UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

CARLO SULZBACH SARTORI

The Pickup and Delivery Problem with Time Windows: Algorithms, Instances, and Solutions

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Computer Science

Advisor: Prof. Dr. Luciana Salete Buriol

Porto Alegre
March 2019

CIP — CATALOGING-IN-PUBLICATION

Sartori, Carlo Sulzbach

The Pickup and Delivery Problem with Time Windows: Algorithms, Instances, and Solutions / Carlo Sulzbach Sartori. – Porto Alegre: PPGC da UFRGS, 2019.

101 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2019. Advisor: Luciana Salete Buriol.

1. Pickup and delivery. 2. Time windows. 3. Metaheuristic. 4. Mathematical programming. 5. Instance generation. I. Buriol, Luciana Salete. II. Título.

“It is a mistake to think you can solve any major problems just with potatoes.”

— DOUGLAS ADAMNS. LIFE, THE UNIVERSE AND EVERYTHING.

ACKNOWLEDGEMENT

First, I would like to thank my advisor, Luciana Buriol, for all the support in more than four years of work. Everything she has taught me contributed to my growth both personally and professionally. Besides, I truly appreciate her kindness, patience, and optimism in moments of nothing but despair.

I thank my father, Jacir Sartori, for the unconditional support and opportunities he provided throughout my entire life – not to mention the multiple advice in my earliest moments of doubt. In the same way, I own a lot of this thesis to my beloved girlfriend, Tainara Silva, who has helped, supported, and cheered me on countless occasions.

The journey was certainly easier to endure thanks to my friends from the Scout Movement. There are many of course, but a special mention goes to Ariel, Azulado, Bia, Dé, Guto, Orlando, Raquel, and Romeu. They are basically my second family. In addition, I thank my hard-to-meet friends from high school, who despite the lack of meetings have accompanied me during this period as well.

The whole laboratory 207 deserves a special acknowledgment. It was really rewarding to be part of the group and meet numerous incredible colleagues. Risking to forget someone, I mention some members: Alberto, Alex, Artur, Gabriel, Henrique, Marcelo, Tadeu, Toni, Victoria, and Wesley. An additional thanks to professor Marcus Ritt, who has taught me about optimization and research in many ways. In this academic context, I thank many other colleagues I have met at INF-UFRGS, who certainly have contributed in some way to my development.

I appreciate the valuable comments and notes regarding this thesis provided by professors Marcus Ritt, Olinto Araújo, and Thibaut Vidal. They were taken into consideration in this final version of the complete text.

At last, I would like to acknowledge the financial support of CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico), and INF-UFRGS for the available infrastructure and support to conduct my research.

ABSTRACT

This work considers the Pickup and Delivery Problem with Time Windows. It is a hard combinatorial optimization problem that generalizes a number of vehicle routing problem and finds applications in courier and dial-a-ride services. The objective is to construct a good set of vehicle routes to service transportation requests from pickup to delivery locations while respecting vehicle capacity and service time constraints. To tackle the problem, we propose a hybrid method combining heuristic components with a mathematical programming formulation – a technique typically referred to as matheuristic. Furthermore, a method to generate instances for the problem based on open data and realistic travel times is provided. A new set of benchmarks is proposed considering the new generation procedure. Computational experiments demonstrate that the proposed method works well on a standard benchmark set of instances for which it found a number of new best-known solutions. On the other hand, results for the new set of instances are in disagreement with the observations for the standard benchmarks and demonstrate relevant differences between the two testbeds. The work presents a series of analyses and discussions regarding the experiments, main components, and variations of the matheuristic algorithm.

Keywords: Pickup and delivery. Time windows. Metaheuristic. Mathematical programming. Instance generation.

O Problema de Coleta e Entrega com Janelas de Tempo: Algoritmos, Instâncias, e Soluções

RESUMO

Este trabalho estuda o Problema de Coleta e Entrega com Janelas de Tempo. Trata-se de um difícil problema de otimização combinatória que generaliza problemas de roteamento de veículos, possuindo aplicações em serviços de entrega e transporte de passageiros. O objetivo é definir o melhor conjunto de rotas de veículos que atendam requisições de transporte entre locais de coleta e entrega, respeitando ao mesmo tempo a capacidade dos veículos e as restrições de tempos de serviço. Para resolver o problema, nós propomos um método híbrido combinando componentes heurísticos com programação matemática – uma técnica geralmente chamada de *matheurística*. Além disso, uma metodologia para gerar instâncias do problema que são baseadas em dados abertos e tempos de viagem reais é proposta. Um novo conjunto de testes foi gerado baseado nesta proposta. Experimentos computacionais demonstraram que o algoritmo proposto é capaz de obter bons resultados no conjunto padrão de testes da literatura, para o qual novas soluções também foram encontradas. Por outro lado, os resultados para o novo conjunto de instâncias diferem dos observados para o conjunto padrão e demonstram diferenças relevantes entre os dois conjuntos de teste. O trabalho apresenta Uma série de análises e discussões a respeito dos experimentos, principais componentes, e variações do algoritmo híbrido proposto.

Palavras-chave: Coleta e entrega. Janelas de tempo. Metaheurística. Programação matemática. Geração de instâncias.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AGES	Adaptive Guided Ejection Search
CPLEX	IBM ILOG CPLEX Optimization Studio
CO	Combinatorial Optimization
CVRP	Capacitated Vehicle Routing Problem
DARP	Dial-a-Ride Problem
ILS	Iterated Local Search
LAHC	Late Acceptance Hill-Climbing
LNS	Large Neighborhood Search
MILP	Mixed Integer Linear Programming
MP	Mathematical Programming
OR	Operations Research
OSM	Open Street Maps
OSRM	Open Source Routing Machine
PDPTW	Pickup and Delivery Problem with Time Windows
SP	Set Partitioning
TSP	Travelling Salesman Problem
VRP	Vehicle Routing Problem
VRPTW	Vehicle Routing Problem with Time Windows

CONTENTS

1 INTRODUCTION

According to Rodrigue, Comtois and Slack (2016), the growth of urban areas has increased the complexity of logistics of the land-based distribution of products. As a matter of fact, road networks are the dominant transport infrastructure of several countries and serve as the primary mode for freight transportation. For instance, in the United States of America, the Bureau of Transportation Statistics (2017) estimates that 63% of all freight transportation is performed by road. In the European Union, the European Commission (2016) estimates 49% for the same scenario. In Brazil, the Confederação Nacional do Transporte (2017) estimates 61%.

In order to cope with large logistic networks, researchers from many fields have contributed to reduce the overall cost and improve the quality of transportation services. One such field is Operations Research (OR), which applies analytical methods to find the best solutions to a problem, subject to side constraints. A well-known transportation model in OR is the *Vehicle Routing Problem* (VRP), first studied 60 years ago by Dantzig and Ramser (1959). The objective is to build optimal vehicle routes to supply geographically distributed customers, a recurrent problem of economic importance in modern society. In fact, the VRP is an active research topic, and several variants exist to model practical scenarios as presented by Toth and Vigo (2014).

The focus of this work is a variant called *Pickup and Delivery Problem with Time Windows*, which is constrained by vehicle capacity, time windows, and precedence relations between pickup and delivery pairs. In the following Section 1.1 the overall class of VRP problems is introduced, and variants are described. The motivations of this thesis are presented in Section 1.2. A list of the main research contributions is provided in Section 1.3. At last, an overview of the entire work is given in Section 1.4.

1.1 Vehicle Routing Problems

One of the fundamental problems in the VRP literature is the well-known Travelling Salesman Problem (TSP). In the TSP, a salesman has to visit a number of cities and return to its departure location. The decision version of the problem asks whether there exists a tour that visits all cities exactly once traveling a distance no longer than a parameter D. The problem was proven $\mathcal{NP}$-Complete by Karp (1972). The optimization version is a $\mathcal{NP}$-Hard problem which tries to find the minimum distance Hamiltonian cycle.

A generalization of the TSP is the VRP, in which multiple tours have to be constructed to visit all requests, generally due to side constraints that would be otherwise violated. Laporte (2007) defines a classical VRP as the *Capacitated Vehicle Routing Problem* (CVRP) – from the seminal work by Dantzig and Ramser (1959). In this variant, a set of customer demands has to be serviced by vehicle routes, which start and end at an origin depot. Each customer request has an associated demand of goods to be supplied by a vehicle, and each vehicle has a maximum capacity to carry goods from the depot, which should never be exceeded. The distance between customer locations is typically referred to as cost, and the cost of a route is the sum of all the costs to travel from one location to the other in that route. The objective of the problem is to find the set of routes attending all requests exactly once, such that the total cost of the routes is minimized.

Schrage (1981) studied the *Vehicle Routing Problem with Time Windows* (VRPTW), which is a generalization of the CVRP. In addition to the demand, each request has an associated time interval that indicates the earliest and latest time service can start – this interval is typically called *time window*. Every location has a service duration indicating how much time the service requires. There is also a cost and time associated with travelling between each pair of locations. Other constraints follow as in the CVRP. A common objective function is to search for routes that minimize, in lexicographic order, the number of vehicles used and the total cost of the routes (BRÄYSY; GENDREAU, 2005).

The variation studied in this work is a generalization of the VRPTW called *Pickup and Delivery Problem with Time Windows* (PDPTW), first presented by Dumas, Desrosiers and Soumis (1991). In this scenario, goods are transported between customer locations, rather than from the depot to the customers as in the CVRP and VRPTW. A customer request is a pair of locations: a pickup and a delivery. Vehicles must transport goods from the pickup to the corresponding delivery, respecting vehicle capacities and time windows of each location. A common objective is the same as the VRPTW, minimize first the number of vehicles used, and then the total cost of the routes.

All the referred variations of the VRP, namely, the CVRP, VRPTW, and PDPTW, are $\mathcal{NP}$-Hard optimization problems because they contain the TSP as a particular case. In other words, there is no known polynomial time algorithm capable of solving those problems, unless $\mathcal{P} = \mathcal{NP}$. Nonetheless, the TSP can already be solved to optimality for cases with thousands of locations (APPLEGATE et al., 2006), and the CVRP for cases with hundreds of requests (PECIN et al., 2017). Despite these results, the most common solution methods in the literature of VRPs are still heuristic-based techniques.

1.2 Motivation

The PDPTW generalizes many practical scenarios. Applications of the model include product delivery and collection (LAPORTE, 2007), courier services (SHEN et al., 1995), dial-a-ride problems (WILSON; WEISSBERG; HAUSER, 1976), airline scheduling and bus routing (NANRY; BARNES, 2000), and ship routing (CHRISTIANSEN et al., 2007). Future trends may include the routing of autonomous vehicles, such as the trucks of Embark[1] recently tested in the United States of America.

Following the number of applications, the PDPTW is a good model to optimize and improve certain aspects of the previously mentioned use of road transportation. These improvements can have a significant impact on both companies and customers. For example, according to Rodrigue, Comtois and Slack (2016), the transportation cost can account for up to 10% of the total cost of a product. Hasle, Lie and Quak (2007) note that vehicle routing tools are typically described as offering savings of 5%-30%, although even cost reductions of 2% are substantial considering the amount of volumes transported.

Moreover, the International Energy Agency (2018) has reported that road transportation is responsible for almost 19% of the CO_2 emissions worldwide. Piecyk and McKinnon (2010) had forecast that one factor to help reduce the carbon footprints of road transportation services by 2020 would be the use of computerized routing systems, due to the usual objectives employed in routing to reduce fleet usage and minimize travel times, and the modest investment and risk carried for companies to apply such systems.

Additionally, both delays and unnecessary waiting times can be avoided by better scheduling the order of visits, leading to increased customer satisfaction. Other benefits of computerized solution methods for the PDPTW and VRP-type problems, reviewed by Sutcliffe and Board (1991) include automatic routing with less interference of people and better and faster-produced routes.

Other than the possible benefits of the PDPTW and VRPs, it is our interest to study the computational aspects of solving these models. The PDPTW is still a hard problem to be solved optimally even for a few hundred requests. Hence, our focus is on improving heuristic solution methods for the problem. Furthermore, we want to provide analyses of specific heuristic components and study how they behave in different scenarios, extending previous works from the literature. There is also a lack of realistic testing scenarios (e.g., instances) for the PDPTW we want to address.

[1] Embark®: Self-Driving Semi Trucks, <https://embarktrucks.com/>, last accessed 03-02-2019.

1.3 Research Objectives and Contributions

The main objectives of this work are to extend and improve the current *state-of-the-art* metaheuristic methods for the PDPTW. To that end, we propose a hybrid metaheuristic algorithm integrated with a Mathematical Programming (MP) component. This combination is often referred to as a *matheuristic* (MANIEZZO; STÜTZLE; VOSS, 2009). The MP module is a Set Partitioning (SP) formulation of the PDPTW, while other metaheuristic modules are based on previous works from the PDPTW literature. A preliminary study about the proposed matheuristic was published at the *IX International Conference on Computational Logistics* (SARTORI; BURIOL, 2018).

Through computational experiments, we analyze the performance of the components and of the whole matheuristic in the context of the PDPTW. The results are also used in statistical tests that demonstrate improvements over previous works from the literature. Additionally, as a result of our method, a number of new best solutions have been found for the standard benchmark instance set of the problem.

Furthermore, we propose a method to generate new instances for the PDPTW and other VRP variants as well. The aim is to build instances with real-world data on locations and travel times. A set composed of 300 new instances was generated according to the new method. All the data used is freely available. In addition, we provide an open source software tool to generate new instances as a way to foster reproducibility and contribute to the VRP community.

1.4 Overview

The thesis is organized as follows. In Chapter 2 the problem is described, and a mathematical formulation is provided. Related works are reviewed in Chapter 3, considering the metaheuristics used in this work, as well as the main methods proposed to solve the PDPTW. The proposed algorithm is described in Chapter 4. In Chapter 5 we describe the current instances of the problem and a method to generate realistic instances. Chapter 6 provides analysis of the computational experiments and results. The work is concluded in Chapter 7.

2 PROBLEM DESCRIPTION

The purpose of this chapter is to present a detailed description of the Pickup and Delivery Problem with Time Windows. In Section 2.1 we describe an instance of the problem, while in Section 2.2 constraints and solution evaluation are defined. A Mixed Integer Linear Programming (MILP) formulation is presented in Section 2.3.

2.1 Instance Information

An instance of the PDPTW is defined on a directed graph $G = (V, A)$, where $V = \{0, 1, \ldots, 2n\}$ is the set of all $2n + 1$ vertices, and A the set of all arcs connecting pairs of vertices. Define subset $P = \{1, \ldots, n\}$ as the set of n pickup locations, and subset $D = \{n + 1, \ldots, 2n\}$ as the set of n corresponding delivery locations (pickup i is associated to delivery $n + i$). Let vertex 0 be the depot where routes start and end. Then, $V = P \cup D \cup \{0\}$. Additionally, for each arc $(i, j) \in A,\ i, j \in V$, there is an associated cost c_{ij} and time t_{ij} to travel from vertex i to j. It is assumed that travel times are non-negative and respect the triangular inequality $t_{ik} \leq t_{ij} + t_{jk},\ \forall\ i, j, k \in V$.

In the PDPTW, a request is a unique pair $(p,\ n + p),\ p \in P,\ n + p \in D$, and products have to be transported from p to $n + p$. Moreover, each vertex $i \in V$ has an associated time window $[e_i, l_i]$, where e_i and l_i are the time limits to start service at location i. The duration of service at location i is denoted by s_i, and the demand is denoted by q_i. For pickup locations, the demand is of $q_p \geq 0$ products to be collected, whereas the corresponding delivery location has demand of $q_{n+p} = -q_p$ goods to be delivered.

By definition, all routes start at time zero, and it is assumed that $e_0 = 0$. Besides, every route has a maximum total time of $l_0 = H$, the scheduling horizon. The demand and service time of the depot are $q_0 = 0$ and $s_0 = 0$. A fleet K of homogeneous vehicles is available at the depot, and each vehicle can transport up to Cap unities of goods.

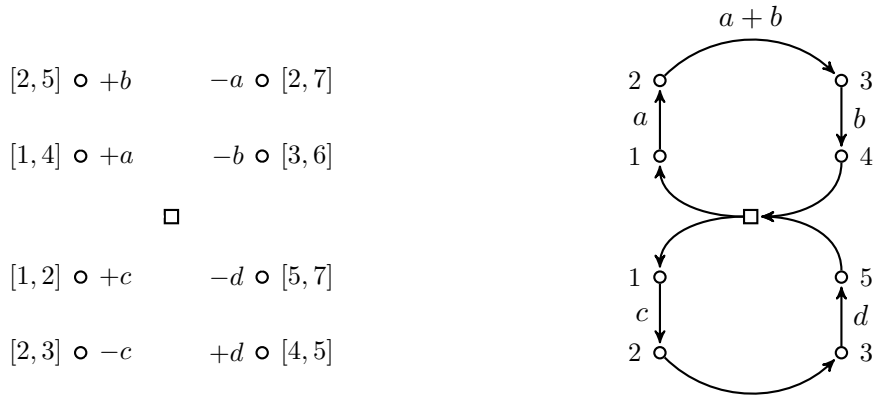
2.2 Constraints and Objective Function

A solution to the PDPTW is a set of routes $s = \{r_1, \ldots, r_m\}$. Each route $r \in s$ is a sequence of locations to be visited by a vehicle, denoted as $r = (v_0, v_1, \ldots, v_h, v_{h+1})$, where $v_k \in P \cup D, k = 1, \ldots, h$, and $v_0 = v_{h+1} = 0$ the depot. The cost of a route r is the

sum of the costs of all arcs connecting vertices in r, denoted by $C(r) = \sum_{i=0}^{h} c_{v_i v_{i+1}}$. The cost of a solution s is the sum of all the costs of its routes, denoted by $C(s) = \sum_{r \in s} C(r)$. Figure 2.1 presents an example of a PDPTW instance and a feasible solution.

The time window constraints allow a vehicle to arrive at location $i \in V$ before e_i and wait to operate, but a vehicle cannot arrive later than l_i. Even though a vehicle may perform as many consecutive pickups as necessary, it must respect the maximum capacity constraint. Therefore if a vehicle is full, it must perform a delivery before another pickup operation. Furthermore, each $p \in P$ must be visited before $n + p$, and by the same vehicle. These are the *precedence* and *pairing* constraints, respectively. At last, all customer locations in $P \cup D$ have to be visited exactly once.

Figure 2.1: Example of a PDPTW instance and solution. (a) An instance with 8 locations, their time windows [min, max] and demands. Pickup location with demand $+a$ is associated with delivery $-a$. Travel times and costs are unitary. There is no service duration. The square node is the depot; (b) A solution to the PDPTW where each node label is the exact time a vehicle leaves the location, and each arc label is the load carried by that vehicle. Note that load d reaches its destination in time 4, but it has to wait until time 5 to unload, due to the time window constraint in $-d$.



(a) A PDPTW instance.

(b) A feasible solution.

Source: The Author.

The objective function of the PDPTW is to minimize the number of vehicles used and the total cost of the routes. These objectives are evaluated in the lexicographic order presented in Function (2.1).

$$f(s) = (\, |s| \, , \, C(s) \,) \tag{2.1}$$

The first term is the number of routes, whereas the second term is the sum of the cost of all routes in solution s. For instance, the solution s in Figure 2.1(b) would have the evaluation $f(s) = (2 \, , \, 10)$, because there are two routes and 10 arcs of unitary cost.

2.3 Mathematical Programming Formulation

In addition to the information of Sections 2.1 and 2.2, we need to define decision variables to correctly formulate the PDPTW. The presented notation and formulation are based on those of Ropke and Cordeau (2009) for the same problem. Furthermore, we duplicate the depot to simplify the formulation, defining an origin depot 0, from where routes start, and a destination depot $2n+1$, where all routes end. Hence, we redefine $V = P \cup D \cup \{0, 2n+1\}$, with P and D the set of pickups and deliveries respectively.

Let x_{ijk} be a binary decision variable that assumes value 1 when vehicle $k \in K$ travels from location $i \in V$ to location $j \in V$, and 0 otherwise. Define real variables B_{ik} to be the moment in time vehicle $k \in K$ starts service at location $i \in V$, and variables Q_{ik} the amount of load being carried by vehicle $k \in K$ when leaving location $i \in V$. Then, the PDPTW can be formulated by the following MILP.

$$\textbf{minimize} \quad \sum_{i \in V} \sum_{j \in V} \sum_{k \in K} c_{ij} x_{ijk} \tag{2.2}$$

subject to

$$\sum_{k \in K} \sum_{j \in V} x_{ijk} = 1 \qquad \forall\, i \in P \tag{2.3}$$

$$\sum_{j \in V} x_{ijk} - \sum_{j \in V} x_{(n+i)jk} = 0 \qquad \forall\, i \in P,\ k \in K \tag{2.4}$$

$$\sum_{j \in V} x_{jik} - \sum_{j \in V} x_{ijk} = 0 \qquad \forall\, i \in P \cup D,\ k \in K \tag{2.5}$$

$$\sum_{j \in V} x_{0jk} = 1 \qquad \forall\, k \in K \tag{2.6}$$

$$\sum_{i \in V} x_{i(2n+1)k} = 1 \qquad \forall\, k \in K \tag{2.7}$$

$$B_{jk} \geq B_{ik} + t_{ij} + s_i - M(1 - x_{ijk}) \qquad \forall\, i, j \in V,\ k \in K \tag{2.8}$$

$$Q_{jk} \geq Q_{ik} + q_j - M(1 - x_{ijk}) \qquad \forall\, i, j \in V,\ k \in K \tag{2.9}$$

$$B_{(n+i)k} \geq B_{ik} + t_{i(n+i)} + s_i \qquad \forall\, i \in V,\ k \in K \tag{2.10}$$

$$e_i \leq B_{ik} \leq l_i \qquad \forall\, i \in V,\ k \in K \tag{2.11}$$

$$\max\{0, q_i\} \leq Q_{ik} \leq \min\{\mathsf{Cap}, \mathsf{Cap} + q_i\} \qquad \forall\, i \in V,\ k \in K \tag{2.12}$$

$$x_{ijk} \in \{0, 1\} \qquad \forall\, i, j \in V,\ k \in K \tag{2.13}$$

The objective function in Equation (2.2) is to minimize the sum of the travelled arcs in the input graph G. Note that the lexicographic objective function of Section 2.2 can be modeled by assigning a large constant weight to the arcs leaving the initial depot, that is, $c_{0j}\ \forall j \in V \backslash \{2n+1\}$. Constraints (2.3) guarantee that all requests will be serviced exactly once. The *pairing condition* is guaranteed by constraints (2.4) so that the vehicle that operates at the pickup location is the same to operate at the corresponding delivery. Flow conservation constraints are stated in constraints (2.5), that is, the vehicle k which arrives at a location i, must be the same that leaves that same location. Constraints (2.6) and (2.7) define that all vehicles must leave the origin depot and reach the destination depot – this has no impact on solution cost, as long as the cost to travel directly from the origin to the destination depot is zero, that is, $c_{0(2n+1)} = 0$.

