions in the set. Thus, if a solution $\langle \pi, z \rangle$ is dominated by any solution in $\mathcal{S}$, it is discarded. Otherwise, $\langle \pi, z \rangle$ is added in $\mathcal{S}$ and all solutions dominated by it are then removed. All the details of our packing heuristic strategy will be presented later in Algorithm 5. For now, we would like to only stress that each packing plan is constructed based upon the tour π and also on the real number α used to define the current weighted-sum objective function. Note that, in our algorithm, a value for α is randomly generated from a probability distribution $\mathcal{D}$ (Line 5). Thus, we can control and emphasize in which intervals of values α should be chosen by using different probability distributions.

The exploitation phase (Lines 8 to 22) begins by generating a new α value (Line 8) and selecting the best non-dominated solution $\langle \pi', z' \rangle$ in $\mathcal{S}$ according to the weighted-sum objective function formed from this new α. The solution $\langle \pi', z' \rangle$ is considered as a pivot for applying two local operators: 2-opt and bit-flip. Basically, a 2-opt move removes two non-adjacent edges and inserts two new edges by inverting two parts of the tour in such a way that a new tour is formed. In turn, a bit-flip move inverts the state of an item j in the packing plan z', i.e., if j is in z' then it is removed; otherwise, it is inserted if its inclusion does not exceed the knapsack capacity. These operators have been successfully incorporated to solve various combinatorial optimization problems, including the single-objective TTP (Faulkner et al., 2015; El Yafrani and Ahiod, 2016; Chand and Wagner, 2016; El Yafrani and Ahiod, 2018), and also the BITTP (Chagas et al., 2020b).

In our algorithm, first, we apply the operator 2-opt over the tour π' while the packing plan z' remains unchanged in order to find a faster tour that is still able to collect the same set of items. As the number of all tours Π (Line 10) obtained from 2-opt moves may be huge for some instances, it is impracticable to analyze them all.

In addition, significantly longer tours have less potential to be faster. For that reason, our algorithm has been restricted to analyze only those tours that are longer than π' up to a limited distance (Lines 13 to 15). The maximum tolerance for accepting a tour is given by the average of the distance ℓ among all pair of cities multiplied by a factor β (Line 14). After analyzing all selected tours, we chose the fastest tour π'', if any, among those that are faster than π', to compose a new solution, and then the set of non-dominated solutions $\mathcal{S}$ is updated from it (Line 16).

Afterwards, bit-flip operations are applied to the packing plan z' in order to find new packing plans that when combined with the tour π' produce new solutions. Because generating all bit-flip moves and evaluating all solutions formed from them may be impracticable for instances with many items, we decided that each bit-flip move is done according to a probability λ (Lines 17 to 22). The solutions generated from bit-flip moves are used to update, if applicable, the set of non-dominated solutions $\mathcal{S}$ (Line 22). At the end of the algorithm (Line 24), all non-dominated solutions found throughout its execution are returned.

4.2.2 A randomized packing strategy

In order to complete the description of the proposed WSM, we now present the strategy used to generate a packing plan from a given tour π. It is important to highlight that even for this scenario, the task of finding the optimal packing configuration remains $\mathcal{NP}$-hard (Polyakovskiy and Neumann, 2015), which makes it impractical for medium and large-size instances due to the time of computing required, especially because this procedure is a subroutine of our entire algorithm that is called many times. For this reason, our proposed strategy is a heuristic approach with the aim of quickly obtaining a packing plan from a tour. Before presenting its details, we would like to emphasize that our strategy is a non-deterministic packing algorithm, i.e., even for the same input parameters, it may exhibit different behaviors on different runs. Our design decision for that has been based on the fact that a non-deterministic mechanism introduces a more broadly exploration of the packing plan space, which may be effective to find regions with high-quality solutions.

Algorithm 5 describes all the steps of our packing heuristic strategy. It seeks to find a good packing plan z^{best} from multiple attempts for the same tour π. At each attempt (Line 3 to 19), a packing plan z is constructed. Due to the non-deterministic nature

Algorithm 4: Weighted-Sum Method - WSM($\mathcal{D}$, η, ρ, γ, β, λ)

1 $\mathcal{S} \leftarrow \varnothing$ `// set of non-dominated solutions`
2 **repeat**
`// exploration phase:`
3 $\pi \leftarrow$ solve the TSP component by the Chained-Lin-Kernighan heuristic♠
4 **for** $k \leftarrow 1$ **to** η **do**
5 $\alpha \leftarrow$ generate a random number from the probability distribution $\mathcal{D}$
6 $z \leftarrow$ RANDOMIZEDPACKINGALGORITHM(π, ρ, α, γ) `// Algorithm 5`
7 update $\mathcal{S}$ with the solution $\langle \pi, z \rangle$
`// exploitation phase:`
8 $\alpha \leftarrow$ generate a random number from the probability distribution $\mathcal{D}$
9 $\langle \pi', z' \rangle \leftarrow$ get from $\mathcal{S}$ the best solution according to α
10 let Π be the set of all 2-opt tours obtained from π'
11 let ℓ be the average of the distance among all pair of cities
12 $\pi'' \leftarrow \pi'$
13 **foreach** $\pi''' \in \Pi$ **do**
14 **if** $d(\pi''') - d(\pi') \leq \ell \times \beta$ **then**
15 **if** $f(\pi''', z', \alpha) > f(\pi'', z', \alpha)$ **then** $\pi'' \leftarrow \pi'''$
16 **if** $\pi'' \neq \pi'$ **then** update $\mathcal{S}$ with the solution $\langle \pi'', z' \rangle$
17 **foreach** item $j \in \{1, 2, \ldots, m\}$ **do**
18 **if** $rand(0, 1) \leq \lambda$ **then**
19 **if** $j \in z'$ **then**
20 update $\mathcal{S}$ with the solution $\langle \pi', z' \setminus \{j\} \rangle$
21 **else if** weight of $z' \cup \{j\}$ is lower than W **then**
22 update $\mathcal{S}$ with the solution $\langle \pi', z' \cup \{j\} \rangle$
23 **until** stopping condition is fulfilled
24 **return** $\mathcal{S}$

♠ http://www.math.uwaterloo.ca/tsp/concorde/downloads/downloads.htm

of our packing algorithm, multiple attempts increase the chance of finding a better packing plan. The number of attempts can be controlled by the parameter ρ (Line 2). Before any of these attempts (Line 1), z^{best} is defined with no items. Afterwards, at the beginning of each attempt, we uniformly select three random values (a, b, and c) between 0 and 1 (Line 3), and then normalize them (Line 4) so that their sum is equal to 1. These values are used to compute a score s_j for each item $j \in \{1, \ldots, m\}$ (Line 5),

Algorithm 5: RANDOMIZEDPACKINGALGORITHM(π, ρ, α, γ)

```
z^best ← ∅
for ρ' ← 1 to ρ do
    a ← rand(0,1), b ← rand(0,1), c ← rand(0,1)
    normalize a, b, and c so that their sum is equal to 1
    compute score for each item using a, b, and c according to Eq. (4.10)
    φ ← ⌈m/γ · α + ε⌉
    z ← z' ← ∅
    newPackingPlan ← false
    k ← k' ← 1
    while k' ≤ m and φ ≥ 1 do
        j ← get item with the k'-th largest score
        if weight of z' ∪ {j} is lower than W then
            z' ← z' ∪ {j}, newPackingPlan ← true
        if k' mod φ = 0 and newPackingPlan = true then
            if f(π, z', α) > f(π, z, α) then
                z ← z', k ← k'
            else z' ← z, k' ← k, φ ← ⌊φ/2⌋
            newPackingPlan ← false
        k' ← k' + 1
    if f(π, z, α) > f(π, z^best, α) then z^best ← z
return z^best
```

where a, b, and c define, respectively, exponents applied to profit p_j, weight w_j, and distance d_j in order to manage their impact. The distance d_j is calculated according to the tour π by summing all the distances from the city where item j is located to the final city of the tour. Equation 4.10 shows how the score of item j is calculated:

$$s_j = \frac{(p_j)^a}{(w_j)^b \cdot (d_j)^c} \tag{4.10}$$

From the foregoing equation, we can note that each score s_j incorporates a trade-off among a distance that item j has to be carried over, its weight, and also its profit. Equation 4.10 is based on the heuristic PACKITERATIVE that has been developed for the TTP (Faulkner et al., 2015). However, unlike these last authors, we have also considered an exponent for the term of distance to vary the importance of its influence. Furthermore, the values of all exponents are randomly selected drawn between 0 and 1, and then they are normalized in such a way that each of them establishes a

percentage of importance in the calculation of the score. After computing all scores, our algorithm uses their values to define the priority of each item in the packing strategy. The higher the score of an item, the higher its priority.

As described in the following, each packing plan z is constructed by selecting items iteratively according to their priorities. After including any item in z, it would be necessary to calculate the objective value of the solution $\langle \pi, z \rangle$ to be sure about its quality. However, since evaluating the objective function many times may be time-consuming, especially for large-size instances, we have introduced a parameter φ for controlling the frequency of the objective value re-computation. In other words, the objective value of the current solution $\langle \pi, z \rangle$ is only evaluated each time that φ items are analyzed. Initially (Line 6), φ is defined as $\lceil m/\gamma \cdot \alpha + \epsilon \rceil$, which depends on the number of items m, a parameter γ and the value α, and also a small value $\epsilon = 10^{-5}$ to avoid that φ assumes 0 when α is 0. Thus, the lower α, the lower φ and, consequently, the higher the frequency of the objective value re-computation. Note that for values close to zero, we look for solutions with faster tours, which requires a packing plan without or with few items. Therefore, for this scenario, a high frequency of re-computation of the objective function is needed in order to select many items without checking whether they improve the quality of the solution.

Each packing plan z is constructed as follows. At first, z and an auxiliary packing plan z' are both defined as empty sets (Line 7). Other auxiliary variables are used to control if there is a new packing plan to be evaluated (Line 8) and also to management which item is currently being analyzed (Line 9). The iterative packing construction process of our algorithm (Lines 10 to 19) start by selecting the item j with the k'-th largest score (Line 11). If the addition of item j does not exceed the knapsack capacity (Line 12), then j is inserted into packing plan z', and it is marked that there is a new packing configuration (Line 13). Every time that φ items have been considered and that the current packing plan z' has not been evaluated (Line 14), we compute the objective function of the solution $\langle \pi, z' \rangle$ and confront its quality against quality of the solution $\langle \pi, z \rangle$. If the solution $\langle \pi, z' \rangle$ is better (Line 15), φ remains the same and z is updated to z'. Otherwise (Line 17), the packing plan z' is updated to z and the algorithm returns to consider the items again starting with the item whose score is the k-th largest (Line 17). In addition, φ is halved in order to provide the chance to improve the solution by collecting fewer items before an evaluation. Each construction of a packing plan terminates either when there is no more items to collect or because no further improvement is possible following our strategy. After completing the

construction of each packing plan z, the best solution $\langle \pi, z^{best} \rangle$ found so far is updated to the solution $\langle \pi, z \rangle$ if it is to improve (Line 20). At the end of the algorithm (Line 21), the packing plan of the best solution found is returned.

4.3 Computational experiments

In this section, we present the experiments performed to study the performance of the proposed algorithm. First, we have conducted an extensive comparison with the algorithm proposed by Chagas et al. (2020b). In addition, we compare our results with those submitted to BITTP competitions, which have been held in 2019 at the *Evolutionary Multi-Criterion Optimization* (EMO2019) and *The Genetic and Evolutionary Computation Conference* (GECCO2019). Lastly, we contrast our results with the single TTP objective scores obtained from efficient algorithms already proposed in the literature for the TTP.

Our algorithm has been implemented in Java. Each run of it has been sequentially (nonparallel) performed on a machine with Intel(R) Xeon(R) CPU X5650 @ 2.67GHz and Java 8, running under CentOS 7.4. Our code, as well as all numerical results can be found at https://github.com/jonatasbcchagas/wsm_bittp.

4.3.1 Benchmarking instances

To assess the quality of the proposed WSM, we have used instances of the comprehensive set of TTP instances defined by Polyakovskiy et al. (2014). These authors have created 9720 instances in such a way that the two components of the problem have been balanced so that the near-optimal solution of one sub-problem does not dominate over the optimal solution of another sub-problem. For a complete and detailed description of how these instances have been created, we refer the interested reader to (Polyakovskiy et al., 2014) and also to (Wagner et al., 2018), which presents a study on the instance features. In our experiments, we have used a subset of the 9720 TTP instances with the following characteristics:

- numbers of cities: 51, 152, 280, 1000, 4461, 13509, 33810, and 85900 (the layout of cities is given according to the TSP instances Reinelt (1991) *eil51*, *pr152*, *a280*, *dsj1000*, *usa13509*, *pla33810*, and *pla85900*, respectively);

- numbers of items per city: *01*, *03*, *05*, and *10* (all cities of a single TTP instance have the same number of items, except for the city in which the thief starts and ends their journey, where no items are available);
- types of knapsacks: weights and values of the items are bounded and strongly correlated (*bsc*), uncorrelated with similar weights (*usw*), uncorrelated (*unc*);
- sizes of knapsacks: *01*, *02*, . . ., *09* and *10* times the size of the smallest knapsack, which is defined by summing the weight of all items and dividing the sum by 11;

By combining all the different characteristics described above, we have 960 instances that compose a broad and diverse sample of all 9720 instances. In the remainder of this chapter, each instance will be identified as `XXX_YY_ZZZ_WW`, where `XXX`, `YY`, `ZZZ`, and `WW` indicate the different characteristics of the instance at hand. For example, `a280_03_bsc_01` identifies the instance with 280 cities (TSP instance *a280*), 3 items per city with their weights and values bounded and strongly correlated with each other, and the smallest knapsack defined.

4.3.2 Parameter tuning

In order to find suitable configuration values for the algorithm's parameters among all possible ones, we have used the Irace package (López-Ibáñez et al., 2016b), which is an implementation of the method I/F-Race (Birattari et al., 2010). The Irace package implements an iterated racing framework for the automatic configuration of algorithms, which has been used frequently due to its simplicity to use and its performance.

Table 5.1 shows the parameter values of our algorithm we have considered in the Irace tuning. These values have been selected following preliminary experiments. Note that for $\beta = -\infty$ and $\lambda = 0$, our algorithm does not perform, respectively, any 2-opt and bit-flip moves. Regarding the stopping criterion of the algorithm, we have set its runtime to 10 minutes. This choice is very often used in TTP research, thus following a pattern already established that allows fairer comparisons among different solution approaches. In addition, as stated by Wagner et al. (2018), this computation budget limit is motivated by a real-world scenario, where a 10-minutes break is enough for a decision-maker, who is interested in what-if analyses, to have a cup of coffee. After this time, the decision-maker analyses the computed results, and then he/she can make the possible next changes to the system to investigate other alternatives.

Table 4.1: Parameter values considered during the tuning experiments.

Parameter	Tested values
$\mathcal{D}$	$\mathcal{U}(0,1)$, $\mathcal{N}(0.5,0.2)$, $\mathcal{B}(3,1.5)$, $\mathcal{B}(1.5,3)$
η	$1,2,\ldots,200$
ρ	$1,2,\ldots,100$
γ	$1,2,\ldots,200$
β	$-\infty, 0, 0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1, 10, 100$
λ	$0, 0.01, 0.02, \ldots, 0.5$

For WSM, to generate α values, we have chosen probability distributions in such a way that some ideas could be tested (Figure 4.1). Firstly, the most natural idea is to use a uniform distribution $\mathcal{U}(0,1)$ that generates values between 0 and 1 with the same probability. Using a normal distribution with mean 0.5 and standard deviation 0.2, denoted as $\mathcal{N}(0.5,0.2)$, we lay emphasis on generating values close to 0.5 in order to focus on weighted-sum objective functions equivalent to the original TTP objective function. Note that the closer to 0.5 the value is, the greater is the interaction between the two components of the problem, and perhaps we should concentrate the algorithm's efforts on these values. On the other hand, maybe we should focus on values close to 0 or 1 when it is the case that one of the components is more easily solved. For example, note that for α values closer to 0, we are looking for TTP solutions with good TSP components (few or no items should be stolen). As we are using the Chained-Lin-Kernighan heuristic, one of the most efficient algorithms for generating near-optimal TSP solutions (Wu et al., 2018), our algorithm might not need to exploit these values much to find good TTP solutions concerning good TSP component. Thus, we can use, for example, a beta distribution $\mathcal{B}(3,1.5)$ that does not generate many values close to 0. In addition, we have also considered in our experiments a beta distribution $\mathcal{B}(1.5,3)$ with their parameters swapped concerning the previous distribution to address scenarios where the Chained-Lin-Kernighan heuristic combined with our packing algorithm is able to find good TTP solutions with a high collected profit without the need for a high emphasis on α values close to 1. For a reference on probability distributions, we refer to Krishnamoorthy (2016).