Consistency of both time and load variables is guaranteed by constraints (2.8) and (2.9), respectively. A large constant M is needed to make such constraints linear, although the value of M can be tightened as proposed by Ropke, Cordeau and Laporte (2007). The so-called *precedence constraints* are stated in constraints (2.10) by forcing that the time to visit a delivery location must be greater than that to visit the corresponding pickup location. Start of service within the limits of time windows is assured by constraints (2.11), and capacity constraints of the vehicles are guaranteed by constraints (2.12). At last, constraints (2.13) define the boundary for the value of x binary variables.

3 RELATED WORK

In this chapter, we present the scientific literature related to this thesis. Since our proposed method is a hybrid metaheuristic, our focus in this review will be in metaheuristic approaches and related topics. In Section 3.1, we present an overview of the methods for Combinatorial Optimization (CO) problems used in this work. The main approaches for the PDPTW are discussed in Section 3.2, both exact and heuristic.

3.1 Metaheuristics for Combinatorial Optimization Problems

In general terms, metaheuristics are solution methods used to explore a possibly huge search space of a CO problem, using specific strategies to avoid or overcome the traps of local optimal solutions. The primary advantage to the use of metaheuristics is their easy definition and good performance in exploring the search space of $\mathcal{NP}$-Hard problems, providing short execution times and good quality solutions.

A multitude of metaheuristics has been proposed over the years to tackle mainly CO problems arising in OR, engineering, and sciences. Among these methods, we review three of them which are used in this work. The first is *Iterated Local Search*, the second is *Large Neighborhood Search*, and the third is the so-called *Matheuristic*. For a wider overview of metaheuristics and related topics, the reader is referred to the book of Gendreau and Potvin (2010).

3.1.1 Iterated Local Search

The *Iterated Local Search* (ILS) metaheuristic is a *stochastic local search* method applied to solve hard combinatorial optimization problems. In a nutshell, the idea is to sequentially reach local optimal solutions and use a perturbation scheme to escape local optima, but keeping a certain amount of information from previously found solutions. Its concept is somewhat related to the Random Restart method, but in this case, a solution is created from scratch when a local optimum is found, which means it does not keep information from one iteration to the next. Furthermore, according to Lourenço, Martin and Stützle (2010), ILS is able to outperform Random Restart methods in most cases.

In order to design an ILS method, four modules are required as presented in Al-

gorithm 1. The first is an *initial solution generator*, in line 1 of the algorithm, which creates an initial solution s. The second module is the *local search*, which is responsible for improving a given solution in lines 2 and 5, returning a new solution. In line 4 of the algorithm, a *perturbation* in the current solution is performed to escape a local optimum and diversify the search. At last, the fourth component is in line 6, the *acceptance criterion*, which chooses to accept the current solution, or a previous one to continue the search. Lines 7-9 update the best solution found s^* if it has been improved.

Algorithm 1 Iterated Local Search

```
1: s ← initial_solution()
2: s, s* ← local_search(s)
3: repeat
4:     s' ← perturbation(s)
5:     s' ← local_search(s')
6:     s ← accept(s*, s, s')
7:     if f(s) is better than f(s*) then
8:         s* ← s
9:     end if
10: until stopping criterion is met
11: return s*
```

Lourenço, Martin and Stützle (2010) note some facts about the main modules. For instance, the solution constructor can be any of a multitude of methods such as greedy constructive algorithm, or a completely random solution generator. The authors demonstrate that in short execution times, using a greedy constructor provides better results, but for longer periods, a random start may reach the same solution quality. Moreover, the local search has not to be an actual local search, in fact, a common practice in recent research is the use of other metaheuristics as the local search module. Some examples include the use of Iterated Tabu Search (CORDEAU; MAISCHBERGER, 2012) and Iterated Variable Neighborhood Descent (SUBRAMANIAN; UCHOA; OCHI, 2013). Despite its actual strategy and implementation, the so-called local search component should be able to reach a local optimum in the search space from a given starting solution.

The perturbation mechanism, on the other hand, has to disrupt the solution just enough for it to move from the current basin of attraction, but not so much as to lose the actual characteristics of the solution; otherwise the method falls back to a simple random restart heuristic. Additionally, the local search should not be able to easily undo the perturbation effect, as to avoid returning to the previous solution. At last, the criterion for accepting solutions can also be one from many, such as always taking the best solution

or the current solution, although the former focuses too much on intensification and the latter on diversification. Other options are to combine strategies and use an acceptance based on previous solutions, number of iterations, or parameter.

Applications of ILS include the TSP (MERZ; HUHSE, 2008), the Quadratic Assignement Problem (STÜTZLE, 2006), the Permutation Flowshop Problem (RUIZ; STÜTZLE, 2007), and the Bin Packing Problem with Conflicts (CAPUA et al., 2018). Particularly for VRPs, there are applications for the VRP with Heterogenous fleet (PENNA et al., 2017), the VRP with cross-docking (MORAIS; MATEUS; NORONHA, 2014), and for a class of VRPs (SUBRAMANIAN; UCHOA; OCHI, 2013).

3.1.2 Large Neighborhood Search

The *Large Neighborhood Search* (LNS) metaheuristic was first proposed by Shaw (1998) in the context of VRPs. Its underlying idea is to perform large modifications in a solution in order to direct the search in the solution space, as opposed to local search methods that only change small portions of a solution at a time. Alternative names for LNS are *Ruin and Repair* and *Ruin and Recreate*.

According to Pisinger and Ropke (2010) an LNS method is defined implicitly by two components. The first one is a *destroy* method, which takes a solution as input and removes part of its elements. A destroy method is usually stochastic so that at each iteration of the metaheuristic, different elements are removed, even if the same solution is used. The second component is the *repair* method, that takes a previously destroyed solution and repairs it by reinserting elements, creating a new full solution, possibly better than the best found and in a whole new area of the search space.

Algorithm 2 outlines the pseudocode of LNS. It takes as input an initial feasible solution s, which is used to initialize the best-found solution s^* in line 1. Then, the metaheuristic iteratively applies a number of operations (lines 2–9), until a given stopping condition is met. In line 3, the current solution s is destroyed, that is, it has some elements removed creating a partial solution s'. The solution s' is repaired in line 4, which means that removed elements are reinserted. A solution between s and the new s' is chosen to guide the search in the next iteration according to some criteria in line 5, while the best solution found s^* is updated if improved in line 7.

Comparing Algorithms 1 and 2, it is possible to note that ILS and LNS have some similarities, at least when it comes to their simplified pseudocodes. However, their prac-

Algorithm 2 Large Neighborhood Search

Input: A feasible initial solution s

```
1:  s* ← s
2:  repeat
3:      s' ← destroy(s)
4:      s' ← repair(s')
5:      s ← accept(s, s')
6:      if f(s) is better than f(s*) then
7:          s* ← s
8:      end if
9:  until stopping criterion is met
10: return s*
```

tical implementations can present a lot of differences. The ILS is commonly used with local search methods that modify few elements of the solutions, while LNS disrupts lots of elements from the solution. Nevertheless, the methods can be combined, as it is proposed in this thesis, to provide an even better performing algorithm.

The main applications of LNS have been for VRPs as the metaheuristic can easily remove and reinsert many requests at once, which for many variants of routing problems performs much better than small modifications. Examples are the seminal work of Shaw (1998), the application in the context of VRPTW (BENT; HENTENRYCK, 2004) and also for the PDPTW (BENT; HENTENRYCK, 2006; ROPKE; PISINGER, 2006; CURTOIS et al., 2017) – the latter will be described in the next section.

3.1.3 Matheuristics

In the scientific literature, the hybridization of metaheuristics and MP is often called *matheuristic* (BOSCHETTI et al., 2009; MANIEZZO; STÜTZLE; VOSS, 2009). According to Boschetti et al. (2009) matheuristics can be used in many fields such as optimization, simulation, and process control. The goal is to exploit characteristics from the mathematical models of the problems while constraining the search space and execution time in a heuristic manner, in order to achieve good quality solutions.

A survey on matheuristics for general CO problems is presented by Ball (2011), which also includes a classification of methods according to how the MP component is embedded in the algorithm. A total of four classes are discriminated. The first class is on decomposition approaches, which break down a problem in sequences of subproblems that are solved to optimality employing a mathematical model. The second class

is composed of improvement methods that integrate MP in the metaheuristic to improve solutions found by the heuristic, similar to a local search component. The third class comprises matheuristic techniques that generate approximate near-optimal solutions by prematurely stopping an exact search, such as branch and bound. The fourth class is on methods that solve a relaxation of the original model to build a feasible solution.

Archetti and Speranza (2014) surveyed the main matheuristic approaches applied specifically to routing problems. The authors follow a similar classification to Ball (2011) by discriminating in three types: decomposition, improvement, and branch and price or column generation-based approaches. The method proposed in this thesis fits the description of the *improvement* class, because as detailed in Chapter 4, the MP component is integrated into the search and used to improve the current solution at each iteration, which is precisely the definition of Archetti and Speranza (2014).

Among matheuristic approaches for VRPs there are applications to the Pollution-Routing Problem (KRAMER et al., 2015), Dial-a-Ride Problem (PARRAGH; SCHMID, 2013), Home Health Care Problem (ALLAOUA et al., 2013), and a class of VRPs (SUBRAMANIAN; UCHOA; OCHI, 2013). Other applications include the High School Timetabling Problem (DORNELES; ARAÚJO; BURIOL, 2014), the Nurse Rostering Problem (DELLA CROCE; SALASSA, 2014), the Bin Packing Problem with Conflicts (CAPUA et al., 2018), and Ship Routing (HOMSI et al., 2018), which is closely related to VRPs.

3.2 Methods for the Pickup and Delivery Problem with Time Windows

Due to the large solution space and hard time window, capacity, and precedence constraints of the PDPTW, the main methods for finding solutions to the problem are metaheuristics. Even though there are exact approaches able to solve tightly constrained instances with up to 500 requests, for more general cases they are still unable to find optimal or even feasible solutions. In this section, we will review the main methods for the PDPTW and some related topics.

One related topic is the *Dial-a-Ride Problem* (DARP), which is a similar problem to the PDPTW. In fact, the DARP is often referred to as a PDPTW of human perspective, because passengers rather than products are transported by vehicles from pickup to drop off locations. The main applications of the DARP are the transportation of disabled and elderly people. To account for customer satisfaction, the DARP has additional constraints to limit the riding period of each passenger. Nonetheless, the literature of the DARP and of

the PDPTW for heuristic search methods is somewhat divergent, particularly because the DARP does not usually aim at minimizing the number of vehicles, whereas the PDPTW does, and this can effectively change the design of heuristic algorithms.

The DARP has been more explore than the PDPTW in the literature, arguably due to its social applicability. For a survey on the DARP, the reader is referred to the work of Doerner and Salazar-González (2014). For further information on solution methods for Pickup and Delivery problems for the transportation of goods, including the PDPTW and other variants, the reader is referred to the survey of Battarra, Cordeau and Iori (2014).

Furthermore, problems in industrial and tramp shipping have similarities to the PDPTW. Typically these problems extend the PDPTW with additional characteristics of the shipping industry that may include heterogeneous fleet, multiple depots, flexible cargo size, and multiple time windows. The work of Christiansen et al. (2013) reviewed some shipping problems related to the PDPTW.

3.2.1 Heuristic Methods

One of the first metaheuristics to solve the *multi-vehicle PDPTW* is due to Nanry and Barnes (2000). The authors proposed a *Reactive Tabu Search* using three basic neighborhood movements in the local search: i) moving a request from one route to another; ii) swapping a request between two routes; iii) relocating a request within its route. The method allowed infeasible solutions to be used during the search procedure. The data set for experiments was based on instances of the VRPTW proposed by Solomon (1987).

Li and Lim (2003) developed a *Tabu-embedded Simulated Annealing with Restarts* approach to solve the PDPTW. The method used the same neighborhoods defined by Nanry and Barnes (2000) within the embedded tabu search. In order to evaluate their metaheuristic, the authors performed experiments using two sets of instances: the one of Nanry and Barnes (2000), and another that they proposed based on the VRPTW instances of Solomon (1987) and Gehring and Homberger (2001). The latter set of instances has become the standard benchmark set for the PDPTW, which has not been fully solved yet. Its instances and solutions are currently kept by the SINTEF (2008) website.

Bent and Hentenryck (2006) applied a *Two-stage Hybrid Algorithm* to solve the PDPTW. In the first stage, a *Simulated Annealing* is responsible for minimizing the number of routes by employing a modified objective function that favors an uneven distribution of requests among the routes – the strategy is to create longer and shorter routes, so

that the procedure could eventually remove the shorter ones. The second stage minimizes the travel distance using a *Large Neighborhood Search* (LNS), which is solved through a *branch-and-bound*. This proposed method was able to produce a series of new solutions for the Li and Lim (2003) benchmark set.

Another two-phase algorithm was proposed by Ropke and Pisinger (2006), based entirely on LNS. The two phases of the method are the same procedure, and the difference between them relies only on the weights used in the objective function. The overall idea is to remove a number of requests at once and reinsert them so that the solution space is explored using large movements. In the first stage, the goal is to minimize the number of routes by completely removing one route at a time and trying to reinsert the unrouted requests in the remaining routes. The second stage is responsible for distance minimization, by iteratively removing and reinserting a number of requests. To remove and reinsert requests Ropke and Pisinger (2006) propose a series of operations, which are chosen at each iteration of the algorithm based on how well they have performed during its execution. The strategy is called *Adaptive Large Neighborhood Search* (ALNS), and it was effective to find good solutions for the standard benchmark set (LI; LIM, 2003).

A variant of the PDPTW in which only vehicle minimization was taken into account was studied by Nagata and Kobayashi (2010a). To solve this problem, the authors proposed the *Guided Ejection Search* (GES) that is based on a simple idea of removing a random route at each iteration and trying to reinsert its requests into the other routes, possibly ejecting some more requests to perform the insertion. Their results were successful in reducing the number of vehicles of standard PDPTW instances, but the method did not account for travel distances. On the other hand, Nagata and Kobayashi (2010b) developed a *Memetic Algorithm Using Selective Route Exchange Crossover* that uses the GES to reduce the number of vehicles, and improves the travel distance with the memetic algorithm. The method was able to find good solutions for the standard benchmark set.

Curtois et al. (2017) presented a hybrid metaheuristic that can be considered the current *state-of-the-art* for the PDPTW. In their work, the authors created an adaptive mechanism to improve the search of the GES heuristic, a modification they named *Adaptive Guided Ejection Search* (AGES). The final algorithm is an iterative combination of an AGES phase, an LNS phase and a local search phase with four neighborhood strategies. Overall, the algorithm presents two stages that are iterated many times, instead of a single execution of each phase as in Bent and Hentenryck (2006), Ropke and Pisinger (2006), a strategy that intensifies the search and is able to take advantage of previous information

in both stages. The results demonstrated a major reduction on the number of vehicles and costs. Indeed the algorithm was able to improve a large number of solutions from Li and Lim (2003) benchmark instances.

3.2.2 Exact Methods

Dumas, Desrosiers and Soumis (1991) developed one of the first branch and price methods applied to the PDPTW. Due to the computational limitations, however, the algorithm was only able to solve instances tightly constrained by demand, with up to 55 requests. Savelsbergh and Sol (1998) also developed a branch and price method and were able to solve less constrained instances with up to 50 requests.

A branch and cut approach was proposed by Lu and Dessouky (2004) for a PDPTW with multiple depots and heterogeneous vehicles. The authors used a two-index compact formulation and valid inequalities to strengthen the formulation. Instances with up to 25 requests and 5 vehicles were solved to optimality. Ropke, Cordeau and Laporte (2007) developed another branch and cut for the classical PDPTW based on a two-index formulation, and introduced valid inequalities to improve the performance. They also extend the models to solve the DARP. Instances with up to 70 requests for the PDPTW and 96 requests for the DARP were solved to optimality.

Ropke and Cordeau (2009) proposed the use of a branch and cut and price method to solve the PDPTW. The method uses column generation to solve the relaxation of a set partitioning formulation in order to compute lower bounds, and considers two pricing subproblems. A new set of harder instances was proposed to evaluate exact methods for the problem. The authors compared two relaxations of the problem and demonstrated that the method was able to outperform the previous one by Ropke, Cordeau and Laporte (2007). Moreover, the branch and cut and price method solved to optimality eight tightly constrained instances with 100 requests and three with 500 requests from the standard benchmark set (LI; LIM, 2003).

The current *state-of-the-art* exact solution method for the PDPTW was proposed by Baldacci, Bartolini and Mingozzi (2011). The method is based on a strengthened set partitioning formulation, and it was tested using two objective functions. All instances from the Ropke and Cordeau (2009) set, except one, were solved to optimality. The method solved six tightly constrained instances with 100 requests from the Li and Lim (2003) set in addition to the ones solved by Ropke and Cordeau (2009).

A compact two-index MILP formulation was proposed by Furtado, Munari and Morabito (2017). The authors experimentally demonstrated that their new formulation is able to provide tighter lower bounds when compared to the compact formulations presented by Ropke and Cordeau (2009) and Lu and Dessouky (2004). Nonetheless, the model was only tested with instances of up to 75 requests, and it is still an open question whether valid inequalities exist to strengthen the formulation.

3.2.3 Matheuristic Methods

Some works in the literature applied matheuristic methods to the PDPTW and its variants. We review four works that are related to this thesis in terms of methods employed and problems solved.

Koning (2011) applied a column generation approach to the PDPTW with disturbances. The generation of new columns in the pricing problem was done through a Simulated Annealing metaheuristic, which employed evolutionary strategies to mutate sequences of requests. The algorithm was tested with the Li and Lim (2003) instances, and provided a few new best solutions. However, its performance was mostly limited to instances with 200 requests due to research decisions.

For the DARP, Parragh and Schmid (2013) applied a matheuristic that generated routes employing an LNS algorithm and used a Set Covering Formulation to recombine a set of previously created routes. However, due to the use of Set Covering, the method required a procedure to make solutions feasible, since the model allowed some requests to be visited more than once in the same solution, rendering it infeasible.

Homsi et al. (2018) proposed a matheuristic to a variant of the PDPTW in the context of Industrial and Tramp Ship Routing. The method applied a Unified Hybrid Genetic Search with an intensification phase that uses a Set Partitioning formulation to recombine routes from previous solutions. The authors showed that the addition of the MP component significantly improved the results and that the method was able to find solutions close to the optimal.

To the best of our knowledge, the only other work applied to the particular PDPTW that is studied in this work is our initial study (SARTORI; BURIOL, 2018). In this preliminary work, we have combined the Set Partitioning component to the AGES and LNS. A tuning procedure was performed to verify the best set of parameters, which demonstrated that the MP phase produced statistically better results. Hence, it was worth the

extra overhead. Additionally, it was verified that the MP component was able to replace a local search phase successfully. This thesis extends these preliminary results.

Even though the algorithm proposed in this thesis may at first appear similar to the works of Parragh and Schmid (2013) and Homsi et al. (2018), there are significant differences between these methods that motivate our research. For example, the heuristic components that the algorithms use are different, especially comparing our proposal to the one of Homsi et al. (2018). The hybrid method of Parragh and Schmid (2013) also uses LNS, but our method uses the AGES and Set Partitioning, instead of Set Covering. The objective function of the PDPTW studied in this thesis differs from the other two methods, because it tries to minimize the number of vehicles, whereas the other two do not seek this goal.

4 A MATHEURISTIC APPROACH

In this chapter, we describe the proposed matheuristic approach for the PDPTW. The method is structured as an Iterated Local Search and uses components from the literature including the Adaptive Guided Ejection Search (CURTOIS et al., 2017) and the Large Neighborhood Search (ROPKE; PISINGER, 2006). Indeed, the matheuristic extends the current *state-of-the-art* hybrid heuristic by Curtois et al. (2017). The choice of the ILS framework to guide the algorithm was straightforward due to the streamlined components embedded within their hybrid method.

Algorithm 3 describes the overall structure of the matheuristic. There are several parameters besides the input instance $\mathcal{I}$ – each defined within its stage in the algorithm. In line 1, solution s is assigned an initial solution by the procedure in Section 4.1. There is a *pool of routes*, denoted as $\mathcal{P}$, which is used by the mathematical programming component in line 7, as detailed in Section 4.4. The pool is initialized in line 2, with routes from solution s. Variables iter and count keep the total number of iterations and the number of iterations without improvement, respectively. The best solution found is denoted by s^*.

Algorithm 3 Matheuristic

Input: PDPTW Instance $\mathcal{I}$; perturbation size Z_M and bias μ; AGES parameters M_A, Z_A; LNS parameters $M_L, L, b_{min}, b_{max}, \omega, k_{min}, k_{max}$

1: $s \leftarrow$ `initial_solution()`
2: $\mathcal{P} \leftarrow$ `initialize_pool(`s`)`
3: iter, count$\leftarrow 0$
4: **repeat**
5: $\quad s \leftarrow$ `AGES(`s, M_A, Z_A, μ`)` ▷ reduce number of routes
6: $\quad s \leftarrow$ `LNS(`$s, M_L, L, b_{min}, b_{max}, \omega, k_{min}, k_{max}$`)` ▷ reduce cost
7: $\quad s^M \leftarrow$ `SP(`$s, \mathcal{P}$`)` ▷ recombine routes
8: $\quad s \leftarrow$ `accept_solution(`s, s^M, iter, count`)`
9: $\quad s \leftarrow$ `perturb(`s, Z_M, μ`)`
10: **until** stopping condition

Lines 4–11 are iterated until a given stopping condition is met. Every iteration is composed of a local search, a solution acceptance, and a perturbation according to the ILS framework. The local search is a combination of three stages. First, an AGES phase to reduce the number of vehicles, described in Section 4.2. Then, an LNS phase to minimize the total cost, as outlined in Section 4.3. The third stage, in line 7, is the MP component that solves a Set Partitioning (SP) formulation of the PDPTW over the pool $\mathcal{P}$ to recombine routes of previous solutions in search for an improving solution s^M.

The solution s to continue the search is chosen according to an acceptance criterion based on the number of iterations $iter$ and $count$, as per Section 4.5. In line 10, the accepted solution s may be perturbed to avoid local minimum traps according to Section 4.6.

In each iteration, the routes of the local minimum solution s returned by the LNS phase are added to the pool of routes $\mathcal{P}$. Therefore, the approach is arguably related to column generation techniques. Furthermore, after every addition of new columns, the model is solved. In case no column is added to $\mathcal{P}$, the model is not solved.

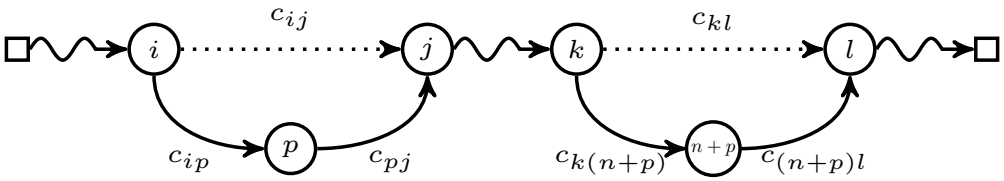
4.1 Greedy Solution Constructor

A simple greedy insertion heuristic creates an initial solution for the matheuristic. The heuristic is based on the procedure of Solomon (1987) and similar to the one by Li and Lim (2003). The insertion heuristic starts from an empty solution and builds feasible PDPTW routes one at a time. Hence it is a *sequential* route construction algorithm.

Initially, the heuristic selects a request to initialize an empty route r. The selection is based on two criteria. All unrouted requests $u = (p, n+p),\ p \in P$ that have earliest start of time window e_p at the pickup location p are chosen. Then, among the previously selected requests, the one that minimizes the cost c_{0p}, from the depot to the pickup location, is taken. In case of ties, the request with largest pickup index p is selected.

Once a route r has been initialized, other requests are iteratevely inserted into r, one at a time, at the position that minimizes the insertion cost (see Equation 4.1 below). Recall that a request $u = (p, n+p)$ is a pair, thus two positions are required to insert a single request, one for p and another for $n+p$. Figure 4.1 depicts the insertion of a request $u = (p, n+p)$ in a route r. Denote by (i, j) the position to insert p between adjacent locations i and j in route r. Similarly, denote by (k, l) the position to insert $n+p$.

Figure 4.1: Example of the insertion of request $(p, n+p)$ in route r. Pickup location p is inserted between locations i and j, while delivery location $n+p$ is inserted between k and l. Dotted arcs represent removed paths. Coiled arcs represent possibly long paths with multiple visits. The square is the depot, and circles are customer locations.



The total insertion cost C_u of request $u = (p, n+p)$ using positions (i, j) and (k, l) in route r is computed according to Equation 4.1 below. Costs of the form c_{ij} were defined in Chapter 2 as the cost of traveling from location i to j.

$$C_u = c_{ip} + c_{pj} + c_{k(n+p)} + c_{(n+p)l} - c_{ij} - c_{kl} \tag{4.1}$$

Therefore, at each iteration of the request insertion phase, the request and insertion positions that minimize C_u are chosen. Note that k is allowed to be p, because a delivery may happen immediately after its pickup. In this case, $k = p, l = j$ and the total insertion cost is slightly modified to simply $C_u = c_{ip} + c_{p(n+p)} + c_{(n+p)j} - c_{ij}$.

The request insertion phase in route r continues until there are no more feasible insertions. In such a case, another route is initialized and the insertions repeated. New routes are created as long as there are unrouted requests and no empty route is generated. Indeed, no limit was imposed on the maximum number of routes generated by the greedy heuristic. Hence, if the procedure is not able to produce a feasible solution, the instance contains at least one request that violates time window or capacity constraints.

4.2 Adaptive Guided Ejection Search

The AGES (CURTOIS et al., 2017) is a heuristic component solely responsible for vehicle minimization. To search for solutions with fewer routes, it applies an aggressive approach that removes an entire route r from the current solution, and then reinserts the requests from r into the remaining routes. Algorithm 4 describes the AGES.

The procedure is iterated (lines 1 – 17) for a maximum number of perturbations M_A. First, a random route r is removed from solution s in lines 2–3. The removal of route r creates a partial solution s'. In line 4, all requests that belong to r are inserted into a stack E, for last-in-first-out (LIFO) ordering (NAGATA; KOBAYASHI, 2010a). Penalty counters $\rho[u]$ for each request u are initialized in line 5.

Next, a request u_{in} is removed from the stack to be reinserted into s'. A random feasible insertion position is selected for u_{in} by selecting a random route and selecting a position using *reservoir sampling*. If there is no available position, the penalty counter $\rho[u_{in}]$ is increased and a number k of requests are ejected from the partial solution. First, the procedure tries to insert u_{in} considering the ejection of a single request ($k = 1$). In case there is still no insertion position available, the method considers the

ejection of $k = 2$ requests. The heuristic strategy used for ejection is based on the penalty counter ρ. For $k = 1$, the request u that minimizes $\rho[u]$ is ejected, while for $k = 2$, requests u and v that minimize $\rho[u] + \rho[v]$ are ejected and inserted in E – u and v belong to the same route. The underlying idea is to remove requests that are easier to reinsert later, as opposed to requests with large penalties that should remain where they currently are. After the corresponding ejections, the current solution s' is perturbed by the procedure described in Section 4.6.

Algorithm 4 Adaptive Guided Ejection Search

Input: Feasible solution s; Maximum perturbations M_A; Perturbation size Z_A; Perturbation bias b

1: **while** maximum number of perturbations M_A is not reached **do**
2: $r \leftarrow$ `random_route(`s`)`
3: $s' \leftarrow$ `remove_route(`s, r`)`
4: $E \leftarrow$ `initialize_stack(`r`)`
5: $\rho[u] \leftarrow 1$, for every request u
6: **while** maximum number of perturbations M_A is not reached **and** $E \neq \emptyset$ **do**
7: $u_{in} \leftarrow$ `remove_request(`E`)` ▷ with LIFO ordering
8: **if** there is a feasible insertion of u_{in} in s' **then**
9: $s' \leftarrow$ `insert_request(`u_{in}, s'`)` ▷ in a random position
10: **else**
11: $\rho[u_{in}] \leftarrow \rho[u_{in}] + 1$
12: $s' \leftarrow$ `eject_and_insert(`u_{in}, s', E`)` ▷ eject $k = \{1, 2\}$ requests from s' that minimize the penalty ρ
13: $s' \leftarrow$ `perturb(`s', Z_A, b`)`
14: **end if**
15: **end while**
16: **if** $E = \emptyset$ **then** $s \leftarrow s'$ **end if**
17: **end while**
18: **return** s

The stopping criterion is based on the total number of perturbations executed. Every time a perturbation is performed in line 12 of the procedure, a counter is increased. On the other hand, whenever the number of unrouted requests decreases below a previous minimum value, the counter is reset. In line 16, and every time the AGES procedure is used during the search in Algorithm 3, the counter is also reset. This is the adaptive mechanism introduced by Curtois et al. (2017) that allows the AGES to run for as long as it seems to make progress, but to terminate quickly otherwise.

In line 16, if stack E is empty, it means a route was successfully removed from solution s and that s' is now a full feasible solution. Therefore, the current solution s is updated and the whole process is repeated.

4.3 Large Neighborhood Search

The primary objective of the LNS is to reduce the solution cost, although it can reduce the number of routes in some cases as well. The applied LNS is based on the works of Ropke and Pisinger (2006) and Curtois et al. (2017). The overall structure is described in Algorithm 5.

In every iteration of the LNS, a removal heuristic h (Section 4.3.1) is selected at random according to a set of weights ω (line 3). A number b of requests are removed from the current solution s by the selected strategy, which generates a partial solution s' in line 4 of the algorithm. The number b is selected at random from $\mathcal{U}[b_{min}, b_{max}]$ in each iteration of the LNS. Then, in line 5, requests are inserted back into s' using the k-regret heuristic (Section 4.3.2), where k is selected from $\mathcal{U}[k_{min}, k_{max}]$. If all requests have been reinserted, solution s' is accepted according to a Late Acceptance Hill Climbing (LAHC) (BURKE; BYKOV, 2008) strategy with list size L as proposed by Curtois et al. (2017). In case some requests were not reinserted, solution s' is restored to its initial state. The method is repeated for a maximum number of iterations without improvement M_L.

Algorithm 5 Large Neighborhood Search

Input: Feasible solution s; Maximum iterations M_L; LAHC list size L; Weight set ω; Values $b_{min}, b_{max}, k_{min}, k_{max}$

1: **repeat**
2: $s' \leftarrow$ `remove_requests`(s, b, h)
3: $s' \leftarrow$ `reinsert_requests`(s', b, k)
4: $s \leftarrow$ `accept_solution`(s, s', L)
5: **until** maximum number of iterations M_L without improvement
6: **return** s^* ▷ The best solution found

4.3.1 Removal Heuristics

Three removal heuristics are employed in the LNS as in the work of Ropke and Pisinger (2006). They are the *Shaw Removal Heuristic*, *Random Removal Heuristic*, and the *Worst Removal Heuristic*. A set $\omega = \{\omega_s, \omega_r, \omega_w\}$ holds the weights used in the selection of the Shaw, Random, and Worst heuristic, respectively. A heuristic h is selected with probability $\omega_h/(\omega_s + \omega_r + \omega_w)$.

4.3.1.1 Shaw Removal

The underlying idea of the Shaw Removal heuristic is the fact that one should remove requests that are related in order to reinsert them more easily afterward. A relatedness measure is employed in the selection of requests to be removed. The measure takes into consideration spatial, temporal and demand information of the requests. Given two requests $u = (p, n + p), v = (q, n + q),\ p, q \in P$, the relatedness $R(u, v)$ between them is computed by Equation 4.2.

$$R(u,v) = \alpha(t_{pq} + t_{n+p,n+q}) + \beta(|B_p - B_q| + |B_{n+p} - B_{n+q}|) + \gamma(|q_p - q_q|) \quad (4.2)$$

Lower values of $R(u, v)$ indicate a greater relationship between the requests. The term weighted by α is the spatial term and considers how spatially close are the requests u and v. Temporal relation is weighted by β and considers the time each request is serviced in the current solution – variables B_i are the exact time service starts at location i as defined in Chapter 2. The term weighted by γ is the demand relation. These weights were fixed to $\alpha = 9$, $\beta = 3$, and $\gamma = 2$ as per Ropke and Pisinger (2006).

Algorithm 6 describes the Shaw Removal. In line 1, a seed request u_1 is selected at random from the solution s. Then, removal operations are iterated in lines 3 – 9, until b requests have been removed. In line 4, a request u_2 is selected at random from the set of removed requests U. A list of routed requests $\mathcal{L}$ is defined in line 5. A position o in this list is selected in line 6 with a randomization criterion y^λ (explained in the following paragraph). Then, in line 7, the request u_3 with o-th lowest relation value $R(u_2, u_3)$ with regards to u_2 is taken using a linear time selection algorithm (CORMEN et al., 2009).

Algorithm 6 Shaw Removal Heuristic

Input: Solution s, number of requests to remove b

1: $u_1 \leftarrow$ `random_request(`s`)`
2: $U \leftarrow u_1 \cup U$
3: **while** $|U| < b$ **do**
4: $\quad u_2 \leftarrow$ `random_request(`U`)`
5: $\quad \mathcal{L} \leftarrow$ list of requests $u \in s \mid u \notin U$
6: $\quad o \leftarrow y^\lambda |\mathcal{L}|,$ where y is a random number from $\mathcal{U}[0, 1)$
7: $\quad$ select request $u_3 \in \mathcal{L}$ such that $R(u_2, u_3)$ is the o-th lowest relation value
8: $\quad U \leftarrow u_3 \cup U$
9: **end while**
10: $s \leftarrow$ `remove_requests(`s, U`)`
11: **return** s

The randomization criterion used in line 6 of the algorithm generates a number at random y and uses a parameter denoted by λ. Whenever $\lambda = 1$ the selection is completely random, although the value has been fixed as $\lambda = 6$ to favor elements with a stronger relationship. In line 8, the selected request is added to set U. In line 10, all requests in U are removed from s.

4.3.1.2 Random Removal

The random removal selects a number b of requests to remove independently at random. Ropke and Pisinger (2006) highlight that the Random Removal is basically a Shaw Removal heuristic in which the parameter $\lambda = 1$. In the concrete implementation, a separate heuristic is available for performance purposes.

Unlike the Shaw Removal, the Random Removal does not consider any information or relation about the removed requests. Nonetheless, its effectiveness cannot be underestimated. In fact, it is a component that can diversify the search.

4.3.1.3 Worst Removal

A different strategy is to consider requests that seem to be placed in the wrong location. In the notation of Ropke and Pisinger (2006), denote by $f_{-u}(s)$ the cost of solution s when request u has been removed. Then, denote the cost of request u in solution s by $cost(u, s) = f(s) - f_{-u}(s)$. The Worst Removal heuristic removes the b requests that contribute the most to the cost of solution s. In other words, it selects those requests that maximize $cost(u, s)$. Note that $f_{-u}(s)$ can be computed by Equation 4.1.

The implementation of the heuristic is similar to the Shaw Removal. However, the difference is that list $\mathcal{L}$ is sorted in decreasing order of $cost(u, s),\ \forall u \in \mathcal{L}$. In this manner, requests with larger values of $cost(u, s)$ are the ones with a higher probability of selection. There is a randomized component y^{λ} as well to avoid deterministic selection.

4.3.2 Insertion by Regret Heuristic

In the repair phase of the LNS, the b requests previously removed are reinserted into the partial solution s'. To reinsert requests, the k-regret heuristic proposed by Ropke and Pisinger (2006) is employed. It performs a lookahead when selecting requests to reinsert. One of the objectives is to choose requests that can only be inserted in a few locations in the current solution s', and thus need to be reinserted first.

The k-regret heuristic works as follows. Denote by $f_{+u,r}(s)$ the insertion cost of u in the best position in route $r \in s$. Consider that routes are ordered in such a way that $r = i$ is the route where u is inserted with i-th lowest cost. In other words, insertion costs are $f_{+u,1}(s) \leq f_{+u,2}(s) \leq \cdots \leq f_{+u,|s|}(s)$. Costs $f_{+u,r}(s)$ are computed by Equation 4.1. Whenever a request u cannot be inserted in a route j, we set insertion costs as $f_{+u,j}(s) = \cdots = f_{+u,|s|}(s) = \infty$. Then, at each iteration of the regret heuristic, a request u from the set of unrouted requests U is selected for reinsertion according to Equation 4.3.

$$\max_{u \in U} \{\sum_{j=1}^{k} (f_{+u,j}(s) - f_{+u,1}(s))\} \tag{4.3}$$

The selected request u is always inserted in its best position with cost $f_{+u,1}(s)$. The maximized term is the so-called *regret value*. Indeed, this value is a quantification of the potential risk of not inserting a request at its best insertion position in the current iteration. Note that requests that can only be inserted in one route will have a much larger regret value. Hence, they are selected before other requests that have more options.

The parameter k in a k-regret heuristic is the size of the lookahead. For instance, when $k = 1$ the heuristic is a simple greedy insertion (as in Section 4.1). For $k \geq 2$, the heuristic considers further moves to better choose a request to insert. When $k = |s|$, all routes in s are considered. In our implementation, at every iteration of the LNS phase, a value k is selected from $\mathcal{U}[k_{min}, k_{max}]$.

In the concrete implementation, we compute all the insertions of each unrouted request $u \in U$ and keep the results in memory. Every time a request is chosen to be inserted in a route r (according to Equation 4.3), we update the insertion cost of all the remaining unrouted requests in U for the particular route r. For all the other routes, the insertions remain the same of previous iterations and do not need to be computed again.

4.4 Set Partitioning Formulation

Balinski and Quandt (1964) presented an Integer Linear Programming model to formulate VRPs using a Set Partitioning formulation. The same model can be used in our context. Let $\mathcal{R}$ define the set of all feasible routes of the PDPTW. There are binary values λ_{ir} associated with each route $r \in \mathcal{R}$, such that $\lambda_{ir} = 1$ if route r services customer $i \in P \cup D$, and $\lambda_{ir} = 0$ otherwise. The term $C(r)$ is the total cost of route r, as defined in Chapter 2.

Furthermore, there are binary decision variables y_r that assume value 1 when route $r \in \mathcal{R}$ is used in the optimal solution, and 0 otherwise. Then, the SP model for the PDPTW can be written as follows.

$$\textbf{minimize} \quad \sum_{r \in \mathcal{R}} y_r C(r) \tag{4.4}$$

$$\textbf{subject to}$$

$$\sum_{r \in \mathcal{R}} \lambda_{ir} y_r = 1 \qquad \forall\, i \in P \tag{4.5}$$

$$y_r \in \{0, 1\} \qquad \forall\, r \in \mathcal{R} \tag{4.6}$$

The size of set $\mathcal{R}$ is, however, exponential in the instance size. To overcome the cardinality problem, it is common to use external methods to generate a subset of the routes, so that solving the model is tractable. In our approach, the pool of routes $\mathcal{P}$ acts as the set $\mathcal{R}$, but with a smaller size. At every iteration of the matheuristic, routes of the local minimum solution s returned by the LNS phase are added to the pool.

The SP model is solved in line 7 and returns a solution s^M, for which necessarily $f(s^M) \leq f(s^*)$. Solution s^* is the best found during the search. This is certain because no routes are removed from the set $\mathcal{P}$ during the search. Thus, the routes of s^* are still available. Besides, in the implementation of our matheuristic, we used *warm start* to improve the efficiency of the MP component by setting s^* as the initial solution.

In computational terms, the set $\mathcal{P}$ is a hash map in which the key is the set of requests serviced by the given route. If two routes r_1 and r_2 service the same requests, regardless of their order, and $C(r_1) < C(r_2)$, then only route r_1 is maintained due to its reduced cost. This keeps memory usage low without any side effects.

4.5 Solution Acceptance

In the matheuristic, whenever the SP phase generates a new solution s^M that is better than the current best, i.e., $f(s^M) < f(s^*)$, s^M is readily accepted. On the other hand, the local minimum s' of the LNS phase is accepted with probability $1 - (count/iter)$. Otherwise, solution s^* is taken to continue the search in the next iteration of the algorithm. Therefore, our matheuristic focuses on intensification in the start, but gradually shifts to diversification to avoid getting trapped in local minima. In case solution s^M is not accepted, the chosen solution s is perturbed by the mechanism in Section 4.6.

4.6 Perturbation

A perturbation mechanism is used in the overall matheuristic and within the AGES heuristic. The perturbation is a modification of the *random request shift* movement applied by previous works (NAGATA; KOBAYASHI, 2010a; CURTOIS et al., 2017). In the original movement, a random request is selected to be shifted from its route to another route taken at random in the current solution.

Instead of selecting the destination completely at random, the perturbation applied in this work uses a so-called *bias* μ. The bias works in the following way. A request u is randomly selected from the current solution s and removed from it entirely. Then, a route $r \in s$ is randomly chosen to receive u. From all possible insertions of u in r, a set of $\mu\%$ of them is taken at random and the best position is selected for the insertion. In case $\mu < 100\%$ there is a bias towards better insertions, but not necessarily the best overall, which maintains the desired diversification in a perturbation mechanism. Whenever there is no feasible insertion for u, the shift is not performed. The concrete implementation of the perturbation is similar to the greedy reinsertion with blinks of Christiaens and Vanden Berghe (2016), although we have used the strategy in a perturbation mechanism rather than in an improvement procedure.

For the matheuristic, a maximum number of Z_M shifts is performed, while in the AGES the maximum is denoted by Z_A. There is a difference because in the AGES the perturbation should not be too large. Although, in both cases, the same bias μ is used.

4.7 Efficient Computations

The AGES, LNS, and the perturbation mechanism, all require the evaluation of movements within a certain neighborhood in the search space. In particular, they perform removals and insertions of requests and always keep solution feasibility. We discuss in this section the techniques used to perform these computations efficiently.

To evaluate how much a movement impacts the solution cost, we compute the differences using Equation 4.1 previously defined. That is, whenever a request is removed or inserted in a route, we compute solely the modification incurred by this movement, i.e., the removal and insertion of a few arcs. Note this evaluation can be performed in $\mathcal{O}(1)$, or constant time.

On the other hand, problems that consider time window constraints pose a difficulty for the solution methods because insertions in a route modify the service time of all subsequent locations, which may cause the solution to become infeasible. The naive approach is to check all the locations that come after the inserted one, but this has complexity $\mathcal{O}(n)$. When tackling large instances and using heuristic components that perform many of such tests, this simple approach will not provide good performance.

Savelsbergh (1992) proposed a technique to perform the feasibility check with regards to time windows in constant time complexity, that is, $\mathcal{O}(1)$. The method is called *Forward Time Slacks*, because it considers the *slack* in time of the requests to follow. Basically, it defines a value $F_i,\ \forall i \in V$, which is the maximum amount of time service at location i can be shifted in time to keep solution feasibility.