To ensure better performance of the proposed algorithm, we have analyzed the influence of its parameters on different types of instances. More precisely, we have divided all 960 instances into 96 groups and then execute Irace on each of them. Each

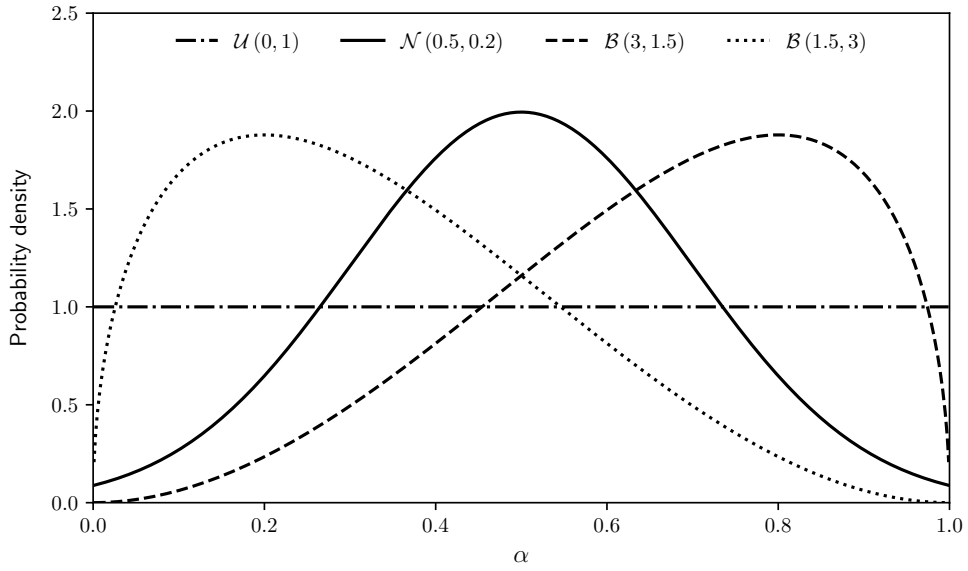


Figure 4.1: Different probability distributions to generate α values.

group contains all 10 instances defined with different sizes of knapsacks. These groups are identified as `XXX_YY_ZZZ`, in the same way as we have identified the instances, except for the lack of `WW`. With this approach, we would like to know whether there exist similar behaviors among the best parameter configurations from different groups of instances. As we have selected 96 groups with a large difference in characteristics among them, it is reasonable to think that whether such behaviors exist, they may also apply to unknown instances.

As Irace evaluates the quality of the output of a parameter configuration using a single numerical value, we should use for multi-objective problems some unary quality measure (López-Ibáñez et al., 2016b), such as the hypervolume indicator or the ϵ-measure (Zitzler et al., 2003). In our experiments, we have used the hypervolume indicator. In addition, we have used all Irace default settings, except for the parameter *maxExperiments*, which has been set to 1000. This parameter defines the stopping criteria of the tuning process. We refer the readers to (López-Ibáñez et al., 2016a) for a complete user guide of Irace package.

From the tuning experiments, we have obtained the results shown in Figure 5.2. Each parallel coordinate plot lists for each of the 96 groups (listed in the left-most column) the configurations returned by Irace (plotted in the other columns). As Irace can return more than one configuration that are statistically indistinguishable given the threshold of the statistical test, multiple configurations are sometimes shown. Each vertical axis indicates a parameter and its range of values, and each configuration of

parameters is described by a line that cuts each parallel axis in its corresponding value. Through the concentration of the lines, we can see which parameter values have been most selected among all tuning experiments. We have used different colors and styles for lines in order to emphasize the results obtained for each group individually. All logs generated by the Irace executions, as well as their settings can be found at the GitHub link along with our code.

We can make several observations from the tuning results. First, we notice that for almost all groups of instances the uniform distribution $\mathcal{U}(0, 1)$ has been chosen. For some groups, especially those that contain larger instances, other distributions have been returned by Irace. Regarding the parameter η, we can observe a strong trend in increasing its value as the number of cities increases. This is not too surprising, as the Chained-Lin-Kernighan heuristic, in general, requires more computational time to address larger TSP instances. Thus, computing a higher number of packing plans from each tour may generate better BITTP solutions than resolve the TSP component many times. We can also observe from the values obtained for the parameter ρ that only a few attempts of our packing strategy are needed to reach good results, which is especially true for larger instances. The low values obtained for the parameter γ for most groups of instances indicate that the frequency of re-computation of the objective function in the packing algorithm may begin with low values without interfering in the quality of the packing plan computed. Although the values of the parameter β do not follow a clear trend, they are strongly related to the number of cities and mainly to the layout that the cities are arranged. For example, when many cities are uniformly arranged, the trend is towards low β values as, for this scenario, higher β values would probably not be efficient, since the algorithm would spend most of the time processing too many tours obtained from 2-opt moves. Finally, we can see that, in general, higher λ values are concentrated in smaller-size instances, which is not surprising since the bit-flip operator would perform too many moves on instances with many items and higher λ values.

With a closer look, we can make additional observations by combining different parameters and characteristics of the instances. For example, η values are low for medium and large knapsack capacities (red and yellow) of eil51, while the opposite is true for the dsj1000 instances, and other large instances. Across almost all instances, the η values are the lowest or among the lowest for instances with uncorrelated (dotted) knapsacks. For ρ, it is difficult to extract patterns, however, we can observe that the tuned configurations for instances with strongly correlated knapsacks (dashed) have

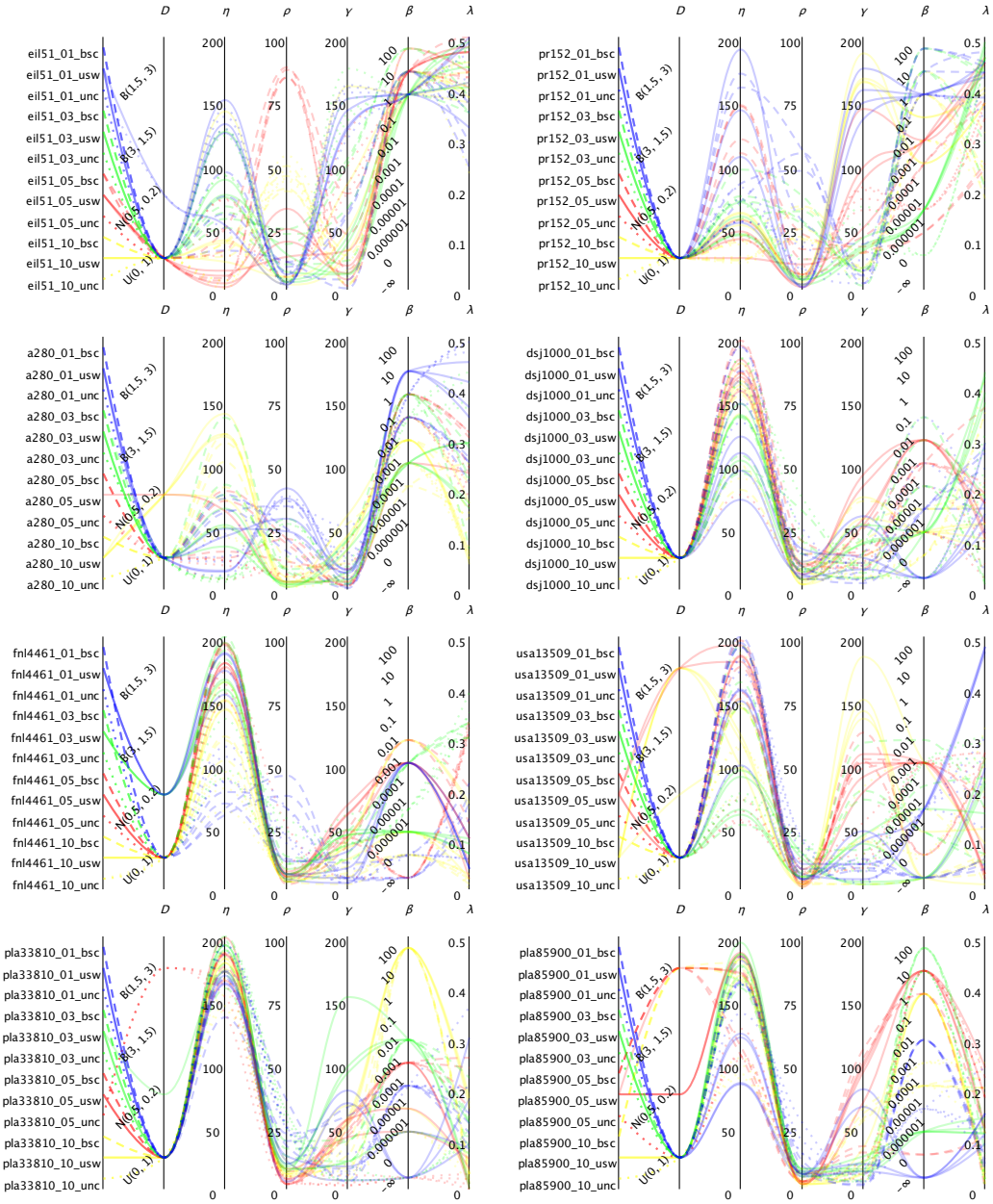


Figure 4.2: Irace results for the 96 groups of instances. Blue, green, red, and yellow lines represent, respectively, groups of instances with 1, 3, 5, and 10 items-per-city. Dashed, solid, and dotted lines are used, respectively, to emphasize the groups of instances with items where their weights and values are bounded and strongly correlated (*bsc*), uncorrelated with similar weights (*usw*), and uncorrelated (*unc*).

the highest ρ values for the groups eil51 (red), pr152 (blue), and fnl4661 (blue). For γ, small knapsacks (blue) with uncorrelated but similar weights (solid) result in high or the highest values for eil51, pr152, and pla33810, but for example not for a280. For

β, we cannot observe clear trends for the knapsack type, however, sometimes the knapsack capacity stands out. For example, for a280, the smallest knapsacks (blue) resulted in the highest values, while blue has the lowest values for dsj1000, and the largest knapsacks resulted in the highest values (yellow) for the tuning experiments for the pla33810 group. We can observe similar 'inversions' also for γ. There, for example the smallest knapsacks with uncorrelated and similar weights (blue, solid) result in the smallest values on some instance groups, but for the largest values on others.

In summary, we can observe many consistent as well as inconsistent patterns for the different groups of instances, and depending on the knapsack type and the knapsack capacity. In combination with instance features (e.g. the ones from Wagner et al. (2018)), this might make for an interesting challenge for per-instance-algorithm-configuration (Hutter et al., 2006), however, this is beyond the scope of the present study.

4.3.3 WSM *vs.* NDSBRKGA

In the first analysis that assesses the quality of our WSM, we compare the solutions obtained by it with the solutions obtained by NDSBRKGA proposed by Chagas et al. (2020b). In order to make a fair comparison, we have tuned the parameters of the NDSBRKGA following the same procedure used in the tuning of the parameters of WSM. The parameter values considered for these experiments have been chosen based on the insights reported in (Chagas et al., 2020b). These parameter values, as well as the results obtained in the experiment are available at the GitHub link along with our other files.[5]

Due to the randomized nature of both algorithms, we have performed 30 independent repetitions on each instance. Each run has been executed for 10 minutes with the best parameter values found in the tuning experiments.

As in (Chagas et al., 2020b), we have used the hypervolume indicator (HV) (Zitzler and Thiele, 1998) as a performance indicator to compare and analyze the results

[5] As neither the HPI algorithm nor the HPI implementation are available, we could not include HPI in this tuning-based comparison. HPI has been the result of a classroom setting and their actual results have been (to some extent) aggregated across multiple teams (and hence implementations), which has been legal w.r.t. the competition rules (as said competition required only the solution files, not any implementations).

obtained. This indicator is one of the most used indicators for measuring the quality of a set of non-dominated solutions by calculating the volume of the dominated portion of the objective space bounded from a reference point. To make the hypervolume suitable for the comparison of objectives with greatly varying ranges, a normalization of objective values is commonly done beforehand. Therefore, before computing the hypervolume, we have first normalized the objective values between 0 and 1 according to their minimum and maximum value found during our experiments. Although maximizing the hypervolume might not be equivalent to finding the optimal approximation to the Pareto-optimal front (Bringmann and Friedrich, 2013; Wagner et al., 2015), we have assumed that the higher the hypervolume indicator, the better the solution sets are, as is commonly considered in the literature.

We compare the performance of the solutions obtained by measuring for each instance the percentage variation of the average hypervolume obtained considering the independent runs of each algorithm. More precisely, for each instance, we have estimated the reference point as the maximum travel time and the minimum profit obtained from the non-dominated solutions, which have been extracted from all solutions returned by the algorithms. Then, we have computed the hypervolume covered by the non-dominated solutions found by each run of each algorithm according to the estimated reference point. Thereafter, we can compute the percentage variation as

$$\left(\mathrm{HV}_{\mathrm{avg}}^{\mathrm{WSM}} - \mathrm{HV}_{\mathrm{avg}}^{\mathrm{NDSBRKGA}}\right) \,/\, \max\left(\mathrm{HV}_{\mathrm{avg}}^{\mathrm{WSM}}, \mathrm{HV}_{\mathrm{avg}}^{\mathrm{NDSBRKGA}}\right) \cdot 100\%$$

, where $\mathrm{HV}_{\mathrm{avg}}^{\mathrm{WSM}}$ and $\mathrm{HV}_{\mathrm{avg}}^{\mathrm{NDSBRKGA}}$ are, respectively, the average hypervolumes obtained by WSM and NDSBRKGA in their independent executions.

In Figure 4.3, we visualise the percentage variations of the average hypervolumes using a heatmap to emphasize larger variations. Each cell of the heatmap informs the results obtained for a specific instance. Note that the vertical axis depicts the characteristics `XXX` and `YY` of instances, while the horizontal axis depicts the characteristics `ZZZ` and `WW`. Note also that positive variation values (highlighted in shades of orange and red) indicate that our WSM has reached a higher hypervolume, while negative variation values (highlighted in shades of blue) indicate the opposite behavior. Besides, the higher the absolute value (more intense color), the higher the difference between the hypervolumes.

From Figure 4.3, we can observe that our WSM is clearly more effective than NDSBRKGA for larger instances. This is especially true for instances with the smallest

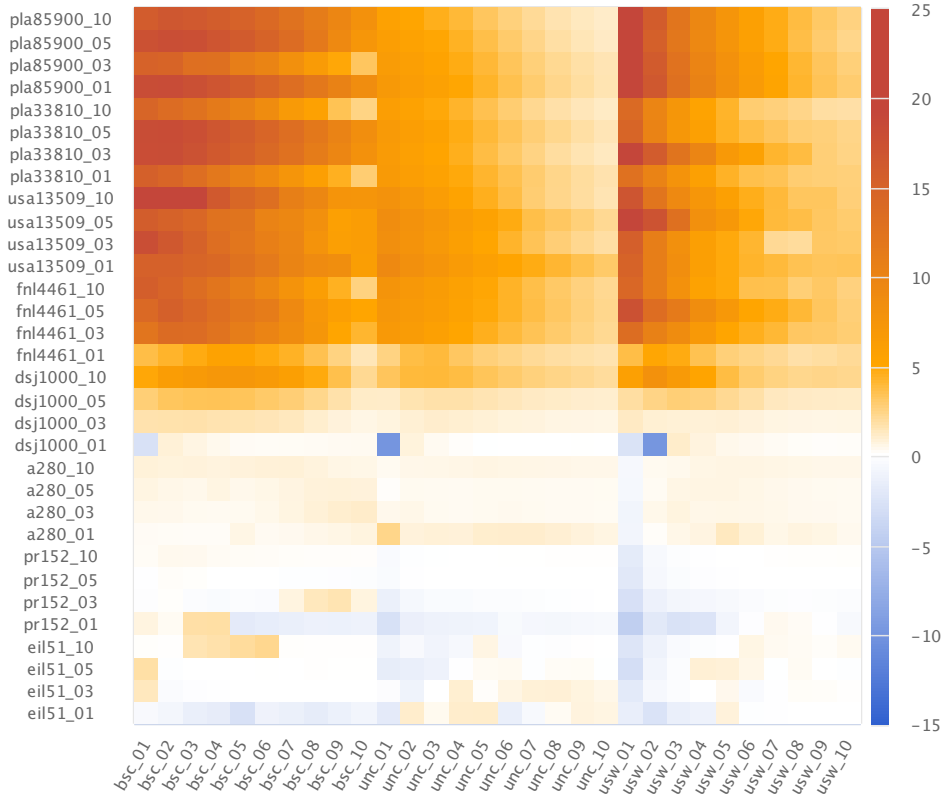


Figure 4.3: Percentage variation of the average hypervolumes. Shades of orange and red indicate in which instances our WSM has reached a higher hypervolume than NDSBRKGA, while shades of blue indicate the opposite.

knapsack capacities. Note that, in general, the smaller the size is of the knapsack, the higher the performance is of WSM concerning the NDSBRKGA.

Note that, although our solutions still cover a higher hypervolume for larger instances with uncorrelated (*unc*) items, it must be stressed that our WSM has obtained the worse performance regarding the NDSBRKGA for these instances. This behavior could be explained by the fact that our packing heuristic might present difficulties in dealing with those items because when there is no correlation between their profits and weights and the weights present a large variety, our packing algorithm may not be able to create a good order of the items for our packing strategy. This is interesting, as *unc* knapsacks are not necessarily seen as difficult (Martello et al., 1999); but in our algorithm, they might end up being due to our strategy for solving the KP component. Another fact that could explain the worse performance of WSM on instances with uncorrelated items would be that NDSBRKGA has a good performance for these instances, making the performance of our algorithm less prominent.