Given a route $r = (v_0, v_1, \ldots, v_h, v_{h+1})$ the forward time slack of each node in the sequence is computed according to Equations 4.7-4.8 below. Variables B_i where defined in Chapter 2 as the time service starts at location i. Value w_i is the waiting time at location i, that is, the difference between the time b_i a vehicle arrives at i, and the actual start of the service B_i, because a vehicle may arrive before e_i and wait. Hence, it is computed as $w_i = B_i - b_i$.

$$F_{v_{h+1}} = l_{v_{h+1}} - B_{v_{h+1}} \tag{4.7}$$

$$F_{v_k} = \min\{l_{v_k} - B_{v_k}, F_{v_{k+1}}\} + w_{v_k}, \quad \forall k = 1, \ldots, h \tag{4.8}$$

Therefore, whenever inserting a location in a route, we have to check if the shift in time created by the insertion is not greater than the slack of the location immediately after. In the case of the PDPTW, we perform the check for both the pickup location and the delivery location. Besides, we need to account for the fact that the shift created for the pickup location affects the delivery location as well because it comes necessarily before the delivery. These tests can be performed in $\mathcal{O}(1)$ time given a location for the request. However, we need to recompute the forward time slacks whenever a route is modified in $\mathcal{O}(n)$ time, which is worth it because the search performs a considerable number of time window feasibility tests.

5 INSTANCES OF THE PROBLEM

The purpose of this chapter is to describe the instances of the PDPTW used in the experiments of this thesis (Chapter 6). In Section 5.1, we detail the standard benchmark set of instances for the PDPTW that is currently used to compare methods in the literature. To further explore the problem, we present a method to generate instances based on realistic information in Section 5.3. A new benchmark set for the PDPTW is proposed based on this method.

5.1 Standard Instances

Li and Lim (2003) proposed the standard set of benchmark instances for the PDPTW. There are 354 instances with sizes of 100, 200, 400, 600, 800, and 1000 locations. Most instances have not been solved to optimality yet. The SINTEF (2008) website keeps the whole set and its corresponding best-known solutions.

Every instance is classified in one of three classes according to the distribution of its locations. The clustered (C) instances have places grouped in specific regions to simulate urban agglomerations. The random (R) instances have locations simply distributed at random. In the random-clustered (RC) instances, locations are partially random and partially clustered. Each distribution class is further separated according to the time windows and planning horizon. Instances of type 1 (C1, R1, RC1) have short planning horizon and time windows, whereas those of type 2 (C2, R2, RC2) have long planning horizon and time windows. For all instances, travel times and arc costs are computed by the *Euclidean distance* with double precision.

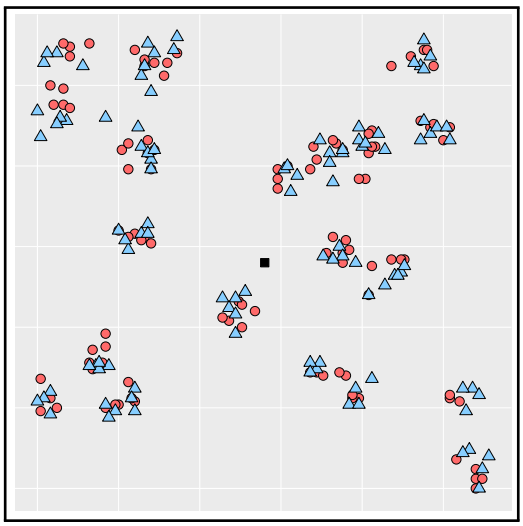
Table 5.1: Summary of the Li and Lim (2003) instances.

	Number of Instances by Type						
Inst. Size	$C1$	$C2$	$R1$	$R2$	$RC1$	$RC2$	Total
100	9	8	12	11	8	8	56
200	10	10	10	10	10	10	60
400	10	10	10	10	10	10	60
600	10	10	10	10	10	10	60
800	10	10	10	10	10	10	60
1000	10	10	10	10	10	8	58
Total	59	58	62	61	58	56	354

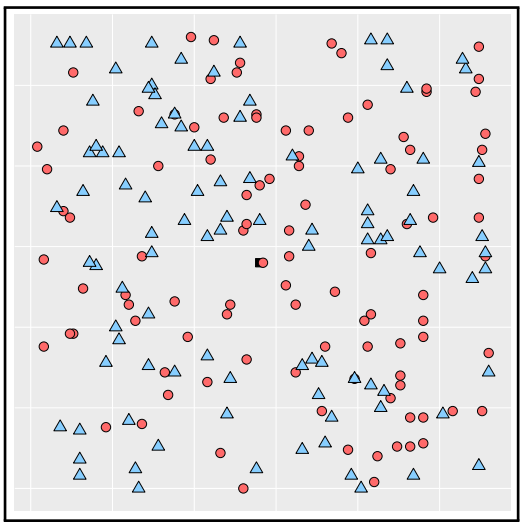
Source: The Author.

Table 5.1 presents a summary of the number of instances per particular group as described. In Figure 5.1, a sample of three instances is depicted, one for each distribution (C, R, RC) in the two-dimensional Cartesian plane.

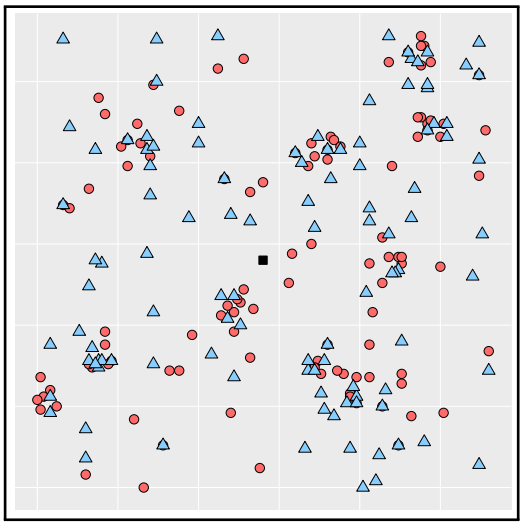
Figure 5.1: Example of three instances proposed by Li and Lim (2003). Red circles are pickup locations. Blue triangles are delivery locations. The black square is the depot.



(a) Instance with 200 (C) locations.



(b) Instance with 200 (R) locations.



(c) Instance with 200 (RC) locations.

Source: The Author.

To create the instances, Li and Lim (2003) used VRPTW instances proposed by Solomon (1987) and Gehring and Homberger (2001). In fact, the only attribute that differs VRPTW and PDPTW instances are the pairing constraints. To pair locations, the authors first solved each VRPTW instance using a heuristic algorithm. Then, locations that appeared in the same route of the VRPTW solution were randomly paired. These pairs correspond to the pickup-delivery pairs in the PDPTW instance.

On the one hand, the procedure generates instances with a particular structure, because the pickup and delivery are well arranged on the same route. Ropke (2005) mentions that the process may not generate realistic pickup-delivery relations as well. On the other hand, the works in the literature seem to agree that the Li and Lim (2003) instances are hard to solve, which is probably the reason the set is widely used. Only a few of these instances have been solved to optimality.

5.2 Instances for Exact Solution Methods

Ropke and Cordeau (2009) proposed a different set of instances especially to test exact methods for the PDPTW. The locations in these instances were selected uniformly at random over the $[0, 50]$x$[0, 50]$ square in the 2D plane. Travel times and arc costs are computed as the Euclidean distance. By definition, the cost of using a vehicle is 10^4.

There are four groups of instances. Each group differs from the others due to the capacity Cap of the vehicles or due to the width W of the time windows. All instances have the same scheduling horizon $H = 600$ time units. Requests were created by randomly pairing locations to create pickup-delivery pairs. Each instance group contains ten instances with 30 up to 75 requests. Most of these instances have already been solved to optimality. Table 5.2 below summarizes this particular set.

Table 5.2: Summary of the Ropke and Cordeau (2009) instances.

Group	Cap	W
AA	15	60
BB	20	60
CC	15	120
DD	20	120

Source: The Author.

5.3 A Proposal to Generate New Instances

A difficulty faced by many researchers in the VRP community is the lack of free available realistic benchmark instances. In order to compare methods, researchers have to rely on artificially generated instances. Typically locations are generated at random in a two-dimensional Cartesian plane in a particular distribution. Besides, distances are computed as the Euclidean or Manhattan distance, which provide strictly symmetrical cost matrices that fail to represent realistic scenarios.

The lack of instances based on real information motivated us to develop a method to create new instances for the PDPTW. These instances use real-world coordinates and travel times obtained from open data and software. Hence, we make the new benchmarks free available[1]. An open source tool was built to generate the instances[2].

The instances were generated within cities to avoid long travel times. In this way, all routes can be performed under a single working day of at most eight hours. The cities we have chosen are Barcelona (Spain), Berlin (Germany), New York City (United States of America), and Porto Alegre (Brazil). The reasoning behind the choice of Barcelona is due to its several one-way streets that provide large asymmetrical cost matrices (on average, 70% of the arcs). Opposed to that, Berlin is more symmetrical (on average, more than 50% of the arcs are symmetrical). Porto Alegre is our town, and it is currently in the process of innovation inspired by the city of Barcelona, a fact that motivated us to choose this location because of the possible future applications. At last, New York City has a particular dataset aggregated by Donovan and Work (2016), detailed in the following section, which provides complete information for PDPTW contexts.

Following, in Section 5.3.1 the strategies to select locations for customer requests is described. The particular method for Barcelona, Berlin, and Porto Alegre instances is presented in Section 5.3.2. There is an alternative procedure for the New York City instances in Section 5.3.3.

5.3.1 Obtaining Addresses and Travel Times

The locations in the new method are selected from a pre-defined pool, rather than randomly generated. The pool is a list $\mathcal{L}$ of geographic coordinates in an area of interest (e.g., country, state, or city). In this work, we use two datasets publicly available to create the pool of locations. One is from the OpenAddresses (2017), which keeps addresses for a number of cities and countries around the world. The second is from the dataset by Donovan and Work (2016) that includes pickup and dropoff coordinates for taxi services in New York City – in addition to other information (see Section 5.3.3).

Note that selecting locations from list $\mathcal{L}$ provides an approximation of urban distributions. Indeed, densely populated regions have more addresses, and the probability of choosing an address from $\mathcal{L}$ that belongs to a populated area is larger. Although, this is not a perfect estimation, because it depends on the input data available.

[1] Instances available at <https://github.com/cssartori/pdptw-instances/>.
[2] Open Source Instance Generator available at <https://github.com/cssartori/ovig>.

Furthermore, the use of geographic locations allows the computation of realistic travel times. In all instances, we employ the Open Source Routing Machine (OSRM) tool by Luxen and Vetter (2011) to compute travel times. The OSRM implements *state-of-the-art* shortest path algorithms that provide realistic results. Moreover, OSRM supports data from the OpenStreetMap (2017) project (OSM). An advantage of the OSRM is the possibility to use the information on speed limits per road, traffic directions, and road limitations to compute travel times. Travel times respect the triangluar inequality (Chapter 2) because OSRM always computes the fastest (shortest time) path between locations.

In all instances, travel times, time windows, service durations, and the scheduling horizon, are measured in minutes. Similar to instances of the TSPLib (REINELT, 1991) and the CVRPLib (UCHOA et al., 2017), we decided to use only integer values for arc weights. Hence, when computing the travel times in minutes, the values are rounded up. We decided not to differentiate arc costs and travel times, thus they are the same.

5.3.2 Method for Barcelona, Berlin, and Porto Alegre Instances

First, we describe the process to generate instances using the OpenAddresses (2017) locations dataset. The list $\mathcal{L}$ contains 129,460 sites for the city of Barcelona, 371,265 for Berlin, and 33,688 for Porto Alegre. Even though our instances are generated with information from these three particular cities, the method can be generalized to any other city or area as long as there are addresses available to them.

5.3.2.1 Selecting Locations

The PDPTW instances have $2n + 1$ locations. There are $2n$ customer locations and one depot. The $2n$ locations are paired to form a total of n requests (pickup-delivery pairs). Next, we describe how the $2n$ locations and the depot are selected from list $\mathcal{L}$. All locations are unique, and cannot be selected twice.

There are three distributions for the locations, analogous to the Li and Lim (2003) instances. In the clustered (C) instances, locations are selected from $\mathcal{L}$ in a way that creates clusters, similar to Uchoa et al. (2017). A set A of seed locations is selected at random, where the size of A is taken from $\mathcal{U}[3, 8]$. Then, the remaining $(2n - |A|)$ locations are selected according to the probability distribution in Equation 5.1, which favors the selection of requests that are closer to the seed locations to create the actual

clusters. The probability of selecting coordinate $x \in \mathcal{L}$ is denoted by $prob(x)$.

$$prob(x) = \sum_{a \in A} \texttt{exp}(-\texttt{hav}(x, a)d), \tag{5.1}$$

where $\texttt{hav}(x, a)$ is the *haversine distance* between coordinates x and a. This approximates the great-circle distance from x to a in the surface of the Earth (seen as a sphere). We opted for the haversine distance approximation in the selection stage, because otherwise we would have to compute the actual distance between all coordinates in $\mathcal{L}$, which is potentially large. Moreover, this approach provides a clusterization that is independent of streets and obstacles. The parameter d controls the density of clusters. Its value is chosen randomly from $\mathcal{U}[0.6, 1.6]$. Larger values of d generate denser clusters.

For random (R) instances, locations are randomly selected from list $\mathcal{L}$. In the random-clustered (RC) instances, a number $\lceil 2nh \rceil$ of locations are chosen to be clustered, and the remaining $\lceil 2n(1-h) \rceil$ are random. The value of h is taken from $\mathcal{U}[0.4, 0.6]$.

At last, the depot can be selected in two ways. In the first, a random location is simply selected from $\mathcal{L}$ to place the depot. The second type is the centered depot, in which the center of the rectangle containing all $2n$ locations is computed, and the location in $\mathcal{L}$ that minimizes the haversine distance to the center is selected to host the depot.

5.3.2.2 Pairing Locations

In the previous section, all $2n$ customer locations were selected from the list of locations $\mathcal{L}$. However, the PDPTW requires pickup-delivery pairs. Hence, in this section, we describe how the $2n$ locations are paired to create n pickup-delivery pairs.

In the literature, there are two basic approaches to pair locations. For example, Li and Lim (2003) paired locations that appeared in the solution of a VRPTW (Section 5.1). On the other hand, Savelsbergh and Sol (1998), Ropke and Cordeau (2009) paired locations in the input graph completely at random. In our generation, the pairs are created according to the distribution of locations in the instance.

For the R and RC instances, pairs are defined by selecting two locations at random, one for the pickup and another for the delivery. Therefore, there is no particular structure regarding pickup-delivery pairs in these instances. On the other hand, for the C instances, locations are paired within their cluster. Initially, clusters are computed by a *k-means algorithm*, where k is the number $|A|$ of seed locations. Then, a procedure is applied to guarantee that all clusters have an even number of locations (call them even-clusters).

Otherwise, it is not possible to pair locations completely. The procedure selects the largest odd-cluster z_1, and location $x_1 \in z_1$ that minimizes the haversine distance to the center of another odd-cluster z_2. The location x_1 is then moved to cluster z_2. The process is repeated until there are only even-clusters. Then, locations are randomly paired within their clusters. This strategy provides a degree of locality to the requests because both the pickup and the delivery belong to the same, or closeby neighborhoods.

5.3.2.3 Times and Scheduling Horizons

The scheduling horizon H is one of 240 minutes or 480 minutes, which is compatible with a working day on most countries. For service durations, we have decided to fix a single value for the whole instance. The duration is selected from 5, 10, or 15 minutes. At last, a time width W is chosen from 60 or 120 minutes to generate the time windows.

The time windows are computed as follows. Given a time window width W and a request $(p, n+p),\ p \in P$, we genereate the center w_p^c of the time window of p according to a uniform distribution considering the minimum and maximum time a vehicle can arrive at p in a feasible solution $\mathcal{U}[t_{0p}, H - t_{p(n+p)} - t_{(n+p)0} - s_p - s_{(n+p)}]$. Then, the window $[e_p, l_p]$ in p is computed as $e_p = w_p^c - W/2$ and $l_p = w_p^c + W/2$. For the corresponding delivery location $n + p$, the time window can be generated in two ways. The first is computed with $e_{(n+p)} = e_p + t_{p(n+p)} + s_p$ and $l_{(n+p)} = e_{(n+p)} + W$, thus there is an overlap between the time windows of p and $n+p$. The second way, however, is computed without any overlap of time windows as $e_p = l_p + t_{p(n+p)} + s_p$ and $l_{(n+p)} = e_{(n+p)} + W$. The latter is used to simulate cases in which the pickup can be performed early and the delivery only later. At most 10% of the requests receive non-overlapping time windows. Additonal 5% to 15% of the requests do not receive time windows, in other words, these requests have the time window $[0, H]$ in the pickup and the delivery location. Under no circumstance, a time window can have $e < 0$ or $l > H$.

5.3.2.4 Demands

The demands are generated based on the maximum capacity of the vehicles Cap. For each request, the demand is selected uniformly from $\mathcal{U}[10, 0.6 \cdot \text{Cap}]$. The maximum capacity Cap is one of 100 or 300 units of goods.

5.3.3 Method for Taxis of New York Instances

A different strategy is used to generate instances based on the data of Taxi trips in New York City by Donovan and Work (2016). The information available is the most complete we could obtain to create realistic PDPTW instances from open data. For example, each taxi trip contains the exact pickup and dropoff location (requests), the number of people riding the taxi (demand), and the precise time of pickup and dropoff (time windows). Therefore, our procedure makes use of all the information to create the instances, instead of generating demands, time windows, or pairs at random. Travel times are computed by the OSRM tool in minutes as well.

5.3.3.1 Selecting Requests and Depots

Instead of a list of locations $\mathcal{L}$, the New York City data provides a list of requests (paired locations). Thus, the selection of requests is performed by randomly choosing n requests from that list. Hence, the pickup-delivery pairs are defined by the input data rather than the method. In this way, the placement of locations in the space is defined by how the pickups and deliveries were distributed in the real application.

The process to select the depot is similar to the one in Section 5.3.2. For random depots, the location is simply chosen at random (from both pickup and delivery locations in the set of requests). In the case of central depots, the location that is closest to the center of the rectangle formed by the $2n$ locations is taken. Note that in both cases, when selecting the depot, the concept of request is ignored and only locations are considered.

5.3.3.2 Times and Scheduling Horizons

The scheduling horizon H is equally one of 240 or 480 minutes. Service times, on the other hand, are fixed in 2 minutes as an upper bound on the amount of time a passenger takes to get in and off a vehicle.

Time windows are computed according to the clock time of the pickup and dropoff reported in (DONOVAN; WORK, 2016). These times are used as the center w_p^c of the time window in the same way to the previous section (instead of generating the center randomly). Hence, for a request $(p, n+p),\ p \in P$ and center w_p^c for p, the time window is computed by $[w_p^c - W/2, w_p^c + W/2]$. For the delivery $n+p$, it is computed through $[w_{(n+p)}^c - W/2, w_{(n+p)}^c + W/2]$. In this manner, we try to keep as much of the

original information as possible when generating the instances. Widths W are fixed by instance and chosen from 30, 60, and 120 minutes. The tighter 30 minutes time windows were introduced because in this context it is reasonable to assume that requests should be attended as close as possible to their original center.

5.3.3.3 Demands

For demands, the input reports the number of passengers in each taxi ride, which we use as the demand of each request. The maximum capacity of all vehicles is fixed in $\text{Cap} = 6$ passengers. This limit was chosen because it is the largest number of passengers in a single ride in the (DONOVAN; WORK, 2016) data. Once again, this is a rough estimation in a way to try to exploit the most out of the information.

5.3.4 Discussion of the New Benchmarks

The new set of instances is intended to provide benchmarks for the PDPTW that mimic certain characteristics of real scenarios. These instances have been generated differently to those of the standard set (LI; LIM, 2003), thus the set can provide a distinct testbed for analysis of solution methods. In addition, we have generated larger instances for the PDPTW to attest the scalability of techniques applied to the problem.

The proposed instances are grouped in sizes of 100, 200, 400, 600, 800, 1000, 1500, 2000, 2500, 3000, 4000, and 5000 locations. For each one of these sizes, we created 25 instances, in a total of 300 benchmarks. In the generation process, we tried to maintain a balanced number of instances for each possible configuration (e.g., city, distribution, horizon, time window, capacity, depots). In this way, the final set has instances with diversified characteristics.

Table 5.3 presents a summary of the number of instances generated for each size and attribute. The information may be interpreted as follows. For instances with 100 locations, there are 6 located in Barcelona, 7 in Berlin, 5 in New York City, and 7 in Porto Alegre. Furthermore, among all the 25 instances, 7 have locations clustered, 11 random, and 7 random-clustered. Note that there are always 5 instances in New York City, and they are classified as randomly distributed. The three remaining cities are evenly distributed in the other 20 instances per size. Appendix D details all the 300 instances and their characteristics.

Table 5.3: Summary of characteristics of the new instances. Total number of instances created per configuration. Acronyms BAR, BER, NYC, and POA, stand for Barcelona, Berlin, New York City, and Porto Alegre, respectively.

	Cities				Distributions			Horizons		Time Windows			Capacities			Depots	
Size	BAR	BER	NYC	POA	C	R	RC	240	480	30	60	120	6	100	300	cent.	rand.
100	6	7	5	7	7	11	7	12	13	1	14	10	5	10	10	12	13
200	7	6	5	7	7	11	7	14	11	2	11	12	5	9	11	12	13
400	7	7	5	6	7	12	6	12	13	2	11	12	5	10	10	13	12
600	7	6	5	7	7	12	6	12	13	2	12	11	5	11	9	13	12
800	7	6	5	7	6	12	7	13	12	2	10	13	5	11	9	11	14
1000	6	7	5	7	6	12	7	12	13	3	11	11	5	10	10	13	12
1500	7	7	5	6	7	12	6	13	12	2	13	10	5	9	11	13	12
2000	7	7	5	6	7	12	6	13	12	1	12	12	5	10	10	13	12
2500	6	7	5	7	6	12	7	11	14	2	13	10	5	11	9	11	14
3000	7	7	5	6	6	12	7	12	13	1	13	11	5	11	9	14	11
4000	6	7	5	7	7	12	6	13	12	2	13	10	5	10	10	14	11
5000	6	7	5	7	7	11	7	13	12	1	12	12	5	10	10	12	13
Total	79	81	60	80	80	141	79	150	150	21	145	134	60	122	118	151	149

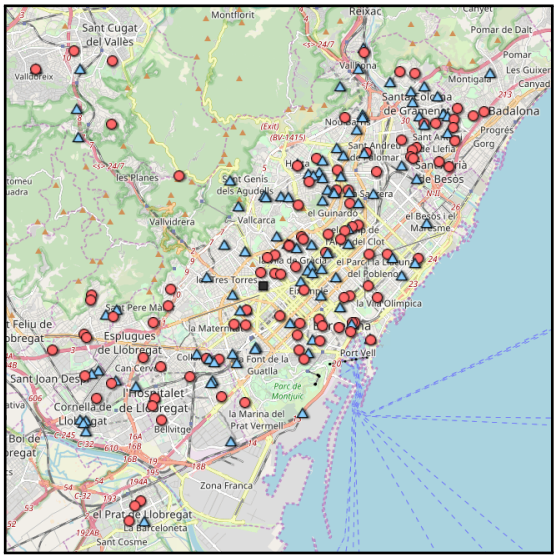
Source: The Author.