Regarding the smaller-size instances, both algorithms have achieved similar performance (almost blank cells). However, with a closer look at Figure 4.3, we can see a slightly better performance of NDSBRKGA. To better analyze these results, we have used another performance measure. For each instance we have merged all the solutions found in order to extract from them a single non-dominated set of solutions. Then, we have computed how many non-dominated solutions have been obtained by each algorithm. Our purpose of this analysis is to evaluate both algorithms regarding their ability to find non-dominated solutions with different objective values. Therefore, duplicate solutions regarding their objective values have been removed, i.e., we have regarded a single non-dominated solution with the same values in both objectives. In Figure 4.4, we present these numbers in percentages according to the total number of non-dominated solutions following the heatmap scheme used previously.

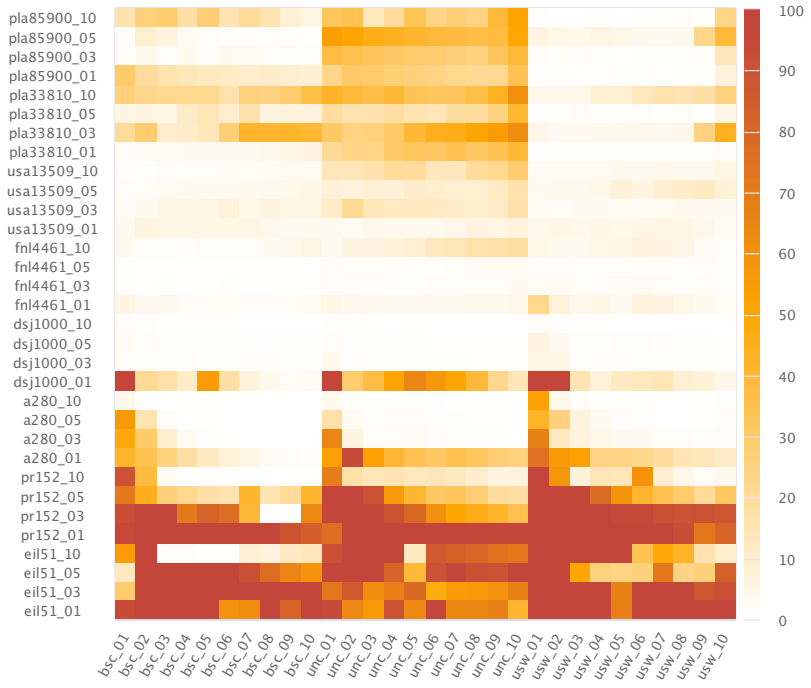
The results shown in Figure 4.4 corroborate those shown in Figure 4.3. As was expected, our algorithm has found more non-dominated solutions especially for those instances where it obtained a higher hypervolume. However, even for the instances in which the NDSBRKGA found better solutions, the difference between the hypervolumes of both algorithms remains low.

To statistically compare the performance of the algorithms, we have used the Wilcoxon signed-rank test on the hypervolumes achieved in the 30 independent runs. With a significance level of 5%, there is no statistical difference between both algorithms in 27 instances (2.8%), our algorithm is significantly better in 789 instances (82.2%) and worse in 144 (15%) ones when compared to NDSBRKGA.

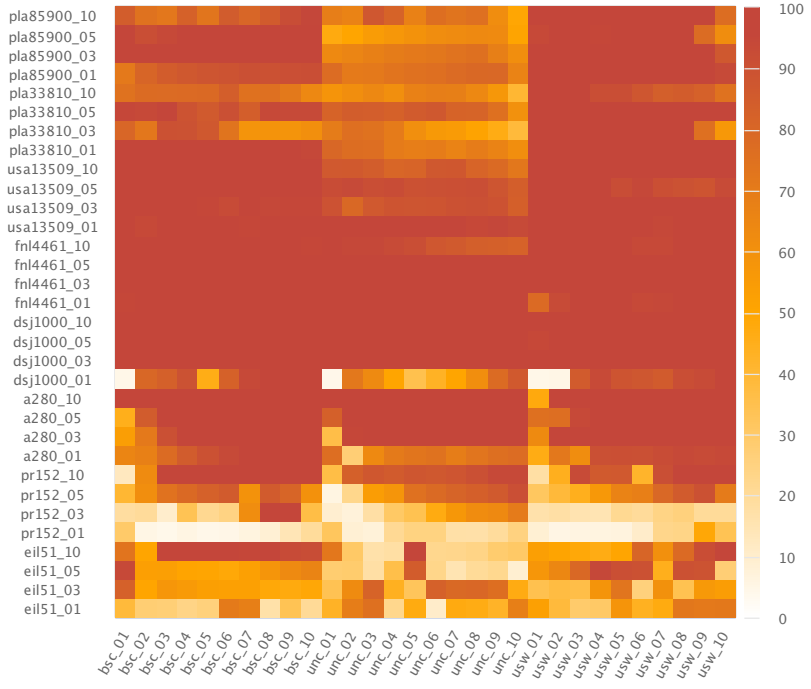
4.3.4 WSM *vs.* competition results

Next, we compare WSM to the results of the BITTP competitions held at EMO2019[6] and GECCO2019[7]. Both competitions have used the same rules and criteria. There were no regulations regarding the running time and the number of processors used. The final ranking used for the competitions was solely based on the solution set submitted by each participant for nine medium/large TTP instances chosen from the TTP benchmark (Polyakovskiy et al., 2014). More precisely, the final ranking was defined according to the hypervolume covered by the solutions. To calculate the

[6] https://www.egr.msu.edu/coinlab/blankjul/emo19-thief/
[7] https://www.egr.msu.edu/coinlab/blankjul/gecco19-thief/



(a) NDSBRKGA



(b) WSM

Figure 4.4: Percentage of non-dominated solutions found by each algorithm.

hypervolumes, the reference points have been defined as the maximum time and the minimum profit obtained from the non-dominated solutions, which have been built

from all submitted solutions. In order to make a fair ranking, the maximum number of solutions allowed for each instance has been limited. In Table 4.2, we list the instances used as well as the maximum number of solutions allowed.

Table 4.2: Maximum number of solutions allowed by each TTP instance used in the BITTP competitions.

Instance	Maximum number of solutions allowed
a280_01_bsc_01	100
a280_05_usw_05	100
a280_10_unc_10	100
fnl4461_01_bsc_01	50
fnl4461_05_usw_05	50
fnl4461_10_unc_10	50
pla33810_01_bsc_01	20
pla33810_05_usw_05	20
pla33810_10_unc_10	20

As our algorithm can return a higher number of solutions than those reported in Table 4.2, we have used the dynamic programming algorithm developed by Auger et al. (2009) in order to find a subset of limited size of the returned solutions such that their hypervolume indicator is maximal. As stated by Auger et al. (2009), this dynamic programming can be solved in time $\mathcal{O}(|A|^3)$, where A would be the set of solutions returned by our algorithm. Note that the application of this strategy has also been used in (Chagas et al., 2020b) for NDSBRKGA, and it is only part of a post-processing needed to fit both algorithms to the competition criteria.

In both competitions, preliminary versions of NDSBRKGA have been submitted as *jomar*, a reference to the two authors (**Jo**natas and **Mar**cone) who first worked on that algorithm. These preliminary versions are presented in (Chagas et al., 2020b) as well as their results achieved in both competitions. In short, *jomar* has won the first and second places, at EMO2019 and GECCO2019 competitions, respectively. After the competitions, some improvements have been incorporated in the preliminary versions of NDSBRKGA, resulting in its final version as is described in (Chagas et al., 2020b). In the following, we compare our WSM with that final version as it presents slightly better results concerning its previous ones.

In Table 4.3, we present for each instance the best results submitted for the competitions and also the results obtained by our WSM. The results of all submissions can be found at web pages previously reported. As the final results of NDSBRKGA have been obtained with 5 hours of processing, we have executed our algorithm for 5 hours as well to make a fair comparison. We would like to mention that we have no information on how the other participants have obtained their results. As we stated before, there were no regulations regarding the running time and the number of processors used. In both competitions, their rankings have been solely based on the solution set submitted by each participant. Furthermore, to the best of our knowledge, there is no description available of the solution approaches submitted.

From Table 4.3, we can notice that WSM has obtained better performance on large-size instances. For the three smallest instances, it has presented the worst results concerning the other results, especially, for those reached by the first (HPI) and second (NDSBRKGA) places at GECCO2019 competition. For the other instances, our results have surpassed all other submissions with a larger difference compared to NDSBRKGA.

If our algorithm had been submitted, it would win first place with 21 points against 17 points obtained by HPI and 14 by NDSBRKGA. These final scores would be computed according to the ranking criteria: after sorting all submissions for each instance according to the hypervolume achieved in decreasing order, the 1st place takes 3 points, 2nd place takes 2 points, 3rd place a single point. According to this final scoring criterion, we classify our algorithm when it runs for the least amount of time. In the following, we present how the final ranking would be, considering different runtimes for our WSM. We compare the results obtained by WSM in each runtime limitation against the final results obtained by HPI and NDSBRKGA:

- ♢ 10 minutes: HPI 22 points | **WSM 15 points** | NDSBRKGA 15 points
- ♢ 20 minutes: HPI 22 points | **WSM 15 points** | NDSBRKGA 15 points
- ♢ 30 minutes: HPI 20 points | **WSM 17 points** | NDSBRKGA 15 points
- ♢ 1 hour: **WSM 19 points** | HPI 19 points | NDSBRKGA 14 points
- ♢ 2 hours: **WSM 19 points** | HPI 19 points | NDSBRKGA 14 points
- ♢ 3 hours: **WSM 20 points** | HPI 18 points | NDSBRKGA 14 points

Table 4.3: Best BITTP competitions results *vs.* WSM.

Instance	Participant/Algorithm	HV
a280_01_bsc_01	HPI	0.898433
	NDSBRKGA	0.895708
	WSM	**0.887205**
	shisunzhang	0.886576
a280_05_usw_05	NDSBRKGA	0.826879
	HPI	0.825913
	shisunzhang	0.820893
	WSM	**0.820216**
a280_10_unc_10	NDSBRKGA	0.887945
	WSM	**0.887680**
	HPI	0.887571
	ALLAOUI	0.885144
fnl4461_01_bsc_01	**WSM**	**0.934685**
	NDSBRKGA	0.933942
	HPI	0.933901
	NTGA	0.914043
fnl4461_05_usw_05	**WSM**	**0.820481**
	HPI	0.818938
	NDSBRKGA	0.814492
	NTGA	0.803470
fnl4461_10_unc_10	**WSM**	**0.882932**
	HPI	0.882894
	NDSBRKGA	0.874688
	SSteam	0.856863
pla33810_01_bsc_01	**WSM**	**0.930580**
	HPI	0.927214
	NTGA	0.888680
	ALLAOUI	0.873717
pla33810_05_usw_05	**WSM**	**0.819743**
	HPI	0.818259
	NDSBRKGA	0.781009
	SSteam	0.776638
pla33810_10_unc_10	**WSM**	**0.876805**
	HPI	0.876129
	NDSBRKGA	0.857105
	SSteam	0.853805

ALLAOUI is formed by Mohcin Allaoui and Belaid Ahiod; **HPI** is formed by Tobias Friedrich, Philipp Fischbeck, Lukas Behrendt, Freya Behrens, Rachel Brabender, Markus Brand, Erik Brendel, Tim Cech, Wilhelm Friedemann, Hans Gawendowicz, Merlin de la Haye, Pius Ladenburger, Julius Lischeid, Alexander Löser, Marcus Pappik, Jannik Peters, Fabian Pottbäcker, David Stangl, Daniel Stephan, Michael Vaichenker, Anton Weltzien, and Marcus Wilhelm; **NTGA** is formed by Maciej Laszczyk and Pawel Myszkowski; **shisunzhang** is formed by Jialong Shi, Jianyong Sun, and Qingfu Zhang; and **SSteam** is formed by Roberto Santana, and Siddhartha Shakya.

One can note that with 10 and 20 minutes of processing time, we would be in 2nd place tied with NDSBRKGA. With exactly 30 minutes, we would occupy the 2nd position alone. With one and two hours, we would share the first position with the HPI team. Then, after 3 hours of processing, we would occupy the 1st place alone.

4.3.5 Dispersed distribution of the non-dominated solutions

We now analyze the dispersion over the objective spaces of the solutions found by our algorithm. As we have stated before, a limitation of WSMs is the fact that, even with a consistent change in weights attributed to the objectives, they may not generate a dispersed distribution of non-dominated solutions found. This limitation does not affect our WSM, as it can be seen in Figure 4.5, where we have plotted the objective values of all non-dominated solutions found by WSM with 10 minutes of runtime for the nine medium/large-size instances used in the aforementioned BITTP competitions. In addition, we have highlighted which α has been used when finding each solution. One can notice dispersed distributions of the solutions as well as the α values. Moreover, as expected, lower α values produce solutions with faster tours, with higher ones produce solutions with good packing plans.

4.3.6 Single-objective comparison

Since BITTP is a bi-objective formulation created from the TTP without introducing any new specification or removing any original constraint, any feasible BITTP solution is also feasible for the TTP. Thus, we can measure the performance of the solutions obtained by our algorithm according to their single-objective TTP scores. However, it is important to emphasize that our algorithm has not been developed with a single-objective purpose. Therefore, we should be careful when comparing it with other algorithms for the TTP.

A fairer comparison can be achieved between our results and those reached by NDSBRKGA, as both approaches have been developed with the same ambition. For this purpose, we have calculated for each instance the Relative Percentage Difference (RPD) between the best TTP scores achieved by WSM and NDSBRKGA, referenced as S_{best}^{WSM} and $S_{best}^{NDSBRKGA}$, respectively. It is important to emphasize that no additional tests have been performed, we only choose the solution with the best TTP score among

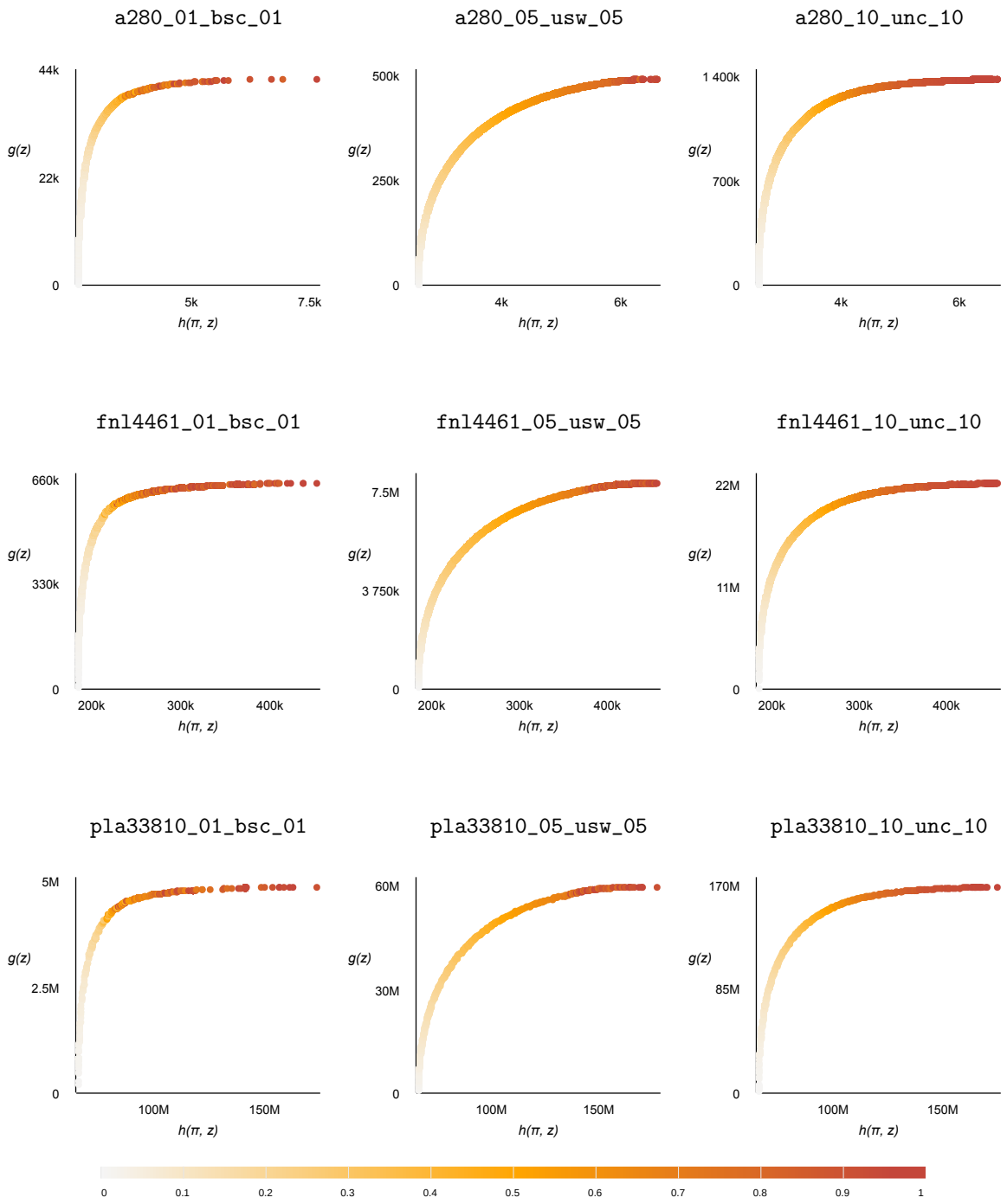


Figure 4.5: Non-dominated points found by WSM. Colors indicate the α values used when finding each point.

the non-dominated solutions found by each algorithm on each instance. The RPD metric has been calculated as

$$\left(S_{\text{best}}^{\text{WSM}} - S_{\text{best}}^{\text{NDSBRKGA}}\right) \; / \; \left|S_{\text{best}}^{\text{NDSBRKGA}}\right| \cdot 100\%$$

, and we plot its values using a heatmap in order to highlight higher differences as depicted in Figure 4.6. Note that positive values (highlighted in shades of orange and red) indicate that our WSM has found higher TTP scores.

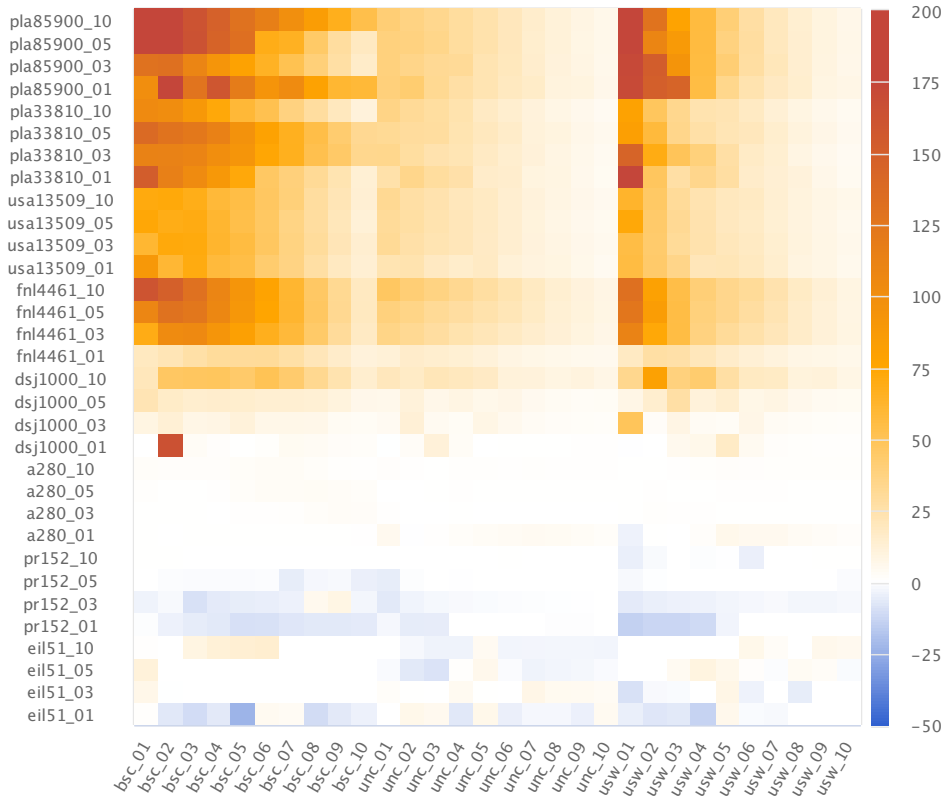


Figure 4.6: WSM *vs.* NDSBRKGA according to their obtained single-objective TTP scores. Shades of orange and red indicate in which instances our WSM has reached better single-objective TTP scores than NDSBRKGA, while shades of blue indicate the opposite.

We can note that the heatmap show in Figure 4.6 has characteristics similar to those in Figure 4.4b, where higher percentages of the number of non-dominated solutions found by our algorithm are highlighted. Therefore, this behavior is not surprising, since dominated solutions have essentially lower TTP scores compared to the non-dominated solutions. Thus, we can confirm a better efficiency of WSM also concerning the TTP scores for larger instances, while its worst performance on smaller-size instances is less expressive.

Although it may not be fair, as we stated earlier, we conclude our analysis by comparing the best TTP scores obtained by WSM with the best single TTP objective scores reported in (Wagner et al., 2018), where the authors have made a comprehensive comparison of 21 algorithms proposed for the TTP over the years. In this comparison,

we use again the RPD metric, which now is calculated for each instance as

$$\left(S_{best}^{WSM} - S_{best}^{21ALGS}\right) \; / \; \left|S_{best}^{21ALGS}\right| \cdot 100\%$$

, where S_{best}^{21ALGS} indicates the best TTP score found among all 21 algorithms analyzed in (Wagner et al., 2018). In Figure 4.7, we plot the calculated RPD values following the same visualization scheme adopted previously. In addition, we highlight with a diamond symbol the instances for which our algorithm has found better solutions.

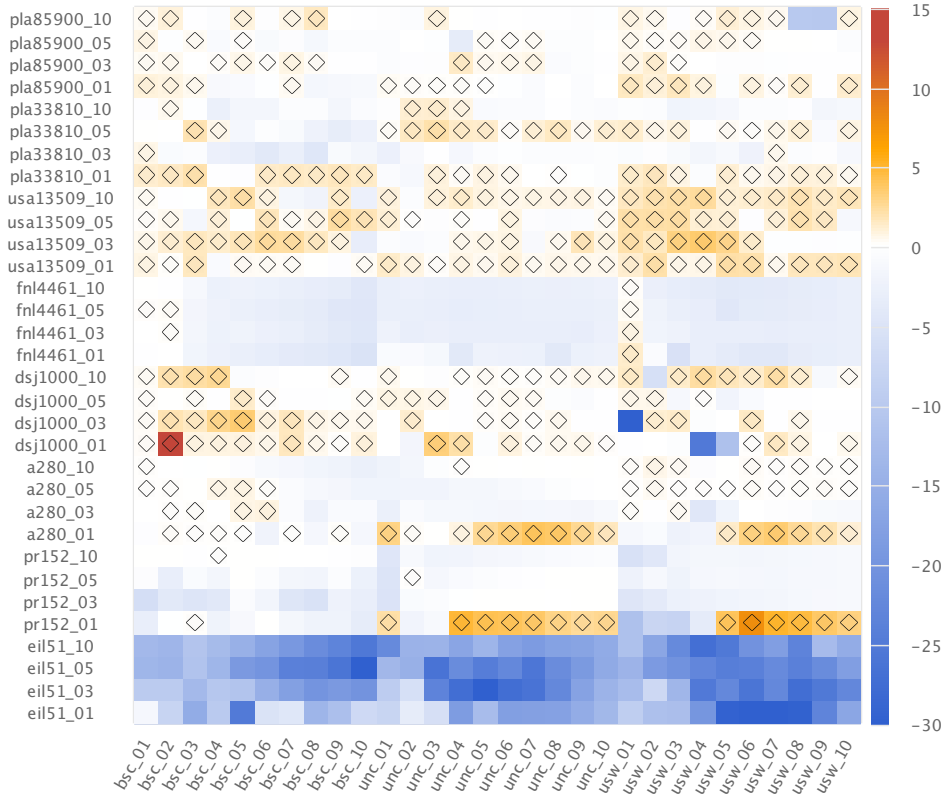


Figure 4.7: WSM *vs.* TTP algorithms according to their obtained single-objective TTP scores. Shades of orange and red indicate in which instances our WSM has reached better single-objective TTP scores than the best algorithm among 21 ones reported in (Wagner et al., 2018), while shades of blue indicate the opposite. Diamond symbols highlight in which the instances our WSM has found better results.

One can note that, in general, our results presented worse performance, which is especially true for the smaller-size instances. However, for many instances our results have outperformed all 21 TTP algorithms. This shows that our WSM can also be competitive to solve the TTP.

4.4 Conclusions

In this work, we have addressed a bi-objective formulation of the Traveling Thief Problem (TTP), an academic multi-component problem that combines two classic combinatorial optimization problems: the Traveling Salesperson Problem and the Knapsack Problem. For solving the problem, we have proposed a heuristic algorithm based on the well-known weighted-sum method, in which the objective functions are summed up with varying weights and then the problem is optimized in relation to the single-objective function formed by this sum. Our algorithm combines exploration and exploitation search procedures by using efficient operators, as well as known strategies for the single-objective TTP; among these are deterministic strategies that we have randomized here. We have studied the effects of our algorithmic components by performing extensive tuning of their parameters over different groups of instances. This tuning also shows that different configurations are needed depending on the instance group, the knapsack type, and the knapsack capacity. Our comparison with multi-objective approaches shows that we could have won the recent optimization competitions, and we have furthermore found new best solutions for the single-objective case along the way.

For future research, we would like to point out as a promising direction the investigation of the influence of different algorithmic components already proposed in the literature over different instance characteristics by investigating tuned configurations. Studies in this data-driven direction have achieved important insights to design better single-objective solvers for fundamental problems and real-world problems (see, e.g. Section "Research Directions" of Agrawal et al. (2020)). Another interesting direction would be to use our algorithm core idea for solving other multi-objective problems with multiple interacting components. By core idea, we refer to how to explore and exploit the space of solutions once efficient operators and strategies are known for solving different components of a multi-objective problem.

Chapter 5

The thief orienteering problem

In this chapter[1], we tackle the Thief Orienteering Problem (ThOP) (Santos and Chagas, 2018), a academic multi-component problem that was proposed based on the Traveling Thief Problem (TTP) (see Chapter 4 for more details), but with different interactions and constraints in mind. It combines the Orienteering Problem (OP) and the Knapsack Problem (KP). The OP is a well-studied problem in operational research (Golden et al., 1987; Chao et al., 1996; Vansteenwegen et al., 2011; Gunawan et al., 2016), where a person starts on a given point, travels through a region visiting checkpoints, and has to arrive at a control point within a given limited time. Each checkpoint has a score, and the objective of the participant is to find the route that maximizes the total score, i.e., whose sum of scores of the checkpoints visited is maximal. In the ThOP context, the person, i.e., the thief does not score points (steal items) by just visiting a checkpoint but has to steal them and carry them in their knapsack until the end of their robbery journey. As in the TTP, in the ThOP, the thief has a capacitated knapsack to carry the items. Moreover, as items are collected, the knapsack becomes heavier, and the speed of the thief decreases. There is no knapsack rental fee and the thief only aims to find a route and a set of items that maximizes their total stolen profit.

Although the ThOP and the TTP appear to be similar due to the KP as a component, the ThOP appears to be more practical due to two key differences: in the ThOP there is (A) no need to visit all the cities, and (B) the interaction is not through a time-dependent rent for the knapsack, but through a constraint that imposes on the thief a time limit to complete the route. While the relaxation of difference (A) might appear trivial, the consideration of this constraint, i.e., to visit all cities, is typically reflected in the design

[1]It has been compiled from paper "Efficiently solving the thief orienteering problem with a max-min ant colony optimization approach". J. B. C. Chagas, and M. Wagner. Submitted to a journal.

of heuristic (Wagner et al., 2018) and exact (Wu et al., 2017; Neumann et al., 2019) approaches to the TTP, with Chand and Wagner (2016)'s Multiple Traveling Thieves Problem (MTTP) being the only exception known to us. Regarding the difference (B), applications with routing time limit frequently arise in real-world scenarios, where there is no enough time and/or capacity to visit/met all possible locations. Examples of this include tourists planning their sight-seeing trips (Fang et al., 2014), rescue teams planning the visit in case of emergencies (Baffo et al., 2017), and politicians or music bands planning their routes (Aksen and Shahmanzari, 2016; Freeman et al., 2018).

For the ThOP, Santos and Chagas (2018) have proposed a Mixed Integer Non-Linear Programming formulation, but no computational results have been presented due to the formulation's complexity. Instead, the authors have proposed two simple heuristic algorithms, one based on Iterated Local Search (ILS) and one based on a Biased Random-Key Genetic Algorithm (BRKGA). The BRKGA outperformed ILS on large instances, and the authors have attributed this to the diversification introduced of the mutant individuals. Afterwards, Faêda and Santos (2020) have proposed a genetic algorithm that is able to overcome the results reported in (Santos and Chagas, 2018) for most instances. We have also addressed the ThOP in a preceding article (Chagas and Wagner, 2020) with a two-phase swarm intelligence approach based on Ant Colony Optimization (ACO) and a new greedy heuristic, to construct, respectively, the route and the packing plan (stolen items) of the thief. Our ACO is able to find better solutions than other aforementioned algorithms for almost all instances. The efficiency of our ACO algorithm is due to the efficient routes found by the ants which allowed their random packing routine to collect items more efficiently.

In this work, we describe a number of improvements that we incorporated into our ACO algorithm, which made it more substantially effective with regard to the quality of the solutions found. In our computation experiments, we have investigated the importance of the parameters of our ACO algorithm considering these improvement changes. In addition, to make a fairer comparison among the other algorithms already proposed for the ThOP, we have also investigated the parameters of those algorithms and then evaluate their performances on a broad set of instances according to the results already presented in the literature.

The remainder of this paper is structured as follows. In Section 5.1, we formally describe the ThOP and present detailed solution examples to demonstrate the interwovenness characteristic of the multi-components of the problem. In Section 5.2, we present our new solution approach for addressing the ThOP. Section 5.3 reports the

experiments and analyzes the performance of the proposed solution approach against previous ones already proposed in the literature. Finally, in Section 5.4, we present the conclusions of this work.

5.1 Problem definition

The Thief Orienteering Problem (ThOP) can be formally described as follows. There is a set $I = \{1, 2, \ldots, m\}$ of m items and a set $C = \{1, 2, \ldots, n\}$ of n cities. Each item $k \in I$ has a profit p_k and weight w_k associated. In addition, each item is associated with only a single city, but a city can have multiple items. Let us denoted by I_i the set of items localized at city i. From the foregoing definition, $\bigcup_{i \in C} I_i = I$ and $\bigcap_{i \in C} I_i = \varnothing$. The items are scattered among all cities, except cities 1 and n ($I_1 = I_n = \varnothing$). Cities 1 and n are, respectively, the cities where the thief starts and ends their journey. Let us denote by $A = \{(i, j),\ \forall i \in C \setminus \{n\},\ \forall j \in C \setminus \{1\} \mid i \neq j\}$ the set of arcs in which the thief can travel. For any pair of cities i and j with $(i, j) \in A$, the distance d_{ij} between them is known. The thief can make a profit throughout their journey by stealing items and storing them in a knapsack with a limited capacity W. Moreover, the thief has a maximum time T to complete their whole robbery journey. As stolen items are stored in the knapsack, it becomes heavier, and the thief travels more slowly, with a velocity inversely proportional to the knapsack weight. Specifically, when the knapsack is empty, the thief can move with their maximum speed v_{max}. However, when the knapsack is full, the thief moves with the minimum speed v_{min}. In general terms, the thief can move with a speed $v = v_{max} - w \cdot (v_{max} - v_{min}) \,/\, W$, where w is the current weight of their knapsack. The objective of the ThOP is to find a path for the thief that starts from city 1 and ends at city n, as well as a robbery plan, i.e., a set of items chosen from the cities visited that maximizes the total profit stolen, ensuring that the capacity of the knapsack W is not surpassed and the total traveling time of the thief is within the time limit T.

We can represent any solution for the ThOP through a pair $\langle \pi, z \rangle$, where $\pi = \langle 1, \ldots, n \rangle$ is a list of visited cities by the thief, and $z = \langle z_1, z_2, \ldots, z_m \rangle$ is a binary vector representing the packing plan ($z_j = 1$ if item j is collected, and 0 otherwise) adopted by the thief throughout their robbery journey. Note that the first and last cities are fixed for any feasible solution. In addition, the number of cities visited may differ for different solutions.

It is important to note that nothing prevents the thief from visiting a city more than once. In this scenario, the thief should collect all items of a city at once on the last visit from that city to minimize their travel time and consequently has more time to collect other items. One more aspect that should be pointed out is that a solution that visits some cities without collecting any items would only be advantageous if the distances between cities do not respect the triangular inequality because it may be convenient that the thief visits a city just to shorten their route. Nevertheless, as Santos and Chagas (2018) have defined the test problems for the ThOP – which we have also used in this work – in such a way that the distances between cities respect triangular inequality, it can be considered as an implicit optimization that any solution is formed by a list of cities π without repetition.

In order to clarify the characteristics of the ThOP, we depict in Figure 5.1 a small worked example of a ThOP instance that involves 4 cities and 5 items. Note that there are no items in the start (1) and end (4) cities, whereas there are some items of different weights and profit distributed in the other cities (2 and 3). The distances from each pair of cities are given in the edges. In the following, we present in detail some solutions for this instance. For this purpose, let us consider $v_{min} = 0.1$, $v_{max} = 1.0$, $W = 3$, and $T = 75$.

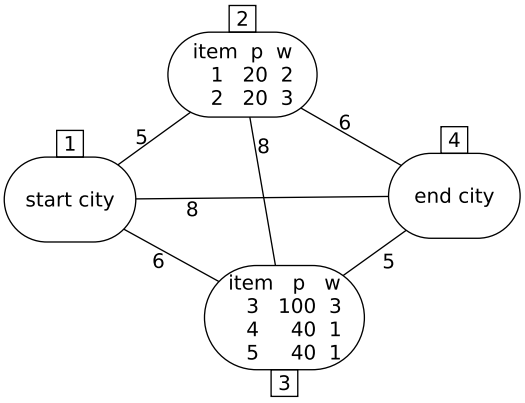


Figure 5.1: A ThOP instance involving 4 cities and 5 items (Santos and Chagas, 2018).