Figure 5.2 depicts four examples of the new benchmarks, one for each city. It is possible to verify that the distributions have selected locations in an expected manner. One could argue that locations do not necessarily appear in areas of the cities where it would be expected a larger number of deliveries (e.g., city centers, or industrial areas). The reason is simply that this is not information available in the input data. Although, areas with more addresses in list $\mathcal{L}$ do tend to have more locations selected. For example, the majority of requests in the New York City instances are located in Manhattan, where most addresses for the city are. However, a few outliers exist in other popular locations in New York City such as the John F. Kennedy Airport.

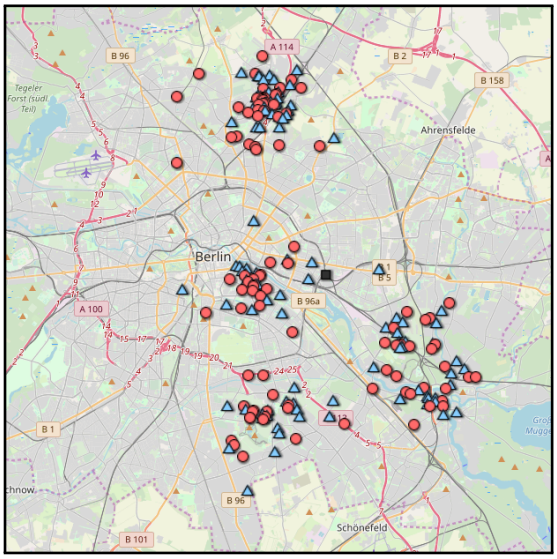
To the best of our knowledge, no previous work proposed similar instances for the PDPTW. There are earlier works in the literature that introduced instances based on real applications for the CVRP, such as the large instances proposed by Arnold, Gendreau and Sörensen (2017), in which the authors were able to extract a distribution of locations for certain regions in Belgium. However, the procedure depends on data provided by logistic operators, which is not readily available in many cases due to legal or contractual reasons. Indeed, our motivation was to propose a method and benchmarks that could be done solely with the best open data available.

In the next chapter, both the standard and the new benchmarks are used to verify the performance of our proposed matheuristic. Besides, initial upper bounds to the new instances are found throughout these experiments.

Figure 5.2: Example of four instances generated. Red circles are pickup locations. Blue triangles are delivery locations. The black square is the depot.

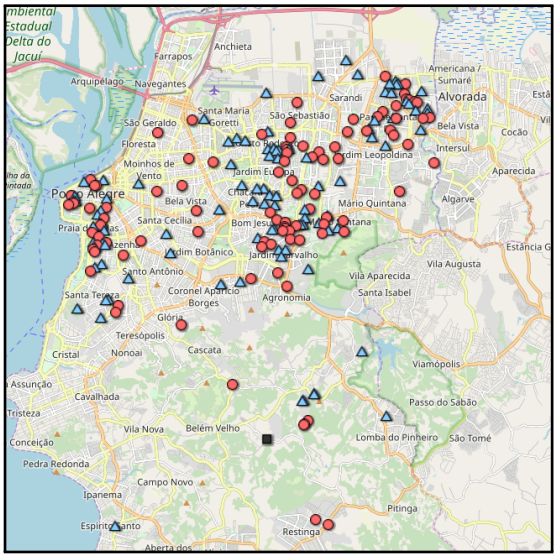


(a) Barcelona with 200 (R) locations.



(b) Berlin with 200 (C) locations.

(c) New York City with 100 (R) locations.



(d) Porto Alegre with 200 (RC) locations.

Source: The Author[3].

[3]Through the GPSVisualizer tool, by Adam Schneider: <https://www.gpsvisualizer.com/>,

6 COMPUTATIONAL EXPERIMENTS

In this chapter, we present the computational experiments to attest the performance of the proposed matheuristic. In Section 6.1, the computational environment and general configurations for the experiments are detailed. The selection of parameters for the algorithm is reported in Section 6.2. The contributions of each component are analyzed in Section 6.3. A comparison with the current *state-of-the-art* metaheuristics is discussed in Section 6.4. Section 6.5 presents the findings related to the new benchmark instances for the PDPTW. Additionally, the results in this chapter are available online[1].

6.1 Environment and Configurations

The experiments were implemented in `C++` and compiled with `g++` version `7.3` and optimization flag `-O3`. For the mathematical programming component, CPLEX version 12.6 was used. The computational environment was equipped with an Intel i7 930 processor running at 2.8 GHz, four physical cores, 12 GB of RAM, and the operational system Ubuntu 18.04 LTS. All experiments were executed in single thread mode.

The stopping criterion in our experiments is running time. The condition is based on the total number of locations of the instances. For sizes 100, 200, 400, 600, 800, and 1000, the maximum running time was 5, 15, 15, 30, 60, and 60 minutes, respectively. These times were used to better compare with the algorithm of Curtois et al. (2017), which uses the same criterion. In the case of the new instances, all those with more than 1000 locations were executed with a timeout of 60 minutes. The MP component was allowed one minute execution time per call in the matheuristic.

6.2 Parameter Tuning

The proposed matheuristic has a number of parameters values that need to be selected. To that end, we employed an automatic algorithm configuration tool called `irace` to tune the parameters. The procedure applies statistical tests to decide the best performant combination of values for all parameters over a training benchmark set. For the training set, we used a randomly selected sample from the Li and Lim (2003) instances.

[1] Online repository with complete and detailed results: <https://github.com/cssartori/math-pdptw/>

Table 6.1 presents for each parameter its notation and description, the range of its possible values, and the value reported in the best configuration by `irace`. A total of 2000 executions of the algorithm were allowed, while the remaining settings of `irace` were the default. The value n stands for the number of requests in the PDPTW instance, that is, some parameters are proportions related to the size of the instance.

One question that arises when proposing an improvement based matheuristic is whether the MP component provides a significant refinement. There are two ways to verify this hypothesis. First, one may compare the solution quality of the algorithm with and without the MP component. Although, it might be that the performance of the MP phase is related to other parameters of the heuristic. A second option is to execute a parameter tuning to verify at the same time which is the best configuration, and whether it is worth the additional overhead of the MP phase. In this work, we perform both tests. The first in Section 6.3, and the second in this section. This is the reason we have allowed `irace` to decide on the use of the SP phase.

Table 6.1: Tuned parameters and their respective values.

Notation	Description	Range	Best
`sp`	use SP model	{`true`,`false`}	`true`
Z_M	Matheuristic perturbation size	$[0.0, 1.0] \cdot n$	$0.83 \cdot n$
μ	Perturbation bias	[0.20,0.80]	0.57
M_A	AGES maximum number of perturbations	$[2, 6] \cdot 10^3$	$4 \cdot 10^3$
Z_A	AGES perturbation size	$[0.0, 1.0] \cdot n$	$0.15 \cdot n$
M_L	LNS maximum number of iterations	[500,1500]	971
L	LAHC acceptance list size	[1000,2000]	1539
b_{min}	LNS minimum removal size	[1,5]	2
b_{max}	LNS maximum removal size	$[0.10, 0.40] \cdot n$	$0.20 \cdot n$
$\{\omega_s, \omega_r, \omega_w\}$	LNS weights for heuristic selection	[0,10]	{6,3,1}
k_{min}	LNS mininum k-regret size	[1,6]	1
k_{max}	LNS maximum k-regret size	[1,6]	4

Source: The Author.

The `sp` parameter was set to `true`, which means that `irace` considered the MP phase of the matheuristic worth using. The remaining configurations are in accordance with the expected values. For instance, parameter Z_A takes a small portion of the requests, and it is smaller than parameter Z_M as previously discussed. The bias μ was assigned a value smaller than 1.0, as expected for a disruptive operation. The LNS parameters M_L and L are in accordance to previous works in the literature (CURTOIS et al., 2017; SARTORI; BURIOL, 2018). The values of b_{min} and b_{max} are smaller than of Ropke and Pisinger (2006) but the implementation is different as well, thus these values

are acceptable. The weights ω show that the Shaw heuristic should be performed more frequently, followed by the random removal, while the least frequent should be the worst removal heuristic, due to its limited usage. Parameter k_{min} indicates that it is worth the use of the greedy reinsertion heuristic (1-regret) in the LNS.

For all of the experimental results to follow, we have made use of the parameter values reported in Table 6.1, unless otherwise stated.

6.3 Statistical Tests and Component Analysis

To strengthen our study, we perform statistical tests and component analysis to verify whether our additional MP component and modified perturbation mechanism are significant. Furthermore, we discuss the behavior and performance of the components.

The analysis of components is conducted with a one-factor-at-a-time method. For each test, we change one element of the matheuristic to verify its contributions to the overall algorithm. The procedure produces four algorithms out of the main heuristic. Table 6.2 presents the difference in components to each one of the four algorithms. The SP term denotes the MP component. The original AGES algorithm as proposed by Curtois et al. (2017) is referred to as AGESa, while our modification as AGESb. The importance of the LNS component was demonstrated by Curtois et al. (2017), hence we have not performed this analysis.

Table 6.2: Algorithms used in the experiments.

Algorithm	SP	AGESa	AGESb
A1	•	-	•
A2	•	•	-
A3	-	-	•
A4	-	•	-

Source: The Author.

For simplicity, we named the algorithms A1, A2, A3, and A4. The matheuristic proposed in this thesis is Algorithm A1, which makes use of the SP and AGESb components. Algorithm A2 is the matheuristic that uses the original AGESa. Algorithms A3 and A4 do not use the SP component. The difference is that A3 uses the modified AGESb, while A4 uses the original AGESa. In that way, we can test the independent performance of both the MP and the modified AGES to guarantee they are relevant.

In the case of A2 and A4 that use AGESa with the original perturbation mechanism, we have set the parameters of the AGESa using the results of the tunning procedure of our preliminary work (SARTORI; BURIOL, 2018) to make fair comparisons. Therefore, the AGESa has $M_A = 1,000,000$ maximum perturbations, and $Z_A = 100$ random shifts or swaps of requests, using a probability $p_{shift} = 0.58$ to perform a random shift. The swap movement selects two requests at random from two different routes and exchanges their routes, inserting each request at its best position in the other route.

Table 6.3 presents the results of each algorithm per instance size in a single run. Due to the large size of the standard set, we ran each algorithm once for each one of the 354 instances following the analysis of Birattari (2004). The table reports the accumulated sum of the results for each instance size – this is the standard approach to summarize results in the literature of the PDPTW. In each row, the accumulated number of vehicles (Veh.) and total cost (Cost) are presented. Detailed results for every instance in the Li and Lim (2003) set are available in Appendix A and in the online repository.

The four algorithms found the same solutions for instances of size 100 – which are the best-known solutions of the 100 locations set. For the remaining sizes, the simplified comparison of the accumulated values demonstrates that algorithm A1 was able to outperform the others in most cases. The sole exception was on instances of size 600, in which algorithm A2 was able to find two fewer vehicles. Nevertheless, the simplified analysis shows small differences between the solutions, particularly for the number of routes, that differ by at most six in absolute terms (A2 and A3 for size 600). Indeed, one could even argue that the performance of the four algorithms is similar.

However, if we investigate the number of times each algorithm found the best solution among the four methods, we can note substantial differences in their performances. For example, A1 was able to find 85 solutions better than the others, while the second-best performant algorithm, A2, found only 44. Furthermore, A4, which does not use any of the modifications proposed, presents the worst performance of the four algorithms.

In addition to the previous analysis, we have performed a pairwise Wilcoxon signed-rank statistical test to compare the solution quality of the methods. Table 6.4 presents the p-values of the pairwise statistical test. In the comparison of two algorithms Ai and Aj ($i, j \in \{1, 2, 3, 4\}$), the null hypothesis is that Ai and Aj do not present different solutions, whereas the alternative hypothesis is that $Ai > Aj$, in other words, the solutions of Ai are statistically greater than those of Aj – for a minimization problem, it means Ai is worse than Aj.

Table 6.3: Results of the component analysis per instance size. Bold results are the best solutions found.

	A1		**A2**		**A3**		**A4**	
Inst.	Veh.	Cost	Veh.	Cost	Veh.	Cost	Veh.	Cost
100	**402**	**58,059.55**	**402**	**58,059.55**	**402**	**58,059.55**	**402**	**58,059.55**
200	**601**	**184,899.04**	601	185,499.00	601	185,598.65	601	185,351.09
400	**1139**	**442,438.04**	1142	445,740.67	1141	442,345.00	1141	447,196.24
600	1641	914,106.45	**1639**	**920,605.83**	1645	915,765.46	1642	918,475.22
800	**2135**	**1,506,254.99**	2135	1,526,631.67	2135	1,510,845.99	2136	1,535,762.44
1000	**2606**	**2,208,991.52**	2606	2,221,614.67	2607	2,222,585.11	2609	2,223,193.65
Best		85		44		39		18

Source: The Author.

Table 6.4: Pairwise Wilcoxon signed-rank test comparing the four algorithms in the standard instances.

	A1	A2	A3
A2	0.00169	-	-
A3	< 0.00001	0.31112	-
A4	< 0.00001	0.00087	0.06888

Source: The Author.

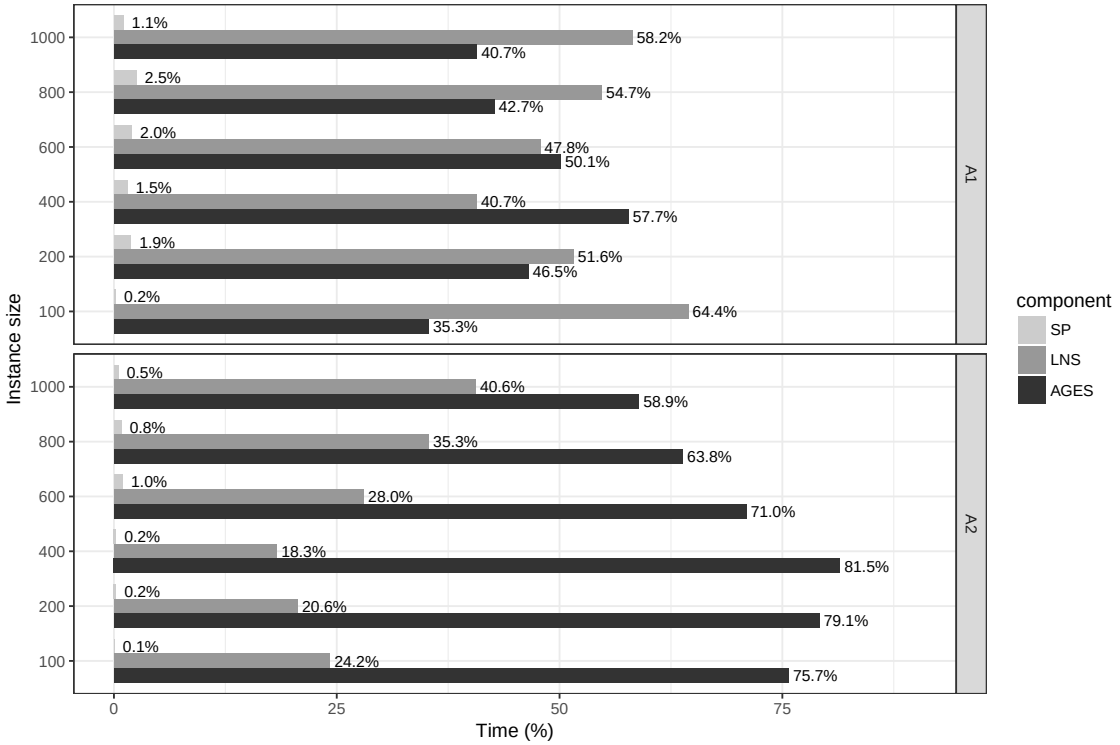
The following analysis of Table 6.4 is done considering a significance level of 0.01. It is possible to consider A1 statistically better than A2, A3, and A4. The simple addition of the MP phase in A2 also presents a significant improvement over A4. However, using only AGESb in A3 and only SP in A2 does not provide relevant differences. Furthermore, the modification in AGESb alone does not incur in significant improvements over AGESa.

The conclusion is that combining the SP and AGESb provides better solutions when compared to the other three algorithms. The next sections discuss the characteristics of A1 that contribute to improved performance.

6.3.1 Analysis of the Adaptive Guided Ejection Search

Figure 6.1 depicts the average percentual time spent by each one of the three main components in algorithms A1 and A2 – AGES, LNS, and SP phase. The AGESa (A2) requires more time than AGESb (A1). The former spends an average of 70% of the time in all instance sizes, whereas the latter spends an average of 45% of the time. A side effect of this difference is the amount of time spent by the remaining components.

Figure 6.1: Comparison between the time spent on each component of the matheuristic when using the biased (A1) and the original (A2) perturbation within the AGES.



Source: The Author.

Note that in A1, the LNS phase spends an average of 53% of the total time. In A2, LNS spends only 28%. Therefore, the LNS in A1 can improve the total cost further, which is one of the main features of the algorithm that can find many new solutions with a reduced total cost, but not with fewer vehicles. Analoguous, the SP formulation is able to spend more time trying to recombine previously visited routes – a detailed analysis of the MP phase is provided in Section 6.3.2.

Arguably the difference between AGESa and AGESb could be caused by the parameters chosen for AGESa. However, our experiments have shown that simply reducing the parameters of AGESa in the same manner as AGESb does not provide better solutions. Table 6.5 compares a new algorithm, A5, to A1 and A2. Algorithm A5 uses AGESa but with the parameters of AGESb (A1). The results demonstrate that A5 is not competitive with the other two algorithms. In our experiments, reducing the maximum number of perturbations for AGESa consistently produced poor results.

Table 6.5: Performance of the original AGES with modified parameters.

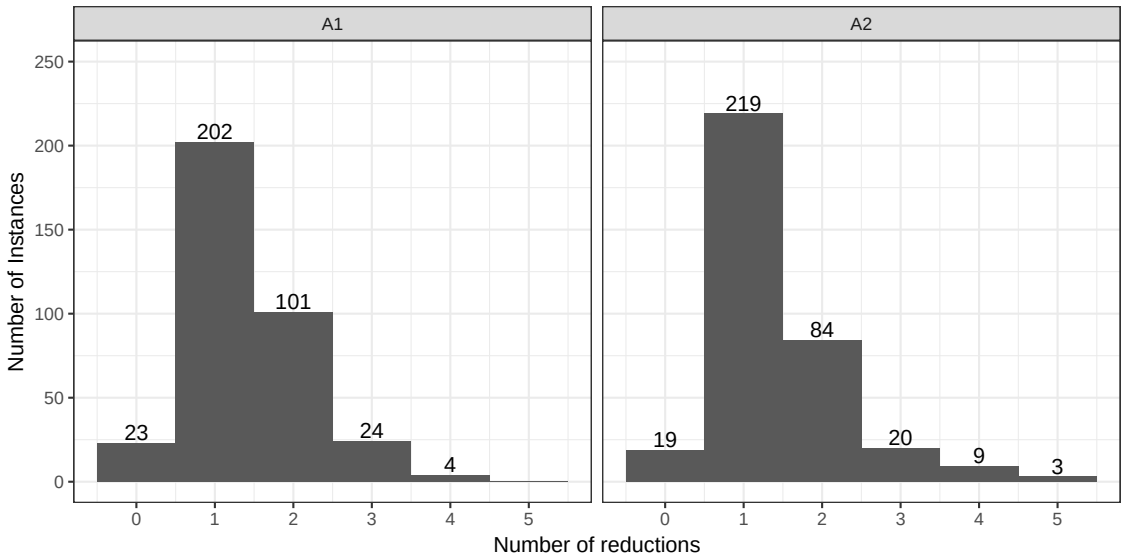
	A1		**A2**		**A5**	
Inst.	#V	Cost	#V	Cost	#V	Cost
100	402	58,059.55	402	58,059.55	402	58,059.55
200	601	184,899.04	601	185,499.00	606	181,715.87
400	1139	442,438.04	1142	445,740.67	1163	419,814.14
600	1641	914,106.45	1639	920,605.83	1674	849,971.38
800	2135	1,506,254.99	2135	1,526,631.67	2189	1,399,484.54
1000	2606	2,208,991.52	2606	2,221,614.67	2680	2,066,081.53

Source: The Author.

An unexpected outcome of the AGES performance was observed by verifying the total number of times the component improved the solution. In other words, how many times was it able to reduce the number of vehicles? The answer is depicted in Figure 6.2. In a few cases (about 5%) the AGES was not able to reduce the number of vehicles. But in the majority of the instances (nearly 60%), the AGES was able to reduce the number of vehicles only once. At most, it reduced four and five times for A1 and A2, respectively. Note that these are the number of times a call to AGES reduced the routes by at least one and not the total number of vehicles removed by the component.

The behavior of the AGES was unexpected because it is called in every iteration of the algorithm. The average number of iterations range from 100 (for size 1000) up to 600 (for size 200) – for more information on the number of iterations, see Table 6.6 below. From these hundreds of iterations, at most in five of them the AGES is able to reduce the number of routes, a small percentage for such a time consuming component.

Figure 6.2: Graphical statistics on the number of times the AGES reduced the number of vehicles for algorithms A1 and A2.



Source: The Author.

In consideration of that, the algorithms could be optimized to halt the use of AGES after a number of reductions. For example, we could set the condition to three reductions as per Figure 6.2 and risk losing solution quality in at most 12 instances (for A2). However, it would not change much the results, since the cases that reduce at most one or two times would not benefit from this optimization. Alternatively, we could allow AGES to reduce at most once, albeit this would incur a significant loss in solution quality. Despite the referred problems, our main question is whether this behavior is inherent to the AGES, or inherent to the instance set. The topic is discussed further in Section 6.5 in the context of the new benchmark instances.

6.3.2 Analysis of the Mathematical Programming Component

The SP phase recombines routes added to the pool $\mathcal{P}$. In fact, the mathematical formulation depends directly on the size of set $\mathcal{P}$, because a small set does not provide enough routes to improve the current solution, while a large pool may impact on the total execution time. From the graphics in Figure 6.1, it is possible to note that in our matheuristic the SP phase does not use more than 3% of the maximum running time. The previous results indicated the relevance of using the SP model, but did not contribute to understand its behavior and performance.

In Table 6.6 we investigate some statistics about the use of the SP formulation. For each one of the four algorithms, we present the total number of iterations (Iter.), the number of times SP was able to improve the solution quality (SPI), and the final size of the pool of routes ($|\mathcal{P}|$). Note that A3 and A4 do not have columns related to the SP phase, because they do not use it.

Table 6.6: Metrics about the four tested algorithms.

	A1			**A2**			**A3**	**A4**
Inst.	Iter.	SPI	$\lvert\mathcal{P}\rvert$	Iter.	SPI	$\lvert\mathcal{P}\rvert$	Iter.	Iter.
100	33,399	1	1,161	12,056	0	1,095	34,534	12,294
200	39,597	95	13,179	16,560	40	5,930	40,801	17,000
400	14,162	331	24,576	6,880	211	11,859	14,200	6,984
600	10,781	520	45,715	6,983	359	25,331	10,825	7,005
800	9,842	756	59,222	8,267	586	39,891	10,065	8,366
1000	6,032	724	50,935	4,990	581	35,052	6,088	5,047

Source: The Author.

There is a clear difference in the pool size $|\mathcal{P}|$ when comparing algorithms A1 and A2, particularly for instances with more than 200 locations. The pool size and number of iterations are consistently larger for A1 when compared to A2. In fact, there is a correlation with the number of improvements of the SP model, which is also greater for A1. These metrics indicate that the reduction in the AGES time and the increased pool size for the SP formulation are the reasons for the good performance of algorithm A1.

Note that the number of iterations for A3 and A4 are greater than for A1 and A2, respectively. This is expected because in A3 and A4 there is no SP phase, which can cause overhead to the execution of the algorithm.

6.4 Comparison with other Methods

In this section, we compare the results of the matheuristic (A1) with the current *state-of-the-art* methods in the literature of the PDPTW. These methods are the Guided Ejection Search of Nagata and Kobayashi (2010a) denoted as GES, and the hybrid algorithm of Curtois et al. (2017) denoted as CLSQL.

It is usually hard to compare with other methods due to the lack of complete results, or homogeneous computational environments. On the one hand, the GES heuristic was only applied to the minimization of vehicles, and final solution costs were not reported. Further, the GES was run five times for each instance with a maximum running

time of 10 minutes per run in an AMD Opteron 2.6 GHz processor – both average and best results were reported. Due to the limited article length, the authors only reported accumulated values for the vehicles, which is hard to use in detailed comparisons. The hybrid algorithm of CLSQL, on the other hand, was run only once per instance, but complete results are available. Therefore, we were able to perform more tests to compare the two algorithms. The computer environment of CLSQL was an Intel Xeon CPU E5-1620 running at 3.5 GHz, and the running times are the same used in this thesis.

For our algorithm, A1, we have run it a total of ten times for instances containing 100, 200, 400, and 600 locations, and five times for 800 and 1000. The average results obtained are referred to as $A1_{avg}$. Table 6.7 details the average results of GES in 5 runs, the results for a single execution of CLSQL, our results in a single execution (A1), and our average results in multiple runs ($A1_{avg}$). Columns are the same as in previous sections.

Table 6.7: Comparison of A1 with other methods from the literature.

	GES		**CLSQL**		**A1**		**$A1_{avg}$**	
Inst.	Veh.	Cost	Veh.	Cost	Veh.	Cost	Veh.	Cost
100	-	-	402	58,163.27	402	58,059.55	402.0	58,059.55
200	601.2	-	601	186,158.61	601	184,899.04	600.9	185,307.83
400	1140.0	-	1142	447,627.43	1139	442,438.04	1140.7	444,680.53
600	1641.8	-	1643	935,948.36	1641	914,106.45	1643.5	909,991.24
800	2147.4	-	2146	1,551,495.36	2135	1,506,254.99	2134.8	1,506,048.14
1000	2624.8	-	2634	2,310,830.27	2606	2,208,991.52	2605.4	2,218,717.76

Source: The Author.

The average results $A1_{avg}$ demonstrate that the method is stable for instances of size 100 because in all ten runs the results reported are the same. For other instance sizes, the matheuristic $A1_{avg}$ was able to find solutions with fewer vehicles than the GES and CLSQL for sizes 200, 800, and 1000. Furthermore, the average cost of our solutions is smaller than the reported results of CLSQL in all cases.

Note that in comparison to the GES, our method uses more execution time (in the same way as CLSQL). Therefore, it is reasonable that our approach was able to find solutions with fewer vehicles, particularly for the larger instances. However, the GES is not concerned in reducing the total cost of the solution, and all of its running time is dedicated to the minimization of vehicles. The impact of considering the minimization of the cost is likely to incur the need for extra execution time of an algorithm.

In comparison with the single run A1, our method has found an accumulated number of vehicles smaller than that of CLSQL for all sizes larger than 200 locations. The total cost of A1, on the other hand, was smaller in all instance sizes when compared to CLSQL.

Although our algorithm was based on CLSQL, the proposed modifications have provided improvements over the previous method, according to these results. Since Curtois et al. (2017) made their results completely available, we were able to compare A1 and CLSQL in each instance separately. The results reveal that A1 was able to find 187 (52%) solutions better than the single run of CLSQL, and found 106 (30%) solutions that matched the results. In 61 instances A1 was not able to outperform CLSQL. A Wilcoxon signed-rank test comparing CLSQL $>$ A1 provides a result of $p < 0.00001$.

Additionally, we compare our results to the best-known solutions (BKS) reported at the SINTEF (2008) website[2]. It is, however, hard to compare purely with the reported BKS, because they are from various methods, many of them from proprietary software or unpublished works. Hence, for many of these methods, there is no information available on how the solutions were obtained, or how much time was actually required. Table 6.8 presents the best results reported by the GES among the five runs (GES*), the best results of CLSQL in multiple runs of 60 minutes each (CLSQL*), and the best results of A1 over the ten or five executions (A1*).

Table 6.8: Comparison between best solutions of BKS, GES, CLSQL, and A1.

	BKS		**GES***		**CLSQL***		**A1***	
Inst.	#V	Cost	#V	Cost	#V	Cost	#V	Cost
100	402	58,059.55	-	-	402	58,059.55	402	58,059.55
200	600	183,793.43	601	-	600	185,103.31	600	184,170.41
400	1130	440,053.34	1139	-	1137	437,879.65	1135	437,606.91
600	1621	900,410.27	1636	-	1637	906,202.65	1635	892,626.37
800	2109	1,506,529.54	2135	-	2129	1,511,357.67	2126	1,490,103.15
1000	2570	2,210,857.15	2613	-	2598	2,210,000.72	2596	2,198,716.33

Source: The Author.

When compared to the current BKS, none of the algorithms was able to match the results – except for instances with 100 locations. Both CLSQL and A1 were able to match the number of vehicles in instances with 200 locations, but not the total cost. For the remaining sizes, our method A1 differs from the BKS by 5 vehicles (400 locations) up to 26 vehicles (1000 locations) in total. Yet, if we analyze instances separately, the largest difference is 3 vehicles in instance `LC1_10_3`. In other words, despite the improvements in our method, it was not able to match the overall BKS. Although, to the best of our knowledge, there is no published work in the literature currently able to match them.

To better verify the behavior of our method, we present the results of algorithm A1 over the instances proposed by Ropke and Cordeau (2009) for exact solution methods.

[2]According to the solutions published until 08 February 2019.

The set, described in Chapter 5, is grouped in four types in a total of 40 instances with 30 up to 75 requests. Baldacci, Bartolini and Mingozzi (2011) solved 39 of these instances to optimality. We denote their method by the acronym BBM.

Table 6.9 presents a summary of the results over these instances. Column BBM presents the average time required by the exact method of Baldacci, Bartolini and Mingozzi (2011) to solve the instances of a given group, in seconds. Algorithm A1 was run 10 times for each instance with 300 seconds of execution time. The column gap(%) presents the percentual deviation between the average solution s_1 found by A1 and the best solution s_b (usually optimal) found by BBM, and it is computed as gap(%) $= 100(s_1 - s_b)/s_b$. The detailed results are available in Appendix B.

Table 6.9: Comparison considering the instances for exact methods.

	BBM	**A1$_{\text{avg}}$**	
Inst.	t(s)	gap(%)	t(s)
AA	721	0.01	300
BB[a]	157	0.01	300
CC	1,534	0.02	300
DD	9,465	1.85	300

[a]: Solution BB75 not solved to optimality with gap of 0.1%.

Source: The Authors.

In terms of solution quality, A1 reached solutions close to the optimal, on average. The largest average gap was 1.85%, due to instance group DD that contains an instance for which our method was not able to match the number of vehicles in all runs, hence a larger gap value. Nonetheless, for the other three groups, the deviation was close to 0.01%, which demonstrates that A1 finds high-quality solutions for these instances as well.

As for execution time, it is hard to compare A1 and BBM, due to the different computing power and characteristics of the algorithms. Indeed, an exact algorithm is likely to need more execution time on average to reach optimal solutions. The results in Table 6.9 suggest that A1 was able to obtain good quality solutions in less time when compared to the exact method of BBM, except for set BB, in which case their method was able to prove the optimality of all instances in a short amount of time. Although we have not properly evaluated it, A1 is likely to be able to reach similar solutions for these sets in less than 300 seconds.

6.5 Extended Experiments

The objective of this section is two-fold. First, we want to perform further experiments to confirm that the behavior verified in the Li and Lim (2003) instances is the same in other types of instances. Second, we aim to provide initial experiments for the new set of instances proposed in Chapter 5, including upper bounds for the solutions.

The experiments compare the four algorithms A1, A2, A3, and A4 using the parameters of Section 6.2. Due to the new large instances, we had to limit the size of the AGESb perturbation in A1 and A3 as $Z_A = \max\{150, 0.15 \cdot n\}$, and of the number of removed requests in the LNS for all algorithms $b_{max} = \max\{150, 0.20 \cdot n\}$ – Ropke and Pisinger (2006) employed a similar strategy in the context of the LNS. Table 6.10 presents the results in the same manner as in the previous sections. For each instance size, we present the accumulated sum of the number of vehicles (Veh.) for all the 25 instances of that size, as well as the total cost (Cost). Note that the costs are all integer values because the arc costs in the new set of instances are all integers. The maximum running times are 5, 15, 15, and 30 minutes for instances with 100, 200, 400, and 600 locations, respectively, and 60 minutes for all the other sizes. Detailed results of algorithm A1 for the new instances are available at Appendix C and in the online repository.

Different from the results for the Li and Lim (2003) instances, algorithm A1 was not the best performant of the four. In fact, A2 is the algorithm that obtained the best performance. It found the best-accumulated results in 8 out of 12 size groups. For sizes 100 and 400, A1 and A4 obtained the best performance, respectively. For sizes 4000 and 5000, A2 and A4, and A1 and A3 reached the same solution – the MP component did not affect these two groups of instances due to the limited computation time.

Moreover, the number of best solutions reached by the algorithms differs from the previous results. Algorithm A2 has the highest score, with 71, followed by A4 with 46 best results. Note that in the Li and Lim (2003) instances, A4 had the worst performance of the four algorithms. Algorithm A1 could find only 24 best solutions, whereas A3 has the worst performance, with 19 best results. Similarly to the analysis of the standard benchmark set, we have performed a pairwise Wilcoxon signed-rank test to compare the four algorithms. Table 6.11 presents the resulting p-values of the test. In a significance level of 0.01, we can confirm that A2 found better solutions than all the other algorithms and that both A2 and A4 outperformed A1 and A3 in terms of solution quality. Besides, we cannot assert that A1 and A3 are statistically different.

Table 6.10: Results of the four algorithms over the new instances. Bold results are the best solutions found.

	A1		**A2**		**A3**		**A4**	
Inst.	Veh.	Cost	Veh.	Cost	Veh.	Cost	Veh.	Cost
100	**164**	**25,388**	165	25,579	164	25,445	165	25,588
200	339	46,218	**337**	**46,587**	339	46,220	337	46,600
400	593	85,352	589	86,193	592	85,184	**589**	**85,887**
600	854	121,755	**840**	**122,130**	852	121,409	843	121,978
800	1167	162,907	**1150**	**163,341**	1167	162,673	1154	163,261
1000	1431	224,966	**1401**	**226,228**	1436	224,682	1406	226,152
1500	2160	301,794	**2115**	**303,478**	2155	302,410	2118	304,564
2000	2963	414,447	**2924**	**425,343**	2964	415,520	2928	426,507
2500	3270	502,309	**3201**	**506,436**	3271	502,976	3201	506,488
3000	4355	598,966	**4276**	**625,072**	4357	597,676	4278	623,979
4000	5991	834,365	**5944**	**866,430**	5991	834,365	**5944**	**866,430**
5000	6864	1,069,925	**6802**	**1,089,677**	6864	1,069,925	**6802**	**1,089,677**
Best		24		71		19		46

Source: The Author.

Table 6.11: Pairwise Wilcoxon signed-rank test comparing the four algorithms in the new instances.

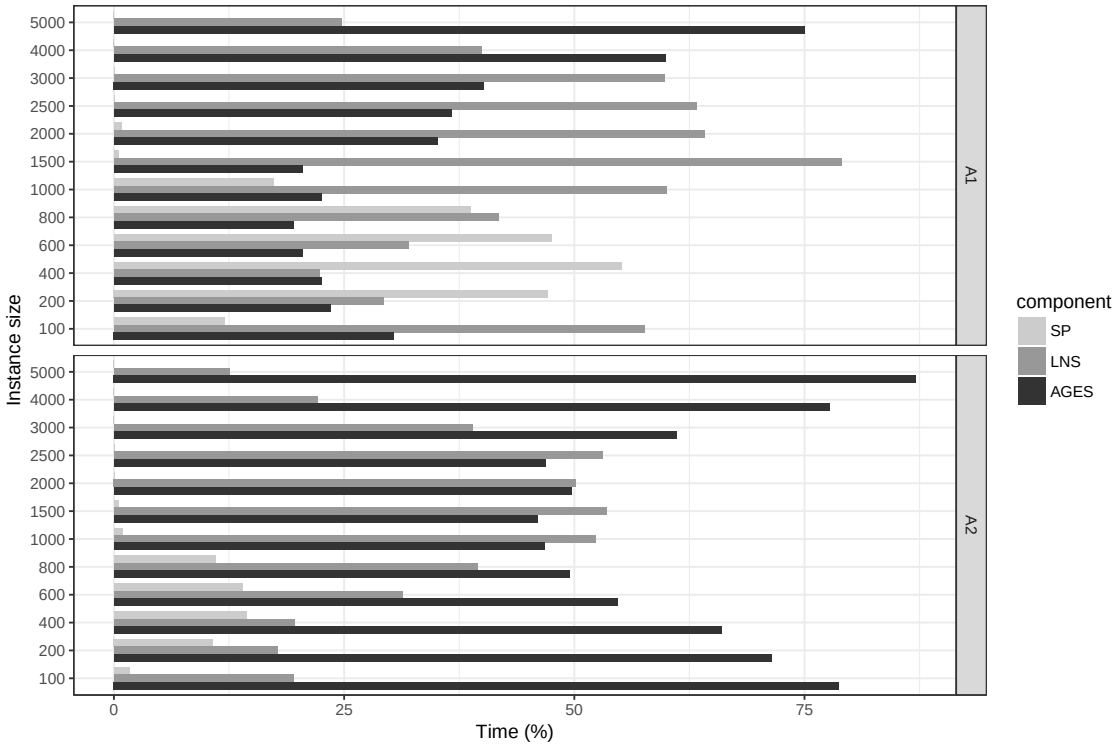
	A2	A4	A1
A4	0.00383	-	-
A1	< 0.00001	< 0.00001	-
A3	< 0.00001	< 0.00001	0.43540

Source: The Author.

6.5.1 Analysis of Components Applied to the New Instances

The evidence presents some disagreement with the results in Section 6.3. However, if we investigate further the reasons for these differences, it is possible to note specific characteristics that contributed to them. For example, Figure 6.3 depicts the percentual time spent on the three major components of the matheurisitic. When compared to the percentual times in Figure 6.1, it is possible to note a significant difference in the time spent by each phase of the algorithm.

Figure 6.3: Comparison of the time spent on the three major components in the context of the new instances by algorithms A1 and A2. Numeric labels were removed for improved visualization.

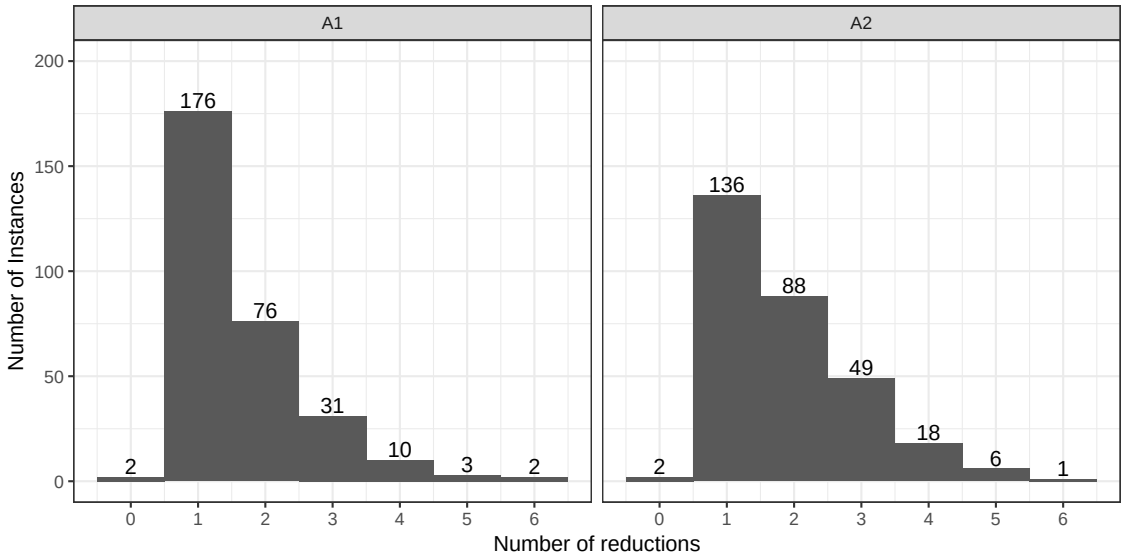


Source: The Author.

The behavior of the AGESb (A1) is still the same when compared to AGESa (A2), the former requires less computation time than the latter. Although, this appears to have a negative impact on solution quality because the total number of vehicles found by A1 is consistently greater than for A2 or A4. Indeed, spending more time to reduce the number of vehicles is more effective in the new instances than trying to balance vehicle and cost minimization, which worked well for the standard instances. In addition, the number of times AGES was able to reduce the number of vehicles is similar to what had been observed in the Li and Lim (2003) instances. Figure 6.4 presents the number of times A1

and A2 reduced the number of vehicles. In most cases, 84% for A1 and 74% for A2, the AGES was able to reduce at most twice the number of vehicles. Hence, the optimizations discussed in Section 6.3 could be tested, because the results analyzed here indicate that the behavior is inherent to the AGES applied.

Figure 6.4: Statistics for the number of times AGES was able to reduce the number of vehicles in the context of the new instances.



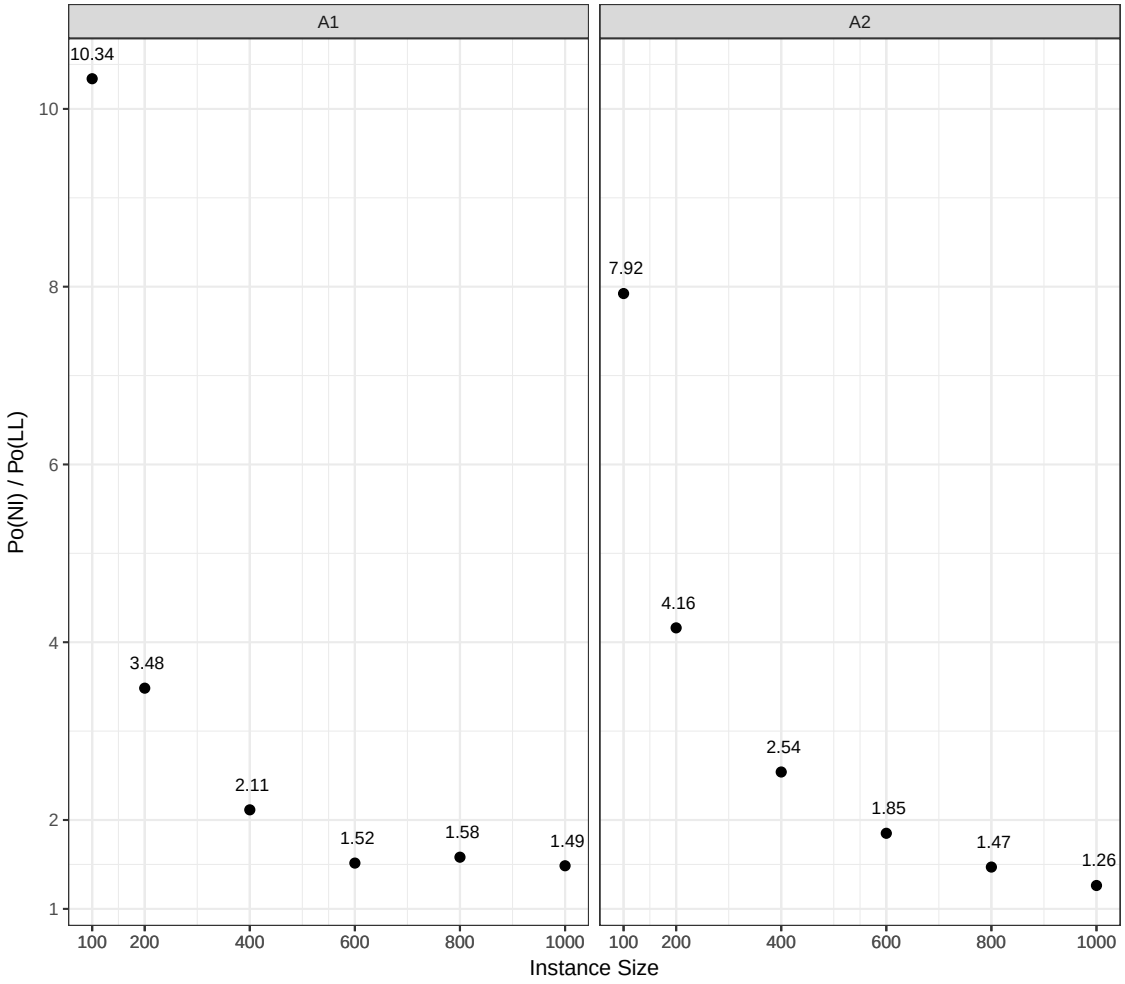
Source: The Author.