According to the solution representation $\langle \pi, z \rangle$, let us consider the following ThOP solutions for the instance previously described:

- $\langle \langle 1, 2, 3, 4 \rangle, \langle 1, 0, 0, 1, 0 \rangle \rangle$: it is a feasible solution with a total profit of $20 + 40 = 60$. The total weight of stolen items is 3 and the total traveling time is 75, which satisfies both limits W and T. The total traveling time is calculated as:

  - travel from the start city to city 2 at maximum speed: time is computed as $d_{12}/v_{max} = 5/1.0 = 5$;
  - at city 2 the thief steals item 1: the speed decreases to $v = 1.0 - 2 \times (1.0 - 0.1) \,/\, 3 = 0.4$;
  - travel from city 2 to city 3: total traveling time is $5 + d_{23}/v = 5 + 8/0.4 = 5 + 20 = 25$;
  - at city 3 item 4 is collected: the speed drops to $v = 1.0 - 3 \times (1.0 - 0.1) \,/\, 3 = 0.1$;
  - travel from city 3 to the end city: total traveling time is $5 + 20 + d_{34}/v = 5 + 20 + 5/0.1 = 5 + 20 + 50 = 75$.

- $\langle\langle 1, 3, 2, 4\rangle, \langle 1, 0, 0, 1, 0\rangle\rangle$: it is an infeasible solution. Although the stolen items are the same as in the previous solution, the total traveling time (77.43) exceeds the time limit:

  - travel from the start city to city 3 at maximum speed: time is computed as $d_{13}/v_{max} = 6/1.0 = 6$;
  - at city 3 the thief steals item 4: the speed decreases to $v = 1.0 - 1 \times (1.0 - 0.1) \,/\, 3 = 0.7$;
  - travel from city 3 to city 2: total traveling time is $6 + d_{32}/v = 6 + 8/0.7 = 6 + 11.43 = 17.43$;
  - at city 2 item 1 is collected: the speed drops to $v = 1.0 - 3 \times (1.0 - 0.1) \,/\, 3 = 0.1$;
  - travel from city 2 to the end city: total traveling time is $6 + 17.43 + d_{24}/v = 6 + 17.43 + 6/0.1 = 6 + 11.43 + 60 = 77.43$.

- $\langle\langle 1, 3, 4\rangle, \langle 0, 0, 1, 0, 0\rangle\rangle$: it is the optimal solution for this instance with a total profit of 100. The total weight is $3 \leq W$ and the total traveling time is $56 \leq T$:

  - travel from the start city to city 3 at maximum speed: time is computed as $d_{13}/v_{max} = 6/1.0 = 6$;
  - at city 3 the thief steals item 3: the speed decreases to $v = 1.0 - 3 \times (1.0 - 0.1) \,/\, 3 = 0.1$;
  - travel from city 3 to the end city: total traveling time is $6 + d_{34}/v = 6 + 5/0.1 = 6 + 50 = 56$.

Note that the packing plan of the optimal ThOP solution for the example instance happens to be the same as the optimal solution for the knapsack problem. However, it is not always that the thief can steal the best knapsack configuration within the

time limit T. To exemplify this, let us now consider a tighter time limit equal to 20 for the previous instance. For this case, the optimal ThOP solution would be $\langle\langle 1,3,4\rangle,\langle 0,0,0,1,1\rangle\rangle$, which has a total profit of 80 and total traveling time of 18.5.

In order to formally describe the ThOP through a mathematical formulation, we have proposed an alternative Mixed Integer Non-Linear Programming (MINLP) formulation to that proposed by Santos and Chagas (2018). In contrast to Santos and Chagas' formulation, the following formulation uses a polynomial number of decision variables in terms of the number of cities and items. These decision variables are detailed bellow:

- x_{ij} : binary variable that gets 1 if the thief crosses arc $(i,j) \in A$, and 0 otherwise.
- y_i : binary variable that gets 1 if the thief visits city $i \in C$, and 0 otherwise.
- z_k : binary variable that gets 1 if the thief collects item $k \in I$, and 0 otherwise.
- q_i : variable that reports the weight of the knapsack after leaving city $i \in C$.
- t_i : variable that informs the thief's arrival time at city $i \in C$.

With these variables, we can describe the following MINLP formulation for the ThOP.

$$\max \sum_{k \in I} p_k \cdot z_k \tag{5.1}$$

$$\sum_{k \in I} w_k \cdot z_k \leq W \tag{5.2}$$

$$y_i \geq z_k \qquad i \in C, k \in I_i \tag{5.3}$$

$$y_i \leq \sum_{k \in I_i} z_k \qquad i \in C \setminus \{1, n\} \tag{5.4}$$

$$y_1 = y_n = 1 \tag{5.5}$$

$$\sum_{j:(i,j)\in A} x_{ij} = y_i \qquad i \in C \setminus \{n\} \qquad (5.6)$$

$$\sum_{i:(i,j)\in A} x_{ij} = y_j \qquad j \in C \setminus \{1\} \qquad (5.7)$$

$$q_j \geq \left(q_i + \sum_{k\in I_j} w_k \cdot z_k \right) \cdot x_{ij} \qquad (i,j) \in A \qquad (5.8)$$

$$t_j \geq \left(t_i + \frac{d_{ij}}{v_{max} - \nu \cdot q_i} \right) \cdot x_{ij} \qquad (i,j) \in A \qquad (5.9)$$

$$x_{ij} \in \{0,1\} \qquad (i,j) \in A \qquad (5.10)$$

$$y_i \in \{0,1\} \qquad i \in C \qquad (5.11)$$

$$z_k \in \{0,1\} \qquad k \in I \qquad (5.12)$$

$$0 \leq q_i \leq W \qquad i \in C \qquad (5.13)$$

$$0 \leq t_i \leq T \qquad i \in C \qquad (5.14)$$

The objective (5.1) is to maximize the total profit of items collected. Constraint (5.2) ensures that the total weight of items collected does not exceed the knapsack capacity. While constraints (5.3) guarantee that the thief must visit a city to collect any item from it, constraints (5.4) ensure that the thief does not visit cities where no items are selected. Note that constraints (5.4) are not needed for the model to produce feasible solutions. However, these constraints strengthen the model by removing unprofitable route combinations. Constraint (5.5) simply imposes that cities 1 and n have to be visited once they are, respectively, the fixed start and end points of any feasible route. Constraints (5.6) and (5.7) guarantee route connectivity. Constraints (5.8) and (5.9) guarantee that the knapsack weight and the route time is properly increasing along the route according to the items, which also avoid subcycles. Note that constraints (5.8) and (5.9) are non-linear. Finally, constraints (5.10)-(5.14) define the scope and domain of the decision variables. Note that, as constraints (5.13) ensure that the knapsack weight must be always less than knapsack capacity W throughout the route, constraint (5.2) could be removed. However, constraints (5.13) may be weaker for the purpose expressed by constraint (5.2) due to the multiplication by the variable x_{ij}, which allows that weak fractional solutions to be considered during the resolution of the formulation.

It is worth mentioning that constraints (5.8) can be linearized by rewriting them using Big-M constants as shown in (5.15). On the other hand, we cannot linearize constraints (5.9) as they involve divisions and multiplications of decision variables. Nevertheless, we have also rewritten them using Big-M constants, as shown in (5.16), to remove their non-linear multiplications. In constraints (5.15) and (5.16), we have used different Big-M constants M'_j and M''_{ij}, which should assume any sufficiently large number that is greater than or equal to $W + \sum_{i \in I_j} w_i$ and $T + d_{ij}/v_{min}$, respectively.

$$q_j \geq q_i + \sum_{k \in I_j} w_k \cdot y_k - M'_j \cdot (1 - x_{ij}) \qquad (i,j) \in A \qquad (5.15)$$

$$t_j \geq t_i + \frac{d_{ij}}{v_{max} - \nu \cdot q_i} - M''_{ij} \cdot (1 - x_{ij}) \qquad (i,j) \in A \qquad (5.16)$$

Although the foregoing mathematical formulation may be used for solving the ThOP, we have not considered it in our experiments due to its complexity. As constraints (5.16) are still non-linear, they greatly increase the complexity of the formulation, making it impracticable to solve even the smallest-size instance defined in the literature for the ThOP (Santos and Chagas, 2018). Therefore, we have bet in a heuristic strategy for helping the thief in their robbery, leaving an improved mathematical formulation and exact algorithms for future investigation. As our MINLP formulation might be used as a starting point for other investigations, we have made it publicly available at https://github.com/jonatasbcchagas/minlp_thop, which has been implemented using PySCIPOpt (Maher et al., 2016), a Python interface for the SCIP Optimization Suite (Gamrath et al., 2020).

5.2 Problem-solving methodology

Throughout this section, we describe our heuristic approach, called ACO++, for solving the ThOP. Our ACO++ is an improved version of the ACO algorithm previously presented in (Chagas and Wagner, 2020). At the end of this section, we highlight the differences between both algorithms.

5.2.1 The overall algorithm

Our solution approach has been loosely based on Wagner's TTP study (Wagner, 2016). As in (Wagner, 2016), we consider the use of swarm intelligence based on Ant Colony Optimization (ACO) (Dorigo and Di Caro, 1999) technique as the core of our algorithm. In brief, ACO algorithms consist of an essential class of probabilistic search techniques that are inspired by the behavior of real ants. These algorithms have proven to be efficient in solving a range of combinatorial problems (Dorigo and Blum, 2005). The basic idea behind ACOs is that ants construct solutions for a given problem by carrying out walks on a so-called construction graph. These walks are influenced by the pheromone values that are stored along the edges of the graph. During the optimization process, the pheromone values are updated according to good solutions found during the optimization, which should then lead the ants to better solutions in further iterations of the algorithm. We refer the interested reader to the book by Dorigo and Birattari (2010) for a comprehensive introduction on ACO concepts.

As in (Wagner, 2016), we have used the ACO for determining the thief's route, while another algorithm for determining their packing plan for each route found by the ants. We have used the MAX-MIN ant system by Stützle and Hoos (2000), which restricts all pheromones to a bounded interval in order to prevent pheromones from dropping to arbitrarily small values. In our implementation, we have used Stützle's ACOTSP 1.0.3 framework[2] for constructing the thief's route. The ACOTSP is an efficient framework that implements several ACO algorithms for the symmetric TSP. The overall logic of the ACOTSP framework remains unchanged in our proposed algorithm. Some minimal modifications have been performed to adapt it to the ThOP specifications. To construct the feasible routes for the thief, we have established that the first and last cities must be those where the thief begins and ends their robbery journey. Unlike the TSP, the ThOP does not require that all cities are visited, therefore we have made an adaptation so that the ants built their routes until the thief's destination city, that is, city n has been visited. Thus, the ants are able to build routes of varying sizes.

In the ACOTSP framework, the pheromone trail updates are performed based on the quality of the TSP routes found by ants. Since the objective of the TSP is to find the shortest possible route visiting each city, the fitness of a given route is inversely proportional to its total distance. In our ACOTSP adaptation, the fitness of each route is set in terms of the quality of the stolen items throughout the route, because stolen

[2]Publicly available online at http://www.aco-metaheuristic.org/aco-code

items define the quality of ThOP solutions, which are defined by our proposed packing routine (to be described later). As the ACOTSP framework is developed explicitly for the TSP, a minimization problem where its solutions have positive objective values, we consider that the fitness of a ThOP's route π is inversely proportional to $UB + 1 - p(z)$, where UB is an upper bound for the ThOP and $p(z)$ is the total profit of packing plan z. Note that in this way we can maintain the same behavior of fitness of the TSP solutions, without modifying the ACOTSP framework structure. The upper bound UB is defined as the optimal solution for the KP version that allows selecting fractions of items. This KP version can be easily solved in $\mathcal{O}(m \log_2 m)$ (Toth and Martello, 1990).

In Algorithm 6, we show the simplified overview of our ACO++. Initially (Line 1), the best ThOP solution (route and packing plan) found by the algorithm is initialized as an empty solution. The algorithm performs its iterative cycle (Lines 2 to 14) as long as the stopping criterion is not fulfilled. At Line 3, each ant constructs a candidate route for the thief. For each route π (Line 4), a packing plan is created (Line 5). The ACOTSP framework allows us to apply on the routes found by the ants some classic local search procedures: 2-opt, 2.5-opt, and 3-opt (Aarts et al., 2003). If any local search is enabled in our algorithm (Line 6), that local search procedure is done on each route π, thus generating a route π' (Line 7), which may be better than π regarding the distance costs. Next, a packing plan z' is created from π' (Line 8). If z' is better than z (Line 9), π and z are replace by π' and z' (Line 10). At Lines 11 to 12, the best solution found is possibly updated. Note that we remove from π all cities where no items have been stolen according to the packing plan z (Line 12) in order to get a faster route. As we have stated before, this is only true as all ThOP instances use distances that preserve the triangular inequality. After every route has been considered, ACO statistics and the pheromone values are updated according to the quality of the ThOP solutions found (Line 13). At the end of the algorithm (Line 21), the best solution found is returned.

5.2.2 Randomized packing heuristic

In order to complete the description of the proposed ACO++ algorithm, we now present our strategy used for constructing a packing plan from a given route π. As stated by Polyakovskiy and Neumann (2015), even when the route of the thief is kept fixed, finding the optimal packing configuration is $\mathcal{NP}$-hard. Therefore, we have used the core idea of the randomized heuristic presented for solving the ThOP in our

Algorithm 6: ACO++ algorithm for the ThOP

```
1  π^best ← ∅, z^best ← ∅
2  repeat
3      Π ← construct routes using ants
4      foreach route π ∈ Π do
5          z ← construct a packing plan from π
6          if local search procedure is activated then
7              π' ← perform a local search procedure on route π
8              z' ← construct a packing plan from π'
9              if profit of z' is higher than profit of z then
10                 π ← π', z ← z'
11         if profit of z is higher than profit of z^best then
12             π^best ← ζ(π), z^best ← z
13     update ACO statistics and pheromone trail
14 until stopping condition is fulfilled
15 return π^best, z^best
```

$\zeta(\pi)$ removes from π all cities where no items are stolen according to the packing plan z.

preliminary work (Chagas and Wagner, 2020). Our packing heuristic algorithm has been developed based on a heuristic strategy used as subroutine in the PACKITERATIVE, an efficient packing algorithm developed for the TTP (Faulkner et al., 2015). In contrast to the deterministic PACKITERATIVE and its subroutine, we have decided to design our packing algorithm in a non-deterministic way. This decision has been motivated once in our preliminary experiments, we have observed that ants for many times have been able to find identical or very similar routes throughout the iterations of the ACO++ algorithm. Thus, with a non-deterministic strategy, we increase the exploration of the packing plan space even for a fixed route, which can lead to finding better configurations.

In Algorithm 7, we present in detail the steps performed in our packing heuristic algorithm. It starts by creating a set of all items that can be stolen by the thief from their route π (Line 1), and initializes the best packing plan found without any items (Line 2). Our packing heuristic algorithm seeks to find a good packing plan z from multiple attempts for the same route π. The number of attempts is defined by *ptries*. Each attempt is described between Lines 4 to 20. At the beginning of each attempt (Line 5), we uniformly select three random values (θ, δ, and γ) between 0 and 1, and then normalize them so that their sum is equal to 1 (Line 6). These values are used to compute a score s_i (Line 8) for each candidate item to be selected (Line 7), where θ, δ,

and γ define, respectively, exponents applied to profit p_i, weight w_i, and distance d_i in order to manage their impact. The distance d_i is calculated according to the route π by sum all distances from the city where the item i is to the end city. Equation (5.17) shows as the score of item i is calculated.

$$s_i = \frac{(p_i)^{\theta}}{(w_i)^{\delta} \cdot (d_i)^{\gamma}} \tag{5.17}$$

Algorithm 7: Randomized Packing Algorithm

```
1  I ← create a set of all items located in any city in π
2  z ← ∅
3  try ← 1
4  repeat
5      θ ← rand(0, 1), δ ← rand(0, 1), γ ← rand(0, 1)
6      normalize θ, δ, and γ so that θ + δ + γ = 1
7      foreach i ∈ I do
8          s_i ← compute score for item i using θ, δ, and γ        // Eq. (5.17)
9      sort the items of I in non-decreasing order of their scores
10     z' ← ∅
11     for i ∈ I do
12         z' ← z' ∪ {i}
13         if weight of z' is higher than W then z' ← z' \ {i}
14         else
15             t ← compute the travel time to steal z' following the order of the
                route π by visiting only cities with items selected
16             if t is longer than T then z' ← z' \ {i}
17     if profit of z' is higher than profit of z then
18         z ← z'
19     try ← try + 1
20 until try > ptries
21 return z
```

Note that each score s_i incorporates a trade-off between a distance that item i has to be carried over, its weight, and its profit. Equation (5.17) is based on the heuristic PACKITERATIVE that has been developed for the TTP (Faulkner et al., 2015). However, unlike in (Faulkner et al., 2015), we consider an exponent for the term of distance to vary the importance of its influence. Furthermore, the values of all exponents are

randomly drawn between 0 and 1 for each attempt (and then normalized) to search the space for greedy packing plans.