In the case of the SP model, there is a more significant discrepancy. For the Li and Lim (2003) instances, algorithm A1 spent an average of 2% of the overall time in the SP model, while in the new instances it spent an average of 18%. In the particular case of 400 locations, A1 spent 55% of the time in the model. Similarly, algorithm A2 spent an average of 0.5% in the standard instances, but in the new set, it spent an average of 4%, with a peak of 14% in instances with 400 locations as well. Further, the scaling of the MP component was quite limited, because, within the stopping conditions used, it was able to have a substantial effect only up to instances with 2000 locations.

The additional time required by the SP model can be explained by the number of routes in the pool $\mathcal{P}$. The average size of the pool for the new instances (denoted as Po(NI)) grows faster than for the Li and Lim (2003) instances (denoted as Po(LL)). Figure 6.5 depicts the proportion between the average sizes Po(NI) and Po(LL) for each one of the six basic instance sizes and for algorithms A1 and A2. For example, in instances with 400 locations A1 had a pool for the new instance twice the size for the standard instances. The difference is even larger for instances with 100 locations, in which case A1 had Po(NI) 10 times larger than Po(LL). Algorithm A2 had an analogous behavior, in

some cases with even larger proportions than A1, but the effects of a larger pool were not so prominent than for A1, which allowed A2 to outperform its counterpart A4 as well.

Figure 6.5: Analysis of the proportions between the average pool size in the new instance set Po(NI) and in the standard set Po(LL) for algorithms A1 and A2 in the six basic instance sizes.



Source: The Author.

6.6 Final Considerations

A possible explanation to the observed results is that the Li and Lim (2003) instances have a particular structure because of how they were generated, whereas the new benchmarks only have a structure in the clustered cases. One may argue that in the standard instances the search is confined to a certain region of the space of locations that lead to a reduction in the diversity of routes generated by the LNS phase and a consequent reduction in the pool of routes. This may be the case because the paired nodes originally

belonged to a VRPTW route, which is unlike to have large detours. On the other hand, in the new benchmarks, the pairs may be located far apart in the cities, and large detours are necessary to compute feasible solutions in such cases.

The existence of large detours in the solutions decreases the total number of requests serviced in each route and consequently increases the total number of routes in the solution. Table 6.12 presents the average number of requests per route (Avg. Req.) and the average number of routes (Avg. Rou.) in the BKS solutions of the Li and Lim (2003) and new instances. These results support the previous claim.

Table 6.12: Average number of requests per route and number of routes in solutions of the standard and new benchmark set of instances

	Li and Lim (2003)		New Instances	
Inst.	Avg. Req.	Avg. Rou.	Avg. Req.	Avg. Rou.
100	22.17	7.18	21.14	6.56
200	31.46	10.00	20.71	13.40
400	35.94	18.76	23.31	23.56
600	39.06	27.02	23.92	33.60
800	40.55	35.58	24.19	45.92
1000	41.76	44.31	26.12	55.92

Source: The Author.

Note that the Avg. Req. is always smaller for the new instances than for the Li and Lim (2003), whereas the Avg. Rou. is greater for the new instances than for the standard set, except for size 100. This confirms that more routes are required to service all requests in the new instances, which may be one reason for the larger pool of routes since at every iteration more routes will be added to the pool. A large number of smaller routes poses difficulties for the SP formulation, requiring more time to be solved. Therefore, it may be the case that the new instances are harder solely for the proposed matheuristic, but more experiments are required to validate this hypothesis.

Nevertheless, for the standard instances, a less aggressive AGES approach (AGESb in A1 and A3) was enough to reach a similar number of vehicles when compared to a more aggressive strategy (AGESa in A2 and A4). Instead, in the new instances, spending more time in the vehicle minimization phase does lead to significantly better results. Although we were not able to explore these topics further to reach stronger conclusions, the results present an initial base to raise the question of whether the two sets of instances independently provide a good discriminating factor for solution methods.

In summary, it appears that algorithm A1 was unintentionally tailored for the Li and Lim (2003). Sörensen (2015) has argued that some methods in metaheuristic research end up optimized for the standard benchmark set at hand. In fact, our research focused entirely on the Li and Lim (2003) instances. Once the algorithms were fully implemented and tested, we proceeded to perform analyses with the new instances. The reason, as previously noted, was because the standard set was the only set available with numerous instances and previous works to compare the solutions.

In spite of that, the MP component still proved to be useful in the new instances. Algorithm A2 was the second best performant method in the Li and Lim (2003) instances and the first in the new set. When compared to the current *state-of-the-art* method for the PDPTW (CLSQL) A2 found 162 improved solutions and had 107 ties, which is a good performance nonetheless. Hence, we may conclude that algorithm A2 had the best performance over all the 654 benchmarks.

7 CONCLUSIONS AND FUTURE WORK

In this work, we have proposed a matheuristic algorithm to solve the PDPTW. The method combines previous techniques from the literature with a mathematical programming component in an Iterated Local Search framework. The method uses a modified Adaptive Guided Ejection Search (CURTOIS et al., 2017) to reduce the number of vehicles, whereas to minimize the cost it uses Large Neighborhood Search (ROPKE; PISINGER, 2006) and a Set Partitioning formulation to recombine previously generated routes in search of an improved solution. Computational experiments over a standard benchmark set of instances demonstrate that the proposed matheuristic can improve on previous methods in terms of solution quality.

Component analysis and statistical test suggest that the mathematical programming component is significant in providing high-quality solutions. The solutions found when using the MP phase had a smaller cost than the algorithm without the component. The proposed modified perturbation mechanism applied to the AGES combined with the MP phase produced the best results to the standard instance set (LI; LIM, 2003) in our analysis. Our research found a total of 62 new best solutions.

Furthermore, a new set of instances for the PDPTW was proposed. The new instances were generated by a method that uses publicly available data for customer locations and travel times. Besides, we have introduced larger instances to test the scalability of solution methods in cases with more than 1000 locations. Although, the computational results of the proposed matheuristic over the new benchmarks presented some disagreement with the experiments with the standard set. The modified AGES proved less significant in the new benchmarks. The cause is possibly the differences in the characteristics of the two sets. In addition, the SP phase used larger portions of the execution time, probably due to the larger size of the pool of routes.

The experimental results, nevertheless, have raised questions that we were not able to explore in this thesis. For example, it has been observed that the AGES was able to reduce the number of vehicles only a limited number of times for all instances tested. Is this behavior inherent to the heuristic? No previous works had analyzed this, even though the GES had been successfully applied to the VRPTW as well (NAGATA; BRÄYSY, 2009). Additionally, could stopping the use of AGES after a given threshold on the number of reductions be able to improve the results significantly?

To prevent the SP phase of waisting too much execution time, further improvements can be considered. For example, a better pool management, by considering the removal of subset routes, i.e., if the requests visited by a route r_1 form a proper subset of another route r_2, but $C(r_1) > C(r_2)$, we may remove the shorter route r_1. Another option is to use an aging mechanism to remove routes from the pool that are not used after a number of iterations. At last, it may be more efficient to use a specific procedure to solve the SP instead of a generic mathematical programming solver.

In the context of the benchmarks tests, is it possible that the procedure used to generate the Li and Lim (2003) instances influences the solution methods applied to solve them? In our experiments, the use of a less time-consuming AGES was able to find a similar number of vehicles for the standard instances when compared to a more time-consuming version of the same component. But for the new set, spending more time on the vehicle minimization phase was a better strategy to find high-quality solutions.

Besides, the new instances provided a much larger number of routes to the pool of the MP component, whereas the standard instances provided a smaller pool, indicating that the former has a greater diversity of routes than the latter. Even though the SP formulation requires a certain amount of routes in the pool to work well, a larger pool impacted on the performance of the algorithm, especially when coupled with the modified and less aggressive AGES. It appears that there are characteristics in the current standard instances which lead to the observed behavior in contrast to the new benchmarks.

Additional experiments are certainly required to answer these questions, in particular referring to the differences between the two instance sets. The analyses should be used to understand better if the new instances have characteristics that motivate their use along the current instances to differentiate new solution methods. In other words, do the new benchmarks have good discriminating power for solution methods of the PDPTW?

At last, we note that the new instances may provide interesting cases for analyses such as the structure of the PDPTW solutions, particularly in the context of the cities. For example, we have noted that solutions for the instances in New York City used the least amount of vehicles due to the high concentration of requests in Manhattan. On the other hand, solutions in Berlin had the largest number of vehicles in 7 out of the 12 sizes. In the 5 remaining sizes, Porto Alegre had the largest number of routes. The solutions in these two cities also have the two largest total number of vehicles. Is there any particular characteristic that could lead to these results? Are they related to the generation of the instances, or do the locations have an impact on these observations?

REFERENCES

ALLAOUA, H. et al. A matheuristic approach for solving a home health care problem. **Electronic Notes in Discrete Mathematics**, v. 41, p. 471 – 478, 2013.

APPLEGATE, D. L. et al. **The traveling salesman problem: a computational study**. [S.l.]: Princeton university press, 2006.

ARCHETTI, C.; SPERANZA, M. G. A survey on matheuristics for routing problems. **EURO Journal on Computational Optimization**, Springer, v. 2, n. 4, p. 223–246, 2014.

ARNOLD, F.; GENDREAU, M.; SÖRENSEN, K. **Efficiently Solving Very Large Scale Routing Problems**. [S.l.]: CIRRELT, Centre interuniversitaire de recherche sur les réseaux d'entreprise, 2017.

BALDACCI, R.; BARTOLINI, E.; MINGOZZI, A. An exact algorithm for the pickup and delivery problem with time windows. **Operations research**, INFORMS, v. 59, n. 2, p. 414–426, 2011.

BALINSKI, M. L.; QUANDT, R. E. On an integer program for a delivery problem. **Operations Research**, INFORMS, v. 12, n. 2, p. 300–304, 1964.

BALL, M. O. Heuristics based on mathematical programming. **Surveys in Operations Research and Management Science**, Elsevier, v. 16, n. 1, p. 21–38, 2011.

BATTARRA, M.; CORDEAU, J.-F.; IORI, M. Chapter 6: pickup-and-delivery problems for goods transportation. In: **Vehicle Routing: Problems, Methods, and Applications, Second Edition**. [S.l.]: SIAM, 2014. p. 161–191.

BENT, R.; HENTENRYCK, P. V. A two-stage hybrid local search for the vehicle routing problem with time windows. **Transportation Science**, INFORMS, v. 38, n. 4, p. 515–530, 2004.

BENT, R.; HENTENRYCK, P. V. A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. **Computers & Operations Research**, Elsevier, v. 33, n. 4, p. 875–893, 2006.

BIRATTARI, M. **On the Estimation of the Expected Performance of a Metaheuristic on a Class of Instances**. [S.l.], 2004. Technical Report TR/IRIDIA/2004-01, IRIDIA, Université Libre de Bruxelles.

BOSCHETTI, M. A. et al. Matheuristics: Optimization, simulation and control. In: BLESA, M. J. et al. (Ed.). **Hybrid Metaheuristics**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 171–177.

BRÄYSY, O.; GENDREAU, M. Vehicle routing problem with time windows, part i: Route construction and local search algorithms. **Transportation science**, INFORMS, v. 39, n. 1, p. 104–118, 2005.

BUREAU OF TRANSPORTATION STATISTICS. **Freight Facts and Figures**. 2017. Data available online at <https://www.bts.gov/sites/bts.dot.gov/files/docs/FFF_2017.pdf>, page 35, last accessed 2017-11-14.

BURKE, E. K.; BYKOV, Y. A late acceptance strategy in hill-climbing for exam timetabling problems. In: **PATAT 2008 Conference, Montreal, Canada**. [S.l.: s.n.], 2008.

CAPUA, R. et al. A study on exponential-size neighborhoods for the bin packing problem with conflicts. **Journal of Heuristics**, Springer, v. 24, n. 4, p. 667–695, 2018.

CHRISTIAENS, J.; Vanden Berghe, G. **A fresh ruin & recreate implementation for the capacitated vehicle routing problem**. [S.l.], 2016. Technical Report. Katholieke Universiteit Leuven.

CHRISTIANSEN, M. et al. Chapter 4 maritime transportation. In: BARNHART, C.; LAPORTE, G. (Ed.). **Transportation**. [S.l.]: Elsevier, 2007, (Handbooks in Operations Research and Management Science, v. 14). p. 189 – 284.

CHRISTIANSEN, M. et al. Ship routing and scheduling in the new millennium. **European Journal of Operational Research**, v. 228, n. 3, p. 467 – 483, 2013. ISSN 0377-2217.

CONFEDERAÇÃO NACIONAL DO TRANSPORTE. **Boletim Estatístico CNT: Junho**. 2017. Data available online at <http://www.cnt.org.br/Boletim/boletim-estatistico-cnt>, last accessed 2017-11-14.

CORDEAU, J.-F.; MAISCHBERGER, M. A parallel iterated tabu search heuristic for vehicle routing problems. **Computers & Operations Research**, Elsevier, v. 39, n. 9, p. 2033–2050, 2012.

CORMEN, T. H. et al. **Introduction to algorithms**. [S.l.]: MIT press, 2009.

CURTOIS, T. et al. Large neighbourhood search with adaptive guided ejection search for the pickup and delivery problem with time windows. **EURO Journal on Transportation and Logistics**, Springer, p. 1–42, 2017.

DANTZIG, G. B.; RAMSER, J. H. The truck dispatching problem. **Management science**, INFORMS, v. 6, n. 1, p. 80–91, 1959.

DELLA CROCE, F.; SALASSA, F. A variable neighborhood search based matheuristic for nurse rostering problems. **Annals of Operations Research**, v. 218, n. 1, p. 185–199, Jul 2014.

DOERNER, K. F.; SALAZAR-GONZÁLEZ, J.-J. Chapter 7: Pickup-and-delivery problems for people transportation. In: **Vehicle Routing: Problems, Methods, and Applications, Second Edition**. [S.l.]: SIAM, 2014. p. 193–212.

DONOVAN, B.; WORK, D. **New York City Trip Data (2010 to 2013)**. University of Illinois at Urbana-Champaign, 2016. Available from Internet: <https://doi.org/10.13012/J8PN93H8>.

DORNELES Árton P.; ARAÚJO, O. C. de; BURIOL, L. S. A fix-and-optimize heuristic for the high school timetabling problem. **Computers & Operations Research**, v. 52, p. 29 – 38, 2014.

DUMAS, Y.; DESROSIERS, J.; SOUMIS, F. The pickup and delivery problem with time windows. **European Journal of Operational Research**, Elsevier, v. 54, n. 1, p. 7–22, 1991.

EUROPEAN COMMISSION. **Statistical Pocketbook 2016: EU Transport in Figures**. 2016. Data available online at <https://ec.europa.eu/transport/sites/transport/files/pocketbook2016.pdf>, page 19, last accessed 2017-11-14.

FURTADO, M. G. S.; MUNARI, P.; MORABITO, R. Pickup and delivery problem with time windows: a new compact two-index formulation. **Operations Research Letters**, Elsevier, v. 45, n. 4, p. 334–341, 2017.

GEHRING, H.; HOMBERGER, J. A parallel two-phase metaheuristic for routing problems with time-windows. **Asia Pacific Journal of Operational Research**, National University of Singapore, v. 18, n. 1, p. 35–48, 2001.

GENDREAU, M.; POTVIN, J.-Y. **Handbook of metaheuristics**. [S.l.]: Springer, 2010.

HASLE, G.; LIE, K.-A.; QUAK, E. **Geometric Modelling, Numerical Simulation, and Optimization - Applied Mathematics at SINTEF**. [S.l.]: Springer Science & Business Media, 2007.

HOMSI, G. et al. Industrial and tramp ship routing problems: Closing the gap for real-scale instances. **arXiv preprint arXiv:1809.10584**, 2018.

INTERNATIONAL ENERGY AGENCY. **CO2 emissions from fuel combustion overview**. 2018. Data available online at <https://webstore.iea.org/co2-emissions-from-fuel-combustion-2018-overview>, last accessed 2018-10-22.

KARP, R. M. Reducibility among combinatorial problems. In: ____. **Proceedings of a symposium on the Complexity of Computer Computations**. Boston, MA: Springer US, 1972. p. 85–103.

KONING, D. **Using column generation for the pickup and delivery problem with disturbances**. Dissertation (Master) — Universiteit Utrecht, 2011. 29 pages. MSc in Computer Science.

KRAMER, R. et al. A matheuristic approach for the pollution-routing problem. **European Journal of Operational Research**, Elsevier, v. 243, n. 2, p. 523–539, 2015.

LAPORTE, G. What you should know about the vehicle routing problem. **Naval Research Logistics (NRL)**, Wiley Online Library, v. 54, n. 8, p. 811–819, 2007.

LI, H.; LIM, A. A metaheuristic for the pickup and delivery problem with time windows. **International Journal on Artificial Intelligence Tools**, v. 12, n. 02, p. 173–186, 2003.

LOURENÇO, H. R.; MARTIN, O. C.; STÜTZLE, T. Iterated local search: Framework and applications. In: GENDREAU, M.; POTVIN, J.-Y. (Ed.). **Handbook of Metaheuristics**. Boston, MA: Springer US, 2010. p. 363–397.

LU, Q.; DESSOUKY, M. An exact algorithm for the multiple vehicle pickup and delivery problem. **Transportation Science**, INFORMS, v. 38, n. 4, p. 503–514, 2004.

LUXEN, D.; VETTER, C. Real-time routing with openstreetmap data. In: **Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems**. New York, NY, USA: ACM, 2011. (GIS '11), p. 513–516.

MANIEZZO, V.; STÜTZLE, T.; VOSS, S. **Matheuristics: Hybridizing Metaheuristics and Mathematical Programming**. [S.l.]: Springer US, 2009. (Annals of Information Systems).

MERZ, P.; HUHSE, J. An iterated local search approach for finding provably good solutions for very large tsp instances. In: SPRINGER. **International Conference on Parallel Problem Solving from Nature**. [S.l.], 2008. p. 929–939.

MORAIS, V. W.; MATEUS, G. R.; NORONHA, T. F. Iterated local search heuristics for the vehicle routing problem with cross-docking. **Expert Systems with Applications**, Elsevier, v. 41, n. 16, p. 7495–7506, 2014.

NAGATA, Y.; BRÄYSY, O. A powerful route minimization heuristic for the vehicle routing problem with time windows. **Operations Research Letters**, Elsevier, v. 37, n. 5, p. 333–338, 2009.

NAGATA, Y.; KOBAYASHI, S. Guided ejection search for the pickup and delivery problem with time windows. In: SPRINGER. **European Conference on Evolutionary Computation in Combinatorial Optimization**. [S.l.], 2010. p. 202–213.

NAGATA, Y.; KOBAYASHI, S. A memetic algorithm for the pickup and delivery problem with time windows using selective route exchange crossover. In: SPRINGER. **International Conference on Parallel Problem Solving from Nature**. [S.l.], 2010. p. 536–545.

NANRY, W. P.; BARNES, J. W. Solving the pickup and delivery problem with time windows using reactive tabu search. **Transportation Research Part B: Methodological**, Elsevier, v. 34, n. 2, p. 107–121, 2000.

OpenAddresses. **The free and open global address collection**. 2017. <https://openaddresses.io/>.

OpenStreetMap. **OSM contributors. Planet dump retrieved from https://planet.osm.org** . 2017. <https://www.openstreetmap.org>.

PARRAGH, S. N.; SCHMID, V. Hybrid column generation and large neighborhood search for the dial-a-ride problem. **Computers & Operations Research**, v. 40, n. 1, p. 490 – 497, 2013.

PECIN, D. et al. Improved branch-cut-and-price for capacitated vehicle routing. **Mathematical Programming Computation**, Springer, v. 9, n. 1, p. 61–100, 2017.

PENNA, P. H. V. et al. A hybrid heuristic for a broad class of vehicle routing problems with heterogeneous fleet. **Annals of Operations Research**, Springer, p. 1–70, 2017.

PIECYK, M. I.; MCKINNON, A. C. Forecasting the carbon footprint of road freight transport in 2020. **International Journal of Production Economics**, Elsevier, v. 128, n. 1, p. 31–42, 2010.

PISINGER, D.; ROPKE, S. Large neighborhood search. In: **Handbook of metaheuristics**. [S.l.]: Springer, 2010. p. 399–419.

REINELT, G. Tsplib—a traveling salesman problem library. **ORSA journal on computing**, INFORMS, v. 3, n. 4, p. 376–384, 1991.

RODRIGUE, J.-P.; COMTOIS, C.; SLACK, B. **The geography of transport systems**. [S.l.]: Taylor & Francis, 2016.

ROPKE, S. **Heuristic and exact algorithms for vehicle routing problems**. Thesis (PhD) — University of Copenhagen, 2005. 256 pages. PhD in Computer Science.

ROPKE, S.; CORDEAU, J.-F. Branch-and-cut-and-price for the pickup and delivery problem with time windows. **Transportation Science**, INFORMS, v. 43, n. 3, p. 267–286, 2009.

ROPKE, S.; CORDEAU, J.-F.; LAPORTE, G. Models and branch-and-cut algorithms for pickup and delivery problems with time windows. **Networks**, Wiley Online Library, v. 49, n. 4, p. 258–272, 2007.