After computing scores for all candidate items, we sort them in non-decreasing order of their scores (Line 9). The scores of items are used to define their priority in the packing strategy. The higher the score of an item, the higher its priority. Between Lines 11 and 16, we create the packing plan for the current attempt by considering the items according to their priorities. If an item violates the constraints of the ThOP (Lines 13 and 16), it is not selected. Note that we calculate travel time (Line 15) from the cities listed on route π, but we ignore those cities where no items are selected. After completing the current attempt's packing plan, its quality is compared to the best packing plan so far (Line 17), which is then possibly updated (Line 18). At the end of all attempts, the best packing plan found is returned (Line 21).

5.2.3 Differences between the ACO++ and ACO approaches

Compared to the ACO described in (Chagas and Wagner, 2020), we have incorporated two new features into our ACO++. These features are described in the following:

1. The ants of ACO++ construct routes that do not necessarily visit all cities, while in our previous ACO the ants always construct complete TSP tours, i.e., all cities are visited. Note that, although both algorithms remove from the route all cities in which no item is selected, there is now a higher consistency with respect to the ThOP's definition, because ants do not have to visit all cities. Note that this allows ants to construct routes that might not be easily constructed from our previous ACO. In addition, when a route visits fewer cities fewer items are available to be collected from that route, which reduces the search space to find a packing plan, thus making our packing routines more computationally efficient. It is also because of this last point, that we had to adapt our randomized packing heuristic algorithm to consider only the items that can be selected from each constructed route. However, its core idea remains unchanged.

2. There is now the possibility to apply different local searches on each route constructed by the ants in our ACO++. While this results in a TSP-bias toward shorter routes, finding shorter routes may help the thief to better plan their robbery journey.

5.3 Computational Study

In this section, we present the experiments performed to study the performance of the proposed algorithm against other algorithms proposed for the ThOP (Santos and Chagas, 2018; Faêda and Santos, 2020; Chagas and Wagner, 2020). As the computational budget of all ThOP algorithms are based on wallclock time, in order to enable a fair comparison, we have rerun all ThOP codes, except for Faêda and Santos (Faêda and Santos, 2020)'s algorithm because we have not had access to their code. In our experiments, each run of each algorithm has been sequentially (nonparallel) performed on a machine with an Intel(R) Xeon(R) CPU X5650 @ 2.67GHz, running CentOS 7.4.

Our algorithm has been implemented based on Thomas Stützle's ACOTSP 1.0.3 framework, which is in C programming language. Our code, as well as all raw results and solutions (tours and packing plans), are publicly available at https://github.com/jonatasbcchagas/acoplusplus_thop.

5.3.1 Benchmarking instances

In order to evaluate the different ThOP approaches, we use all 432 ThOP instances from (Santos and Chagas, 2018). These have been created based on the TTP instances (Polyakovskiy et al., 2014) by removing the items in city n and by adding a maximum travel time. The instances have the following characteristics:

- numbers of cities: 51, 107, 280, and 1000 (TSP instances: *eil51*, *pr107*, *a280*, *dsj1000*);
- numbers of items per city: *01*, *03*, *05*, and *10* (all cities of a single ThOP instance have the same number of items, except for the cities in which the thief starts and ends their journey, where no items are available);
- types of knapsacks: weights and values of the items are bounded and strongly correlated (*bsc*), uncorrelated (*unc*), or uncorrelated with similar weights (*usw*);
- sizes of knapsacks: *01*, *05* and *10* times the size of the smallest knapsack, which is defined by summing the weight of all items and dividing the sum by 11;
- maximum travel times: *01*, *02*, and *03* classes. These values refer to 50%, 75%, and 100% of instance-specific reference times defined in the original ThOP paper (Santos and Chagas, 2018).

The 432 ThOP instances can be obtained by combining the different characteristics described above. In the remainder of this chapter, each instance is identified as `XXX_YY_ZZZ_WW_TT`, where `XXX`, `YY`, `ZZZ`, `WW` and `TT` indicate the different characteristics of the instance at hand. For example, `pr107_05_bsc_01_01` identifies the instance with 107 cities (TSP instance *pr107*), 5 items per city with their weights and values bounded and strongly correlated with each other, and the smallest knapsack and time limit defined.

5.3.2 Parameter tuning

The ACOTSP framework allows us to set a large number of parameters. We consider the following: *ants* defines the number of ants used; *alpha* controls the relative importance of pheromone trails in the construction of routes; *beta* defines the influence of distances between cities for construction the routes; *rho* sets the evaporation rate of the pheromone trail; and *localsearch* controls whether and what local search procedure is applied to tours. Besides, we analyze the influence of our parameter *ptries*, which is used for deciding how many attempts our randomized packing algorithm performs to determine the set of stolen items. Regarding the stopping criterion of our algorithm, as in the previous work on the ThOP, we have defined as stopping criteria the execution time equal to $\lceil 0.1m \rceil$ seconds, which is given in terms of the number of items m of each particular instance.

In order to find suitable configuration values for these parameters, we have followed the same tuning experiments used in (Chagas and Wagner, 2020), i.e., we have used the Irace package (López-Ibáñez et al., 2016b), which implements an iterated racing framework for the automatic configuration of algorithms (Birattari et al., 2010), for analyzing the influence of parameter values across different types of instances. We have divided all 432 instances into 48 groups and then execute Irace on each of them. Each group is identified as `XXX_YY_ZZZ`, where `XXX` informs the TSP base group, `YY` the number of items per city and `ZZZ` the type of knapsack. Each group `XXX_YY_ZZZ` contains all nine instances defined with different sizes of knapsacks and maximum travel time.

Table 5.1 shows the parameter values we have considered in our analysis. These values have been selected following preliminary experiments. In our experiments, we have used all Irace default settings, except for the parameter *maxExperiments*, which

has been set to 5000. This parameter defines the stopping criteria of the tuning process. We refer the readers to (López-Ibáñez et al., 2016a) for a complete user guide of the Irace package.

Table 5.1: Parameter values considered during the tuning experiments.

Parameter	Investigated values
ants	$\{10, 20, 50, 100, 200, 500, 1000\}$
alpha	$\{0.00, 0.01, 0.02, \ldots, 10.00\}$
beta	$\{0.00, 0.01, 0.02, \ldots, 10.00\}$
rho	$\{0.00, 0.01, 0.02, \ldots, 1.00\}$
ptries	$\{1, 2, 3, 4, 5\}$
localsearch	{no local search, 2-opt, 2.5-opt, 3-opt}

In Figure 5.2, we plot for each group all configurations returned by Irace at the end of its run. Each parallel coordinate plot lists for each of the 48 groups (shown in the left-most column) the configurations returned by Irace (shown in the other columns). As Irace can return more than one configuration, multiple configurations are sometimes shown. Each axis indicates a parameter and its range of values, and each configuration of parameters is described by a line that cuts each parallel axis in its corresponding value. Through the concentration of the lines, we can see which parameter values have been most selected among all tuning experiments. We have used different colors and styles for lines in order to emphasize the results obtained for each group individually. All logs generated by the Irace executions, as well as their settings can be found at the GitHub link along with our code.

We can make several observations. Firstly, we notice that the number of ants is typically higher than 100. The importance of the pheromone trail (α) is typically low, especially for the groups of instances that consider the TSP bases *eil51*, and *a280*. In turn, the importance of distances between cities (β) varies depending on the underlying TSP instance. This is not too surprising, as the underlying TSP instances are different in nature and not normalized, hence requiring different values of beta. The evaporation rate of the pheromone trail has had a behavior more spread, although it seems to have a compensation correlation between the parameter *beta*: the higher the influence of distances between cities, the lower the evaporation rate of the pheromone trail. We can also observe that nearly all tuned configurations require the multiple invocation of our randomized packing heuristic, with the number of packing attempts

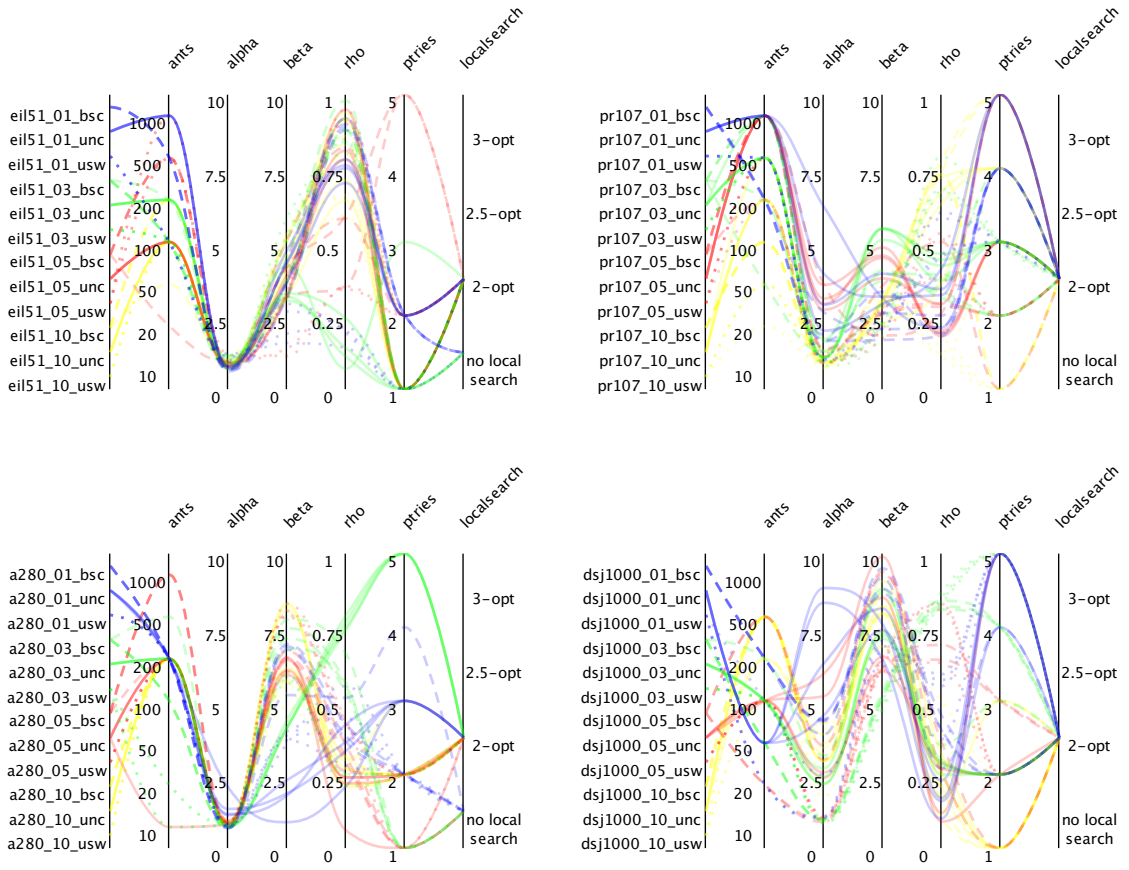


Figure 5.2: Irace results for the 48 groups of instances. Blue, green, red, and yellow lines represent, respectively, groups of instances with 1, 3, 5, and 10 items-per-city. Dashed, solid, and dotted lines are used, respectively, to emphasize the groups of instances with items where their weights and values are bounded and strongly correlated (*bsc*), uncorrelated with similar weights (*usw*), and uncorrelated (*unc*).

widely spread among each other. Regarding the application of local searches on routes, one can note that for most groups of instances the use of 2-opt moves produces better ThOP solutions. However, some configurations do not include the use of any local search. Note that there are no configurations that indicate the use of 2.5-opt and 3-opt moves. Potentially, this is because high-quality TSP routes do not necessarily result in high-quality ThOP routes. Therefore, there may be no need to use local searches with larger neighborhood moves based solely on route distances as an improvement phase for ThOP routes.

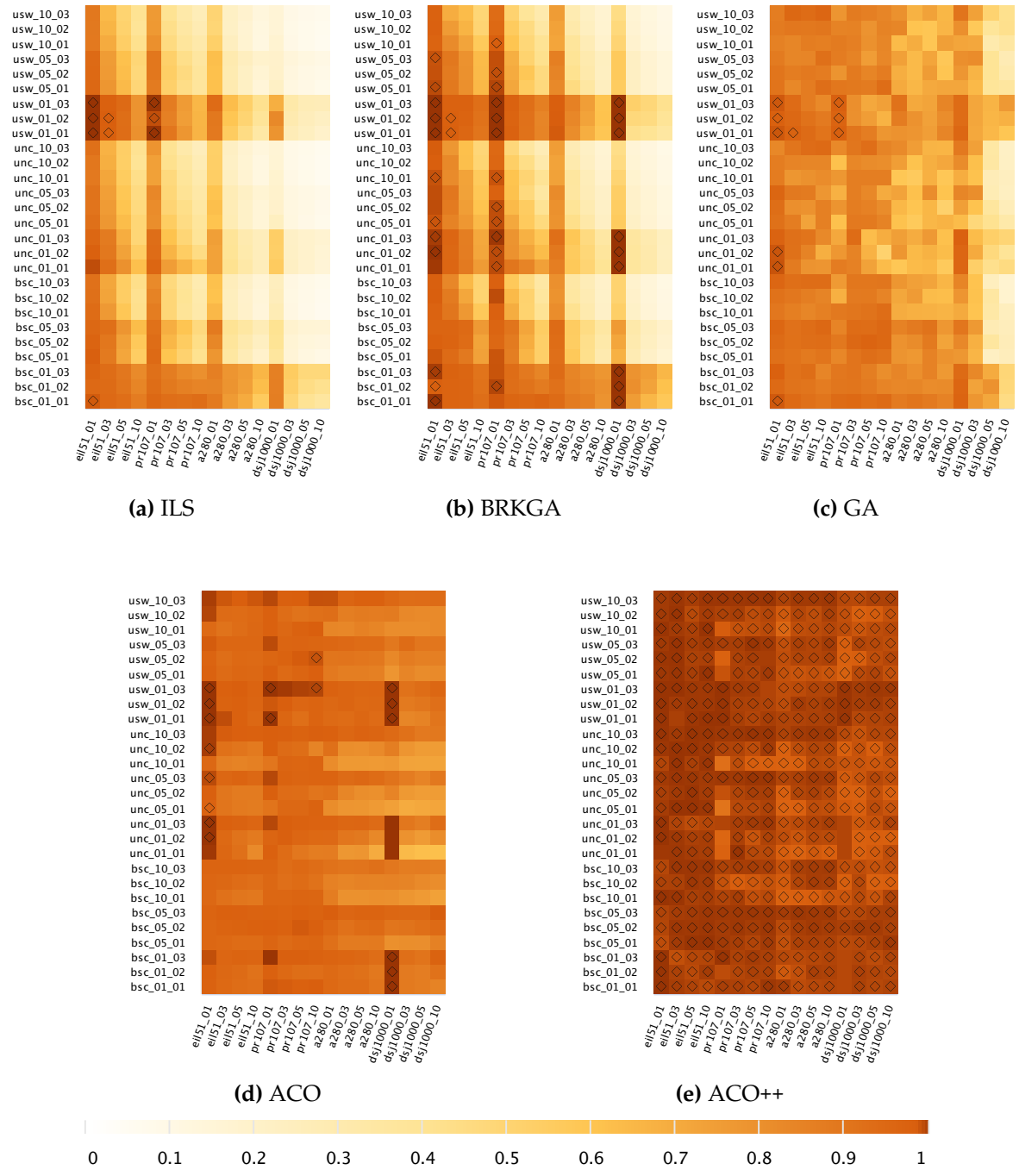
5.3.3 Comparison of ThOP solution approaches

We compare the quality of the solutions obtained by ACO++ with the quality of the solutions obtained by other algorithms (ILS (Santos and Chagas, 2018), BRKGA (Santos and Chagas, 2018), GA (Faêda and Santos, 2020), ACO (Chagas and Wagner, 2020)) already proposed for the ThOP. In order to make a fair comparison, we have tuned the parameters of the BRKGA and ACO following the same procedure used in the tuning of the parameters of ACO++, i.e., we have individually executed the Irace for each 48 diferent groups of instances. The ILS algorithm has no parameters to be tuned (Santos and Chagas, 2018), while for the GA, we have not investigated its parameters because we have not had access to its code. Therefore, for the GA, we have made our analysis based on the results reported in (Faêda and Santos, 2020).

Due to the randomized nature of all algorithms, we have performed 30 independent repetitions on each instance. Each run has been executed with the parameter values with the best mean performance among those returned by Irace. ILS, BRKGA, and ACO codes, as well as their tuned configurations, raw results and solutions found, are also available at the GitHub link along with our ACO++.

In the first analysis, we compare the performance of the solutions obtained by measuring for each instance the approximation ratio of each algorithm. More precisely, for each instance and algorithm, we take the average objective value obtained considering the independent runs of that algorithm and compute the ratio between that average objective value and the best objective value found among all algorithms. Note that the higher the approximation ratio, the higher the average performance of that particular algorithm. In Figure 5.3, we plot for every instance and algorithm the approximation ratio as a heatmap in order to highlight larger differences. In addition, we highlight with a diamond symbol the instances for which each algorithm has found best known solutions.

From Figure 5.3, we can make several observations. As stated by Santos and Chagas (2018), we can also confirm that their BRKGA has outperformed their ILS for most instances, with higher prominence on the larger-size instances. In addition, one can note that their algorithms perform better for instances that involve only one item per city. Regarding the best-known solutions, we can see that their algorithms have not been able to find many of them. The GA proposed by Faêda and Santos (2020) has outperformed, in general, both BRKGA and ILS solution approaches. Although BRKGA has found more best-known solutions, the GA has a more uniform



(a) ILS **(b)** BRKGA **(c)** GA

(d) ACO **(e)** ACO++

Figure 5.3: Approximation ratio of the solution approaches. Diamond symbols highlight in which the instances each algorithm has found the best solutions.

behavior regarding the dimensions of the instances. Note that our previous ACO algorithm (Chagas and Wagner, 2020) has reached a better approximation ratio for almost all instances when compared to GA and also to ILS and BRKGA. In turn, our current ACO++ algorithm has presented a better or equal performance regarding the

other algorithms for almost all instances. Similarly, it has typically found the best solutions for most of the instances.

In order to compare each pair of algorithms as to the best solutions found by them, we show in Table 5.2 the percentage of the number of instances in which every algorithm found better or equal quality solutions than another algorithm. The results shown in this table corroborate with those shown in Figure 5.3. In addition to showing that the ACO++ algorithm outperformed all other algorithms by more than 96% of the total of instances, we can also see that our previous ACO also is more efficient than ILS, BRKGA, and GA by over 88% of instances. In turn, GA is more efficient than ILS and BRKGA, and BRKGA overcomes ILS.

Table 5.2: Percentage of the number of instances in which algorithm i found better or equal quality solutions than algorithm j.

$i \downarrow \quad j \rightarrow$	ILS	BRKGA	GA	ACO	ACO++
ILS	-	3.01%	20.37%	4.63%	2.55%
BRKGA	99.54%	-	37.50%	14.58%	8.56%
GA	81.71%	64.58%	-	2.78%	2.31%
ACO	96.99%	88.89%	98.61%	-	5.32%
ACO++	99.54%	96.06%	99.54%	98.15%	-

As both algorithms based on ACO metaheuristics have had the best and most similar performances, we statistically compare the quality of their solutions by using the Wilcoxon signed-rank test on the results achieved in the 30 independent runs. At a significance level of 5%, the performance of ACO++ has been statistically worse than ACO in only 11 instances, in 12 instances there is no difference between the performance of both algorithms, while in 409 instances (about 95% of total) ACO++ has been better than ACO.

In Table 5.3, we summarize the results obtained with a closer analysis of the solutions found by ACO and ACO++. For each TSP base instance (`XXX`) and number of items per city (`YY`), which resulted in 27 instances each, we show averaged information concerning all the best solutions achieved by both approaches. Column $\mathcal{D}$ shows the ratio between the total distance traveled and the number of cities visited by the thief, while columns $\%T$ and $\%W$ report, respectively, the percentage spent of the time limit and the percentage used of the knapsack capacity. If values in these last two columns are close to 100%, then these indicate limiting factors. Note that both algorithms have

a similar use of the time limit. On the other hand, the solutions found by ACO++ have used more the knapsack capacity, especially for instances with more cities and items. From the values in column $\mathcal{D}$, we can understand this behavior. Note that the ratio between the total distance traveled and the number of cities visited of the solutions found by ACO is higher than those found by ACO++. Note that the solutions found by ACO have a ratio between the total distance traveled and the number of cities visited higher than those solutions found by ACO++, which indicates that ACO has found the most spread-out routes and/or with more edge crossings. Therefore, as the routes found by ACO++ are more condensed and/or efficient, the thief is able to travel more effectively and, consequently, uses better the knapsack capacity, thus managing to collect a better set of items. To illustrate this behavior, Figure 5.4 shows, for some instances where the resulting quality differs significantly, the best solution found by each algorithm.

In summary, we can see that our ACO++ has been able to find significantly more efficient routes, which allows achieving better packing plans, and, consequently, achieving higher profits.

5.4 Conclusions

In this chapter, we have proposed a swarm-based approach to the Thief Orienteering Problem (ThOP), an academic multi-component problem that combines the Orienteering Problem and the Knapsack Problem. For solving the problem, we have combined a heuristic approach based on Ant Colony Optimization with a randomized packing heuristic. Using extensive tuning on groups of instances, we have studied the effects of our algorithmic components. Furthermore, we have evaluated the performance of the algorithm on the complete set of instances available in the literature. The experiments show that our solution strategy is able to find better solutions with large improvements when compared to the other solution methods proposed for the problem. Based on our analysis, the efficiency of our algorithm is due to the fact that ants have been able to find more efficient routes, which has allowed our packing heuristic to select a better set of items.

For future research, we can point out as a promising direction the investigation of a version of the problem that considers multiple thieves in order to provide the

Table 5.3: Information on the structure of the best solutions found. $\mathcal{D}$ is the ratio between the total distance traveled and the number of cities visited; $\%T$ and $\%W$ denote the percentage spent of the time limit and the percentage used of the knapsack capacity.

TSP base (XXX)	Number of items per city (YY)	ACO			ACO++		
		$\mathcal{D}$	%T	%W	$\mathcal{D}$	%T	%W
eil51	01	10.91	98.60	78.06	10.46	99.00	79.55
	03	9.46	99.60	83.85	8.63	99.55	85.34
	05	9.26	99.62	83.39	8.53	99.78	85.48
	10	9.12	99.84	85.26	8.20	99.88	86.57
pr107	01	718.92	99.66	79.58	680.85	99.74	81.82
	03	498.71	99.85	81.78	476.67	99.85	83.84
	05	471.77	99.93	83.22	445.23	99.95	83.79
	10	449.09	99.95	84.95	417.47	99.93	84.40
a280	01	16.52	99.61	79.57	14.23	99.80	83.94
	03	12.60	99.74	81.83	10.45	99.76	85.68
	05	11.84	99.95	82.72	9.61	99.93	86.27
	10	11.17	99.92	83.02	9.22	99.92	86.26
dsj1000	01	44632.08	74.72	79.31	37015.71	72.08	86.16
	03	26635.61	99.90	82.91	18943.46	99.89	89.57
	05	25648.23	99.85	82.43	18064.09	99.82	89.91
	10	23795.22	99.92	84.03	17700.59	99.68	90.35

more general problem of team orienteering. Another interesting direction would be to approach the problem in a bi-objective version, where are to maximize the total profit and minimize the total distance traveled. By combining both foregoing directions, a very interesting, challenging, and general problem would be created, with potential applications in real-world scenarios with routing problems under time-dependent limitations.

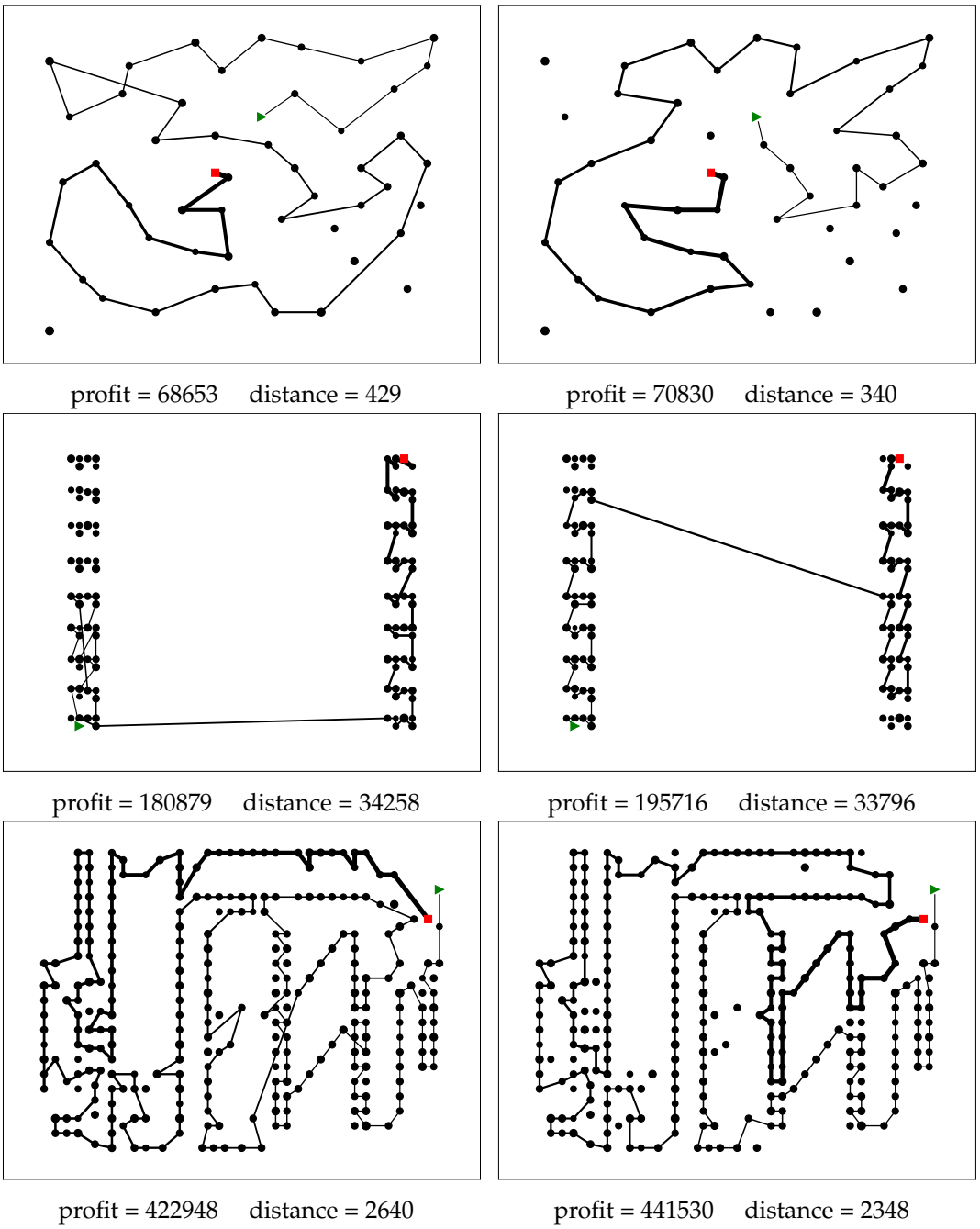


Figure 5.4: The graphical representation of the solutions found by ACO (left) and ACO++ (right) for instances *eil51_10_bsc_01_03* (top), *pr107_05_usw_10_02* (middle), and *a280_10_unc_01_03* (bottom). The cities are plotted in their respective coordinates. The initial and final cities are represented by a green triangle and a red square, respectively, while black points represent the other cities. The continuous lines connecting pairs of cities represents the route performed by the thief. The line thickness increases according to the total weight picked by the thief.

Chapter 6

Conclusions and open perspectives

We have addressed four combinatorial problems with multiple interacting components. Two of them are pickup and delivery problems with loading constraints: the Double Vehicle Routing Problem with Multiple Stacks (DVRPMS), and the Double Traveling Salesman Problem with Partial Last-In-First-Out Loading Constraints (DTSPPL). The other two are nonlinear problems that combine classic combinatorial optimization problems in their formulations: the Traveling Thief Problem (TTP), and the Thief Orienteering Problem (ThOP). The DTSPPL and ThOP are more realistic variants of the DVRPMS and TTP, respectively. All these problems have practical and theoretical relevance as they follow the realistic and integrated trends on current real-world optimization problems.

For each of the problems studied here, we have developed mathematical models and/or heuristic algorithms for treating it. Specifically, for the DVRPMS, a new Integer Linear Programming (ILP) formulation and a heuristic algorithm basead on the Variable Neighborhood Search (VNS) have been proposed. For the DTSPPL, we have mathematically formulated it by means two ILPs and also proposed a heuristic algorithm based on the Biased Random-Key Genetic Algorithm (BRKGA). We have approached a bi-objective formulation of the TTP with two heuristic algorithms; the first one is based on the BRKGA and the Non-Dominated Sorting Genetic Algorithm II (NSGA-II), while the second algorithm is a weighted-sum method combined with a two-stage heuristic. With respect to ThOP, we have formulated it through mathematical nonlinear models, and also developed heuristic algorithms based on BRKGA, Iterated Local Search (ILS), and Ant Colony Optimization (ACO) algorithms.

Our main contributions have consisted of proposing solution approaches able to improve the results of problems already known in the literature, and also proposing

more realistic variations of these problems. As they are all highly complex problems, our heuristic algorithms have generally been able to find better solutions than those found by our mathematical models. Indeed, our mathematical models played a major role in the formulation of the problems and/or in the validation of the results.

Regarding the DVRPMS, from the results of our mathematical model, we have found new upper-bound values for many instances. In addition, our VNS algorithm proposed for the DVRPMS has found solutions with higher quality in shorter computational time for most instances when compared to the methods already present in the literature. For the DTSPPL, both ILP formulations have been able to solve to proven optimality only the smaller instances, while the BRKGA has found high-quality solutions for all instances, requiring on average short computing times. Regarding the TTP, with our algorithm based on the BRKGA and NSGA-II, we have won the first and second places, at EMO2019 and GECCO2019 competitions,[1,2] respectively. In addition, our last proposed algorithm based on the weighted-sum method combined with a two-stage heuristic has reached significantly better results than those presented in the competitions. Finally, for the ThOP, we have not presented the results of our mathematical models due to their high complexity. However, the results of our heuristic algorithms have shown able to find high-quality solutions, especially those based on the ACO technique.

There are many possibilities for extending this work. With respect to the DVRPMS, one could investigate better ways to evaluate and represent the solutions in order to address large instances of the problem. For the DTSPPL, maybe the most relevant study is to model and solve the Double Traveling Salesman Problem with Multiple Stacks and Partial Last-In-First-Out Loading Constraints (DTSPMSPL), the version of the DTSPPL where the loading compartment of the vehicle is divided into multiple stacks. For the TTP and ThOP, it would be interesting to investigate the influence of different algorithmic components over different instances. With this knowledge, one could develop customized algorithms based on the instance characteristics, thus achieving better solutions.

[1] *EMO-2019* https://www.egr.msu.edu/coinlab/blankjul/emo19-thief/
[2] *GECCO-2019* https://www.egr.msu.edu/coinlab/blankjul/gecco19-thief/

Bibliography

E. Aarts, E. H. Aarts, and J. K. Lenstra. *Local search in combinatorial optimization*. Princeton University Press, 2003.

A. Aggelakakis, J. Bernardino, M. Boile, P. Christidis, A. Condeco, M. Krail, A. Papanikolaou, M. Reichenbach, J. Schippl, et al. The future of the transport industry. Technical report, Institute for Prospective and Technological Studies, Joint Research Centre, 2015.

A. Agrawal, T. Menzies, L. L. Minku, M. Wagner, and Z. Yu. Better software analytics via" duo": Data mining algorithms using/used-by optimizers. *Empirical Software Engineering*, 25(3):2099–2136, 2020.

D. Aksen and M. Shahmanzari. A periodic traveling politician problem with time-dependent rewards. In A. Fink, A. Fügenschuh, and M. J. Geiger, editors, *OR*, Operations Research Proceedings, pages 277–283. Springer, 2016. ISBN 978-3-319-55701-4; 978-3-319-55702-1.

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

D. Applegate, W. Cook, and A. Rohe. Chained lin-kernighan for large traveling salesman problems. *INFORMS Journal on Computing*, 15(1):82–92, 2003.

A. Auger, J. Bader, D. Brockhoff, and E. Zitzler. Investigating and exploiting the bias of the weighted hypervolume to articulate user preferences. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 563–570. ACM, 2009.

I. Baffo, P. Carotenuto, and S. Rondine. An orienteering-based approach to manage emergency situation. *Transportation Research Procedia*, 22:297 – 304, 2017. ISSN 2352-1465.

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

M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle. F-race and iterated f-race: An overview. In *Experimental methods for the analysis of optimization algorithms*, pages 311–336. Springer, 2010.

J. Blank, K. Deb, and S. Mostaghim. *Solving the Bi-objective Traveling Thief Problem with Multi-objective Evolutionary Algorithms*, pages 46–60. Springer, 2017.

M. R. Bonyadi, Z. Michalewicz, and L. Barone. The travelling thief problem: The first step in the transition from theoretical problems to realistic problems. In *2013 IEEE Congress on Evolutionary Computation*, pages 1037–1044. IEEE, 2013.

M. R. Bonyadi, Z. Michalewicz, M. R. Przybylek, and A. Wierzbicki. Socially inspired algorithms for the TTP. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 421–428. ACM, 2014.

M. R. Bonyadi, Z. Michalewicz, M. Wagner, and F. Neumann. Evolutionary computation for multicomponent problems: opportunities and future directions. In *Optimization in Industry*, pages 13–30. Springer, 2019.

K. Bringmann and T. Friedrich. Approximation quality of the hypervolume indicator. *Artificial Intelligence*, 195:265 – 290, 2013.

F. Carrabs, R. Cerulli, and M. G. Speranza. A branch-and-bound algorithm for the double tsp with two stacks. Technical report, Dipartimento di Matematica e Informatica, Università di Salerno, Fisciano, Italy, 2010.

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

J. B. Chagas and M. Wagner. Ants can orienteer a thief in their robbery. *Operations Research Letters*, 48(6):708 – 714, 2020. ISSN 0167-6377.

J. B. Chagas, U. E. Silveira, A. G. Santos, and M. J. Souza. A variable neighborhood search heuristic algorithm for the double vehicle routing problem with multiple stacks. *International Transactions in Operational Research*, 27(1):112–137, 2020a.

J. B. C. Chagas and A. G. Santos. A branch-and-price algorithm for the double vehicle routing problem with multiple stacks and heterogeneous demand. In *16th International Conference on Intelligent Systems Design and Applications (ISDA)*, pages 921–934, Porto, Portugal, 2016. Springer.

J. B. C. Chagas and A. G. Santos. An effective heuristic algorithm for the double vehicle routing problem with multiple stack and heterogeneous demand. In *17th International Conference on Intelligent Systems Design and Applications (ISDA)*, pages 785–796, Delhi, India, 2017. Springer.

J. B. C. Chagas, U. E. F. Silveira, M. P. L. Benedito, and A. G. Santos. Simulated annealing metaheuristic for the double vehicle routing problem with multiple stacks. In *19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1311–1316, Rio de Janeiro, Brasil, 2016. IEEE.

J. B. C. Chagas, J. Blank, M. Wagner, M. J. F. Souza, and K. Deb. A non-dominated sorting based customized random-key genetic algorithm for the bi-objective traveling thief problem. *Journal of Heuristics*, 2020b. URL https://doi.org/10.1007/s10732-020-09457-7.

S. Chand and M. Wagner. Fast heuristics for the multiple traveling thieves problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 293–300. ACM, 2016.

I.-M. Chao, B. L. Golden, and E. A. Wasil. A fast and effective heuristic for the orienteering problem. *European Journal of Operational Research*, 88(3):475–489, 1996.

B. Cheng, Y. Yang, and X. Hu. Supply chain scheduling with batching, production and distribution. *International Journal of Computer Integrated Manufacturing*, 29(3):251–262, 2016.

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

J.-F. Côté, C. Archetti, M. G. Speranza, M. Gendreau, and J.-Y. Potvin. A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with multiple stacks. *Networks*, 60(4):212–226, 2009.

J.-F. Côté, C. Archetti, M. G. Speranza, M. Gendreau, and J.-Y. Potvin. A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with multiple

stacks. *Networks*, 60(4):212–226, 2012.

M. Črepinšek, S.-H. Liu, and M. Mernik. Exploration and exploitation in evolutionary algorithms: A survey. *ACM computing surveys (CSUR)*, 45(3):1–33, 2013.

G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management science*, 6 (1):80–91, 1959.

I. Das and J. E. Dennis. A closer look at drawbacks of minimizing weighted sums of objectives for pareto set generation in multicriteria optimization problems. *Structural optimization*, 14(1):63–69, 1997.

M. Dorigo and M. Birattari. *Ant colony optimization*. Springer, 2010.

M. Dorigo and C. Blum. Ant colony optimization theory: A survey. *Theoretical computer science*, 344(2-3):243–278, 2005.

M. Dorigo and G. Di Caro. Ant colony optimization: a new meta-heuristic. In *IEEE Congress on Evolutionary Computation (CEC)*, volume 2, pages 1470–1477. IEEE, 1999.

B. Eksioglu, A. V. Vural, and A. Reisman. The vehicle routing problem: A taxonomic review. *Computers & Industrial Engineering*, 57(4):1472–1483, 2009.

M. El Yafrani and B. Ahiod. Cosolver2b: an efficient local search heuristic for the travelling thief problem. In *2015 IEEE ACS 12th International Conference of Computer Systems and Applications (AICCSA)*, pages 1–5. IEEE, 2015.

M. El Yafrani and B. Ahiod. Population-based vs. single-solution heuristics for the travelling thief problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 317–324. ACM, 2016.

M. El Yafrani and B. Ahiod. Efficiently solving the traveling thief problem using hill climbing and simulated annealing. *Information Sciences*, 432:231–244, 2018.

L. M. Faêda and A. G. Santos. A genetic algorithm for the thief orienteering problem. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2020.

S. H. Fang, E. H. C. Lu, and V. S. Tseng. Trip recommendation with multiple user constraints by integrating point-of-interests and travel packages. In *IEEE 15th International Conference on Mobile Data Management*, volume 1, pages 33–42, July 2014.

H. Faulkner, S. Polyakovskiy, T. Schultz, and M. Wagner. Approximate approaches to the traveling thief problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 385–392. ACM, 2015.

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

F. Forster and A. Bortfeldt. A tree search procedure for the container relocation problem. *Computers & Operations Research*, 39(2):299–309, 2012.

N. K. Freeman, B. B. Keskin, and I. Çapar. Attractive orienteering problem with proximity and timing interactions. *European Journal of Operational Research*, 266(1): 354–370, 2018.

L. Galand and O. Spanjaard. Exact algorithms for owa-optimization in multiobjective spanning tree problems. *Computers & Operations Research*, 39(7):1540–1554, 2012.

G. Gamrath, D. Anderson, K. Bestuzheva, W.-K. Chen, L. Eifler, M. Gasse, P. Gemander, A. Gleixner, L. Gottwald, K. Halbig, G. Hendel, C. Hojny, T. Koch, P. Le Bodic, S. J. Maher, F. Matter, M. Miltenberger, E. Mühmer, B. Müller, M. E. Pfetsch, F. Schlösser, F. Serrano, Y. Shinano, C. Tawfik, S. Vigerske, F. Wegscheider, D. Weninger, and J. Witzig. The SCIP Optimization Suite 7.0. ZIB-Report 20-10, Zuse Institute Berlin, March 2020. URL http://nbn-resolving.de/urn:nbn:de:0297-zib-78023.

B. L. Golden, L. Levy, and R. Vohra. The orienteering problem. *Naval Research Logistics*, 34:307–318, 1987.

J. F. Gonçalves and M. G. Resende. Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics*, 17(5):487–525, 2011.

J. F. Gonçalves and M. G. Resende. A parallel multi-population biased random-key genetic algorithm for a container loading problem. *Computers & Operations Research*, 39(2):179–190, 2012.

J. F. Gonçalves and M. G. Resende. A biased random key genetic algorithm for 2d and 3d bin packing problems. *International Journal of Production Economics*, 145(2): 500–510, 2013.

J. F. Gonçalves and M. G. Resende. A biased random-key genetic algorithm for the unequal area facility layout problem. *European Journal of Operational Research*, 246(1):

86–107, 2015.

A. Gunawan, H. C. Lau, and P. Vansteenwegen. Orienteering problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research*, 255(2):315 – 332, 2016. ISSN 0377-2217.

G. Gutin and A. P. Punnen. *The traveling salesman problem and its variations*, volume 12. Springer Science & Business Media, 2006.

P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3):449–467, 2001.

P. Hansen and N. Mladenović. Variable neighborhood search. In *Search methodologies*, pages 313–337. Springer, 2014.

A. Hottung, S. Tanaka, and K. Tierney. Deep learning assisted heuristic tree search for the container pre-marshalling problem. *Computers & Operations Research*, 113:104781, 2020.

F. Hutter, Y. Hamadi, H. H. Hoos, and K. Leyton-Brown. Performance prediction and automated tuning of randomized and parametric algorithms. In *Principles and Practice of Constraint Programming*, pages 213–228. Springer, 2006.

L. M. Hvattum, G. Tirado, and Á. Felipe. The double traveling salesman problem with multiple stacks and a choice of container types. *Mathematics*, 8(6):979, 2020.

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

M. Iori and J. Riera-Ledesma. Exact algorithms for the double vehicle routing problem with multiple stacks. *Computers & Operations Research*, 63:83–101, 2015.

B. Jin, W. Zhu, and A. Lim. Solving the container relocation problem by an improved greedy look-ahead heuristic. *European Journal of Operational Research*, 240(3):837–847, 2015.

K. Krishnamoorthy. *Handbook of statistical distributions with applications*. CRC Press, 2016.

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

E. Lalla-Ruiz, J. L. González-Velarde, B. Melián-Batista, and J. M. Moreno-Vega. Biased random key genetic algorithm for the tactical berth allocation problem. *Applied Soft Computing*, 22:60–76, 2014.

M. López-Ibáñez, L. P. Cáceres, J. Dubois-Lacoste, T. Stützle, and M. Birattari. The irace package: User guide. *IRIDIA, Université Libre de Bruxelles, Belgium, Tech. Rep. TR/IRIDIA/2016-004*, 2016a.

M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016b.

L. Lovász. *Combinatorial problems and exercises*, volume 361. American Mathematical Soc., 2007.

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

S. Maher, M. Miltenberger, J. P. Pedroso, D. Rehfeldt, R. Schwarz, and F. Serrano. PySCIPOpt: Mathematical programming in python with the SCIP optimization suite. In *Mathematical Software – ICMS 2016*, pages 301–307. Springer International Publishing, 2016. doi: 10.1007/978-3-319-42432-3_37.

A. Maity and S. Das. Efficient hybrid local search heuristics for solving the travelling thief problem. *Applied Soft Computing*, page 106284, 2020.

R. T. Marler and J. S. Arora. The weighted sum method for multi-objective optimization: new insights. *Structural and multidisciplinary optimization*, 41(6):853–862, 2010.

S. Martello, D. Pisinger, and P. Toth. Dynamic programming and strong bounds for the 0-1 knapsack problem. *Management Science*, 45(3):414–424, 1999.

M. S. R. Martins, M. El Yafrani, M. R. B. S. Delgado, M. Wagner, B. Ahiod, and R. Lüders. Hseda: A heuristic selection approach based on estimation of distribution algorithm for the travelling thief problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, page 361–368. Association for Computing Machinery, 2017.

Y. Mei, X. Li, and X. Yao. On investigation of interdependence between sub-problems of the TTP. *Soft Computing*, 20(1):157–172, 2014.

M. Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

C. Moon, J. Kim, G. Choi, and Y. Seo. An efficient genetic algorithm for the traveling salesman problem with precedence constraints. *European Journal of Operational Research*, 140(3):606–617, 2002.

M. Namazi, C. Sanderson, M. A. H. Newton, and A. Sattar. A cooperative coordination solver for travelling thief problems, 2019. arXiv e-print, available at https://arxiv.org/abs/1911.03124.

F. Neumann, S. Polyakovskiy, M. Skutella, L. Stougie, and J. Wu. A fully polynomial time approximation scheme for packing while traveling. In Y. Disser and V. S. Verykios, editors, *Algorithmic Aspects of Cloud Computing*, pages 59–72. Springer, 2019. ISBN 978-3-030-19759-9.

H. L. Petersen. *Decision Support for Planning of Multimodal Transportation with Multiple Objectives*. PhD thesis, Technical University of Denmark (DTU), 2009.

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

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

M. Pinedo. *Scheduling*, volume 29. Springer, 2012.

T. Pinto, C. Alves, and J. Valério de Carvalho. Variable neighborhood search algorithms for the vehicle routing problem with two-dimensional loading constraints and mixed linehauls and backhauls. *International Transactions in Operational Research*, 2018. Available at: https://doi.org/10.1111/itor.12509.

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

S. Polyakovskiy and F. Neumann. Packing while traveling: Mixed integer programming for a class of nonlinear knapsack problems. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*,

pages 332–346. Springer, 2015.

S. Polyakovskiy, M. R. Bonyadi, M. Wagner, Z. Michalewicz, and F. Neumann. A comprehensive benchmark set and heuristics for the traveling thief problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 477–484. ACM, 2014.

Y. Qi, Z. Hou, H. Li, J. Huang, and X. Li. A decomposition based memetic algorithm for multi-objective vehicle routing problem with time windows. *Computers & Operations Research*, 62:61–77, 2015.

R. Ramanathan. Abc inventory classification with multiple-criteria using weighted linear optimization. *Computers & Operations Research*, 33(3):695–700, 2006.

G. Reinelt. Tsplib—a traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991.

M. G. Resende. Biased random-key genetic algorithms with applications in telecommunications. *Top*, 20(1):130–153, 2012.

A. H. Sampaio and S. Urrutia. New formulation and branch-and-cut algorithm for the pickup and delivery traveling salesman problem with multiple stacks. *International Transactions in Operational Research*, 24:77–98, 2016.

A. G. Santos and J. B. C. Chagas. The thief orienteering problem: Formulation and heuristic approaches. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, pages 1191–1199, Rio de Janeiro, Brasil, 2018. IEEE.

M. W. Savelsbergh and M. Sol. The general pickup and delivery problem. *Transportation Science*, 29(1):17–29, 1995.

U. E. F. Silveira, M. P. L. Benedito, and A. G. Santos. Heuristic approaches to double vehicle routing problem with multiple stacks. In *15th International Conference on Intelligent Systems Design and Applications (ISDA)*, pages 231–236, Marrakesh, Marocco, 2015. IEEE.

A. L. Souza, J. B. C. Chagas, P. H. V. Penna, and M. J. F. Souza. A late acceptance hill-climbing heuristic algorithm for the double vehicle routing problem with multiple stacks and heterogeneous demand. In *18th International Conference on Intelligent Systems Design and Applications (ISDA)*, pages 761–771, vellore, India, 2018. Springer.

I. P. Stanimirovic, M. L. Zlatanovic, and M. D. Petkovic. On the linear weighted sum method for multi-objective optimization. *Facta Acta Universitatis*, 26(4):49–63, 2011.

T. Stützle and H. H. Hoos. Max–min ant system. *Future generation computer systems*, 16 (8):889–914, 2000.

P. E. Sweeney and E. R. Paternoster. Cutting and packing problems: a categorized, application-orientated research bibliography. *Journal of the Operational Research Society*, 43(7):691–706, 1992.

E.-G. Talbi. *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons, 2009.

R. F. Toso and M. G. Resende. A C++ application programming interface for biased random-key genetic algorithms. *Optimization Methods and Software*, 30(1):81–93, 2015.

P. Toth and S. Martello. *Knapsack problems: Algorithms and computer implementations*. Wiley, 1990.

P. Toth and D. Vigo. *Vehicle routing: problems, methods, and applications*. SIAM, 2014.

F. Tricoire, K. F. Doerner, R. F. Hartl, and M. Iori. Heuristic and exact algorithms for the multi-pile vehicle routing problem. *OR spectrum*, 33(4):931–959, 2011.

P. Vansteenwegen, W. Souffriau, and D. Van Oudheusden. The orienteering problem: A survey. *European Journal of Operational Research*, 209(1):1–10, 2011.

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

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

T. Vidal, T. G. Crainic, M. Gendreau, and C. Prins. A unified solution framework for multi-attribute vehicle routing problems. *European Journal of Operational Research*, 234(3):658–673, 2014.

T. Vidal, G. Laporte, and P. Matl. A concise guide to existing and emerging vehicle routing problem variants. *European Journal of Operational Research*, 286(2):401–416, 2020.

M. Wagner. Stealing items more efficiently with ants: A swarm intelligence approach to the travelling thief problem. In M. Dorigo, M. Birattari, X. Li, M. López-Ibáñez, K. Ohkura, C. Pinciroli, and T. Stützle, editors, *Swarm Intelligence*, pages 273–281. Springer, 2016.

M. Wagner, K. Bringmann, T. Friedrich, and F. Neumann. Efficient optimization of many objectives by approximation-guided evolution. *European Journal of Operational Research*, 243(2):465 – 479, 2015.

M. Wagner, M. Lindauer, M. Mısır, S. Nallaperuma, and F. Hutter. A case study of algorithm selection for the traveling thief problem. *Journal of Heuristics*, 24(3): 295–320, Jun 2018.

G. Wang. Integrated supply chain scheduling of procurement, production, and distribution under spillover effects. *Computers & Operations Research*, page 105105, 2020.

L. Wei, Z. Zhang, and A. Lim. An adaptive variable neighborhood search for a heterogeneous fleet vehicle routing problem with three-dimensional loading constraints. *IEEE Computational Intelligence Magazine*, 9(4):18–30, 2014.

L. Wei, Z. Zhang, D. Zhang, and A. Lim. A variable neighborhood search for the capacitated vehicle routing problem with two-dimensional loading constraints. *European Journal of Operational Research*, 243(3):798–814, 2015.

J. Wu, M. Wagner, S. Polyakovskiy, and F. Neumann. Exact approaches for the travelling thief problem. In *Simulated Evolution and Learning*, pages 110–121. Springer, 2017.

J. Wu, S. Polyakovskiy, M. Wagner, and F. Neumann. Evolutionary computation plus dynamic programming for the bi-objective travelling thief problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 777–784. ACM, 2018.

M. E. Yafrani, S. Chand, A. Neumann, B. Ahiod, and M. Wagner. Multi-objectiveness in the single-objective traveling thief problem. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '17, pages 107–108. ACM, 2017.

L. Zadeh. Optimality and non-scalar-valued performance criteria. *IEEE Transactions on Automatic Control*, 8(1):59–60, 1963.

Q. Zhang and H. Li. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731, 2007.

E. Zitzler and L. Thiele. Multiobjective optimization using evolutionary algorithms—a comparative case study. In *International Conference on Parallel Problem Solving from Nature*, pages 292–301. Springer, 1998.

E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. Da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, 2003.

W. Zouari, I. Alaya, and M. Tagina. A new hybrid ant colony algorithms for the traveling thief problem. In *Genetic and Evolutionary Computation Conference (GECCO) Companion*, page 95–96. ACM, 2019. ISBN 9781450367486.