ROPKE, S.; PISINGER, D. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. **Transportation science**, INFORMS, v. 40, n. 4, p. 455–472, 2006.

RUIZ, R.; STÜTZLE, T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. **European Journal of Operational Research**, Elsevier, v. 177, n. 3, p. 2033–2049, 2007.

SARTORI, C. S.; BURIOL, L. S. A matheuristic approach to the pickup and delivery problem with time windows. In: CERULLI, R.; RAICONI, A.; VOSS, S. (Ed.). **Computational Logistics**. Cham: Springer International Publishing, 2018. p. 253–267.

SAVELSBERGH, M.; SOL, M. Drive: Dynamic routing of independent vehicles. **Operations Research**, INFORMS, v. 46, n. 4, p. 474–490, 1998.

SAVELSBERGH, M. W. The vehicle routing problem with time windows: Minimizing route duration. **ORSA journal on computing**, INFORMS, v. 4, n. 2, p. 146–154, 1992.

SCHRAGE, L. Formulation and structure of more complex/realistic routing and scheduling problems. **Networks**, Wiley Online Library, v. 11, n. 2, p. 229–232, 1981.

SHAW, P. Using constraint programming and local search methods to solve vehicle routing problems. In: SPRINGER. **International conference on principles and practice of constraint programming**. [S.l.], 1998. p. 417–431.

SHEN, Y. et al. A computer assistant for vehicle dispatching with learning capabilities. **Annals of operations research**, Springer, v. 61, n. 1, p. 189–211, 1995.

SINTEF. **Li & Lim Benchmark Instances**. 2008. Available online at <https://www.sintef.no/projectweb/top/pdptw/li-lim-benchmark/>, last accessed 2017-11-15.

SOLOMON, M. M. Algorithms for the vehicle routing and scheduling problems with time window constraints. **Operations research**, INFORMS, v. 35, n. 2, p. 254–265, 1987.

SÖRENSEN, K. Metaheuristics—the metaphor exposed. **International Transactions in Operational Research**, Wiley Online Library, v. 22, n. 1, p. 3–18, 2015.

STÜTZLE, T. Iterated local search for the quadratic assignment problem. **European Journal of Operational Research**, Elsevier, v. 174, n. 3, p. 1519–1539, 2006.

SUBRAMANIAN, A.; UCHOA, E.; OCHI, L. S. A hybrid algorithm for a class of vehicle routing problems. **Computers & Operations Research**, Elsevier, v. 40, n. 10, p. 2519–2531, 2013.

SUTCLIFFE, C.; BOARD, J. The ex-ante benefits of solving vehicle-routeing problems. **Journal of the Operational Research Society**, Taylor & Francis, v. 42, n. 2, p. 135–143, 1991.

TOTH, P.; VIGO, D. **Vehicle routing: problems, methods, and applications**. [S.l.]: SIAM, 2014.

UCHOA, E. et al. New benchmark instances for the capacitated vehicle routing problem. **European Journal of Operational Research**, Elsevier, v. 257, n. 3, p. 845–858, 2017.

WILSON, N. H.; WEISSBERG, R. W.; HAUSER, J. **Advanced dial-a-ride algorithms research project**. [S.l.], 1976.

APPENDIX A — DETAILED RESULTS FOR THE STANDARD INSTANCES

Table A.1: Results of algorithm A1 for the Li and Lim instances with 100 locations.

Instance	Veh.	Cost	Time (min.)	Instance	Veh.	Cost	Time (min.)	Instance	Veh.	Cost	Time (min.)
lc101	10	828.94	5	lr104	9	1013.39	5	lrc101	14	1708.80	5
lc102	10	828.94	5	lr105	14	1377.11	5	lrc102	12	1558.07	5
lc103	9	1035.35	5	lr106	12	1252.62	5	lrc103	11	1258.74	5
lc104	9	860.01	5	lr107	10	1111.31	5	lrc104	10	1128.40	5
lc105	10	828.94	5	lr108	9	968.97	5	lrc105	13	1637.62	5
lc106	10	828.94	5	lr109	11	1208.96	5	lrc106	11	1424.73	5
lc107	10	828.94	5	lr110	10	1159.35	5	lrc107	11	1230.14	5
lc108	10	826.44	5	lr111	10	1108.90	5	lrc108	10	1147.43	5
lc109	9	1000.60	5	lr112	9	1003.77	5	lrc201	4	1406.94	5
lc201	3	591.56	5	lr201	4	1253.23	5	lrc202	3	1374.27	5
lc202	3	591.56	5	lr202	3	1197.67	5	lrc203	3	1089.07	5
lc203	3	591.17	5	lr203	3	949.40	5	lrc204	3	818.66	5
lc204	3	590.60	5	lr204	2	849.05	5	lrc205	4	1302.20	5
lc205	3	588.88	5	lr205	3	1054.02	5	lrc206	3	1159.03	5
lc206	3	588.49	5	lr206	3	931.63	5	lrc207	3	1062.05	5
lc207	3	588.29	5	lr207	2	903.06	5	lrc208	3	852.76	5
lc208	3	588.32	5	lr208	2	734.85	5				
lr101	19	1650.80	5	lr209	3	930.59	5				
lr102	17	1487.57	5	lr210	3	964.22	5				
lr103	13	1292.68	5	lr211	2	911.52	5				

Table A.2: Results of algorithm A1 for the Li and Lim instances with 200 locations

Instance	Veh.	Cost	Time (min.)	Instance	Veh.	Cost	Time (min.)	Instance	Veh.	Cost	Time (min.)
LC1_2_1	20	2704.57	15	LR1_2_1	20	4819.12	15	LRC1_2_1	19	3606.06	15
LC1_2_2	19	2764.56	15	LR1_2_2	17	4621.21	15	LRC1_2_2	15	3671.02	15
LC1_2_3	17	3133.75	15	LR1_2_3	14	4402.38	15	LRC1_2_3	13	3161.75	15
LC1_2_4	17	2693.41	15	LR1_2_4	10	3027.50	15	LRC1_2_4	10	2631.82	15
LC1_2_5	20	2702.05	15	LR1_2_5	16	4760.18	15	LRC1_2_5	16	3715.81	15
LC1_2_6	20	2701.04	15	LR1_2_6	13	4826.74	15	LRC1_2_6	16	3572.16	15
LC1_2_7	20	2701.04	15	LR1_2_7	12	3543.36	15	LRC1_2_7	14	3697.71	15
LC1_2_8	19	3404.95	15	LR1_2_8	9	2759.32	15	LRC1_2_8	13	3390.10	15
LC1_2_9	18	2724.24	15	LR1_2_9	14	4343.86	15	LRC1_2_9	13	3184.14	15
LC1_2_10	17	2942.13	15	LR1_2_10	11	3700.06	15	LRC1_2_10	12	2950.59	15
LC2_2_1	6	1931.44	15	LR2_2_1	5	4073.10	15	LRC2_2_1	6	3620.80	15
LC2_2_2	6	1881.40	15	LR2_2_2	4	3796.00	15	LRC2_2_2	5	3184.23	15
LC2_2_3	6	1844.33	15	LR2_2_3	4	3098.36	15	LRC2_2_3	4	3115.20	15
LC2_2_4	6	1767.12	15	LR2_2_4	3	2486.00	15	LRC2_2_4	3	2858.70	15
LC2_2_5	6	1891.21	15	LR2_2_5	4	3438.39	15	LRC2_2_5	5	2776.93	15
LC2_2_6	6	1857.78	15	LR2_2_6	3	4861.14	15	LRC2_2_6	5	2707.96	15
LC2_2_7	6	1850.13	15	LR2_2_7	3	3116.09	15	LRC2_2_7	4	3019.81	15
LC2_2_8	6	1824.34	15	LR2_2_8	2	2529.49	15	LRC2_2_8	4	2399.89	15
LC2_2_9	6	1854.21	15	LR2_2_9	3	4422.09	15	LRC2_2_9	4	2208.49	15
LC2_2_10	6	1817.45	15	LR2_2_10	3	3274.96	15	LRC2_2_10	3	2535.37	15

Table A.3: Results of algorithm A1 for the Li and Lim instances with 400 locations

Instance	Veh.	Cost	Time (min.)	Instance	Veh.	Cost	Time (min.)	Instance	Veh.	Cost	Time (min.)
LC1_4_1	40	7152.06	15	LR1_4_1	40	10639.75	15	LRC1_4_1	36	9124.52	15
LC1_4_2	38	8007.79	15	LR1_4_2	31	9975.40	15	LRC1_4_2	31	8346.06	15
LC1_4_3	32	8741.18	15	LR1_4_3	22	9473.37	15	LRC1_4_3	24	7821.28	15
LC1_4_4	30	6912.73	15	LR1_4_4	16	6916.62	15	LRC1_4_4	19	5803.31	15
LC1_4_5	40	7150.00	15	LR1_4_5	28	11937.96	15	LRC1_4_5	32	8919.46	15
LC1_4_6	40	7154.02	15	LR1_4_6	24	9823.12	15	LRC1_4_6	30	8545.23	15
LC1_4_7	40	7149.43	15	LR1_4_7	19	8472.73	15	LRC1_4_7	28	8252.91	15
LC1_4_8	39	7111.16	15	LR1_4_8	14	6196.66	15	LRC1_4_8	26	7958.76	15
LC1_4_9	36	7451.20	15	LR1_4_9	24	9884.27	15	LRC1_4_9	25	8296.01	15
LC1_4_10	35	7634.68	15	LR1_4_10	20	8474.78	15	LRC1_4_10	23	7348.49	15
LC2_4_1	12	4116.33	15	LR2_4_1	8	9726.88	15	LRC2_4_1	12	7454.14	15
LC2_4_2	12	4144.29	15	LR2_4_2	7	9946.91	15	LRC2_4_2	10	7539.93	15
LC2_4_3	12	4414.39	15	LR2_4_3	6	8393.55	15	LRC2_4_3	8	6570.81	15
LC2_4_4	12	4038.00	15	LR2_4_4	4	7297.57	15	LRC2_4_4	5	5309.88	15
LC2_4_5	12	4030.63	15	LR2_4_5	6	11246.73	15	LRC2_4_5	10	8549.23	15
LC2_4_6	12	3900.29	15	LR2_4_6	5	9379.17	15	LRC2_4_6	9	7600.84	15
LC2_4_7	12	3962.51	15	LR2_4_7	5	7568.45	15	LRC2_4_7	8	7453.36	15
LC2_4_8	12	3844.45	15	LR2_4_8	4	5579.67	15	LRC2_4_8	7	6762.81	15
LC2_4_9	12	4188.93	15	LR2_4_9	6	8017.25	15	LRC2_4_9	6	6884.11	15
LC2_4_10	12	3828.44	15	LR2_4_10	5	8343.99	15	LRC2_4_10	6	5669.56	15

Table A.4: Results of algorithm A1 for the Li and Lim instances with 600 locations

Instance	Veh.	Cost	Time (min.)	Instance	Veh.	Cost	Time (min.)	Instance	Veh.	Cost	Time (min.)
LC1_6_1	60	14095.64	30	LR1_6_1	59	22821.65	30	LRC1_6_1	52	18477.83	30
LC1_6_2	57	15048.16	30	LR1_6_2	45	20143.98	30	LRC1_6_2	43	16804.68	30
LC1_6_3	50	14688.89	30	LR1_6_3	37	17870.86	30	LRC1_6_3	36	13993.18	30
LC1_6_4	48	13330.50	30	LR1_6_4	28	13214.00	30	LRC1_6_4	25	10819.45	30
LC1_6_5	60	14086.30	30	LR1_6_5	38	22921.67	30	LRC1_6_5	46	16969.48	30
LC1_6_6	60	14090.79	30	LR1_6_6	32	21175.64	30	LRC1_6_6	42	17594.33	30
LC1_6_7	60	14083.76	30	LR1_6_7	25	17212.16	30	LRC1_6_7	38	16207.71	30
LC1_6_8	58	14880.70	30	LR1_6_8	18	12388.53	30	LRC1_6_8	33	15598.22	30
LC1_6_9	54	14600.28	30	LR1_6_9	31	21986.08	30	LRC1_6_9	34	15238.43	30
LC1_6_10	52	15823.95	30	LR1_6_10	26	19590.20	30	LRC1_6_10	30	14165.75	30
LC2_6_1	19	7977.98	30	LR2_6_1	11	21835.20	30	LRC2_6_1	16	16293.38	30
LC2_6_2	18	10325.34	30	LR2_6_2	9	24389.57	30	LRC2_6_2	13	17306.40	30
LC2_6_3	18	7436.50	30	LR2_6_3	7	20220.59	30	LRC2_6_3	10	12527.86	30
LC2_6_4	17	7926.49	30	LR2_6_4	6	12345.37	30	LRC2_6_4	7	10191.51	30
LC2_6_5	19	8047.37	30	LR2_6_5	9	22169.67	30	LRC2_6_5	13	14907.19	30
LC2_6_6	18	9480.48	30	LR2_6_6	7	21298.88	30	LRC2_6_6	12	16639.47	30
LC2_6_7	19	7997.96	30	LR2_6_7	6	16652.09	30	LRC2_6_7	10	16494.21	30
LC2_6_8	18	7579.93	30	LR2_6_8	5	12299.29	30	LRC2_6_8	9	15399.61	30
LC2_6_9	18	8959.66	30	LR2_6_9	8	21459.78	30	LRC2_6_9	9	15696.09	30
LC2_6_10	18	7479.38	30	LR2_6_10	7	18709.55	30	LRC2_6_10	8	14136.85	30

Table A.5: Results of algorithm A1 for the Li and Lim instances with 800 locations

Instance	Veh.	Cost	Time (min.)	Instance	Veh.	Cost	Time (min.)	Instance	Veh.	Cost	Time (min.)
LC1_8_1	80	25184.38	60	LR1_8_1	80	39291.32	60	LRC1_8_1	66	32299.70	60
LC1_8_2	77	26910.98	60	LR1_8_2	59	34123.36	60	LRC1_8_2	56	27940.22	60
LC1_8_3	64	26280.66	60	LR1_8_3	44	29624.24	60	LRC1_8_3	48	24450.06	60
LC1_8_4	60	23048.00	60	LR1_8_4	25	21239.28	60	LRC1_8_4	34	18240.98	60
LC1_8_5	80	25211.22	60	LR1_8_5	49	39526.16	60	LRC1_8_5	58	31512.41	60
LC1_8_6	80	25164.25	60	LR1_8_6	39	36778.66	60	LRC1_8_6	55	28633.52	60
LC1_8_7	80	25158.38	60	LR1_8_7	30	28256.44	60	LRC1_8_7	51	28354.98	60
LC1_8_8	78	25348.45	60	LR1_8_8	20	20754.01	60	LRC1_8_8	45	27110.23	60
LC1_8_9	73	25986.22	60	LR1_8_9	41	38009.90	60	LRC1_8_9	45	25144.11	60
LC1_8_10	70	26866.55	60	LR1_8_10	31	30676.35	60	LRC1_8_10	40	24494.47	60
LC2_8_1	24	11687.06	60	LR2_8_1	15	36029.50	60	LRC2_8_1	20	23119.30	60
LC2_8_2	24	13939.09	60	LR2_8_2	12	33295.11	60	LRC2_8_2	17	22620.74	60
LC2_8_3	25	12452.19	60	LR2_8_3	9	28683.95	60	LRC2_8_3	15	18149.95	60
LC2_8_4	24	12289.19	60	LR2_8_4	7	20413.49	60	LRC2_8_4	11	15353.86	60
LC2_8_5	25	12329.80	60	LR2_8_5	11	39241.12	60	LRC2_8_5	16	25200.13	60
LC2_8_6	24	12777.52	60	LR2_8_6	9	33075.07	60	LRC2_8_6	15	27484.83	60
LC2_8_7	25	11854.44	60	LR2_8_7	7	30453.51	60	LRC2_8_7	14	25742.09	60
LC2_8_8	24	11454.33	60	LR2_8_8	5	20062.84	60	LRC2_8_8	12	21756.80	60
LC2_8_9	24	11629.41	60	LR2_8_9	10	32524.30	60	LRC2_8_9	11	22672.77	60
LC2_8_10	24	11574.90	60	LR2_8_10	9	31439.75	60	LRC2_8_10	9	35328.46	60

Table A.6: Results of algorithm A1 for the Li and Lim instances with 1000 locations

Instance	Veh.	Cost	Time (min.)	Instance	Veh.	Cost	Time (min.)	Instance	Veh.	Cost	Time (min.)
LC1_10_1	100	42488.66	60	LR1_10_1	100	56744.91	60	LRC1_10_1	82	49231.54	60
LC1_10_2	94	44836.37	60	LR1_10_2	80	49377.63	60	LRC1_10_2	72	44992.82	60
LC1_10_3	82	42299.99	60	LR1_10_3	54	42719.26	60	LRC1_10_3	53	35855.32	60
LC1_10_4	74	38793.09	60	LR1_10_4	28	31455.61	60	LRC1_10_4	40	27530.32	60
LC1_10_5	100	42477.41	60	LR1_10_5	60	60748.39	60	LRC1_10_5	73	50122.41	60
LC1_10_6	101	42838.39	60	LR1_10_6	49	49771.43	60	LRC1_10_6	68	44575.32	60
LC1_10_7	100	42854.99	60	LR1_10_7	36	39832.31	60	LRC1_10_7	61	41371.73	60
LC1_10_8	98	42949.56	60	LR1_10_8	26	29767.75	60	LRC1_10_8	56	41351.90	60
LC1_10_9	91	44481.12	60	LR1_10_9	49	53369.24	60	LRC1_10_9	54	39876.82	60
LC1_10_10	88	42790.86	60	LR1_10_10	39	46468.72	60	LRC1_10_10	49	36544.29	60
LC2_10_1	30	16879.24	60	LR2_10_1	18	59306.65	60	LRC2_10_1	22	34890.32	60
LC2_10_2	31	19092.08	60	LR2_10_2	14	59836.95	60	LRC2_10_2	20	33450.55	60
LC2_10_3	30	17732.50	60	LR2_10_3	11	43760.22	60	LRC2_10_3	16	28988.73	60
LC2_10_4	29	18465.46	60	LR2_10_4	8	30227.24	60	LRC2_10_4	11	24579.99	60
LC2_10_5	31	17137.53	60	LR2_10_5	14	53837.14	60	LRC2_10_5	17	36298.78	60
LC2_10_6	31	17194.08	60	LR2_10_6	11	52235.00	60	LRC2_10_6	17	31071.38	60
LC2_10_7	31	19185.26	60	LR2_10_7	9	41765.65	60	LRC2_10_7	16	33723.56	60
LC2_10_8	30	17015.03	60	LR2_10_8	7	28847.48	60	LRC2_10_10	11	35761.59	60
LC2_10_9	31	17550.38	60	LR2_10_9	12	56798.61	60				
LC2_10_10	30	16581.15	60	LR2_10_10	11	48260.81	60				

APPENDIX B — DETAILED RESULTS FOR THE EXACT INSTANCES

Table B.1: Results of algorithm A1 (average) for the instances for exact methods.

Instance	Cost	Gap(%)	Time (min.)	Instance	Cost	Gap(%)	Time (min.)
AA30	31119.1	0.00	5	CC30	31087.8	0.00	5
AA35	31299.8	0.00	5	CC35	31230.6	0.00	5
AA40	31515.9	0.00	5	CC40	31359.1	0.00	5
AA45	31759.9	0.00	5	CC45	31509.1	0.00	5
AA50	41775.0	0.00	5	CC50	41690.4	0.01	5
AA55	41907.8	0.00	5	CC55	41841.3	0.01	5
AA60	42140.7	0.00	5	CC60	42017.8	0.02	5
AA65	42253.4	0.01	5	CC65	42179.9	0.04	5
AA70	42482.1	0.07	5	CC70	52221.6	0.04	5
AA75	52473.7	0.02	5	CC75	52382.8	0.05	5
BB30	31086.3	0.00	5	DD30	21133.3	0.00	5
BB35	31281.2	0.00	5	DD35	31216.4	0.02	5
BB40	31493.4	0.00	5	DD40	31354.0	0.01	5
BB45	41555.1	0.00	5	DD45	31484.0	0.00	5
BB50	41701.3	0.00	5	DD50	31601.3	0.00	5
BB55	41885.7	0.00	5	DD55	31746.7	0.01	5
BB60	62420.5	0.00	5	DD60	37952.9	18.35	5
BB65	62639.2	0.00	5	DD65	42127.3	0.05	5
BB70	62978.8	0.04	5	DD70	42226.3	0.03	5
BB75	63166.4	0.06	5	DD75	42381.0	0.05	5

APPENDIX C — DETAILED RESULTS FOR THE NEW INSTANCES

Table C.1: Results of algorithm A1 for the new proposed instances in sizes 100, 200, and 400

Instance	Veh.	Cost	Time (min.)	Instance	Veh.	Cost	Time (min.)	Instance	Veh.	Cost	Time (min.)
bar-n100-1	6	735	5	bar-n200-1	22	1839	15	bar-n400-1	32	3123	15
bar-n100-2	5	556	5	bar-n200-2	23	2078	15	bar-n400-2	30	2775	15
bar-n100-3	6	746	5	bar-n200-3	9	1564	15	bar-n400-3	11	2573	15
bar-n100-4	12	1160	5	bar-n200-4	13	841	15	bar-n400-4	18	1786	15
bar-n100-5	6	838	5	bar-n200-5	5	858	15	bar-n400-5	41	3402	15
bar-n100-6	3	788	5	bar-n200-6	9	863	15	bar-n400-6	21	2986	15
ber-n100-1	13	1857	5	bar-n200-7	12	1772	15	bar-n400-7	12	2987	15
ber-n100-2	6	1491	5	ber-n200-1	28	3189	15	ber-n400-1	34	5633	15
ber-n100-3	3	713	5	ber-n200-2	12	3292	15	ber-n400-2	35	5595	15
ber-n100-4	3	494	5	ber-n200-3	9	899	15	ber-n400-3	43	3552	15
ber-n100-5	5	944	5	ber-n200-4	5	1089	15	ber-n400-4	19	2217	15
ber-n100-6	14	2147	5	ber-n200-5	27	3944	15	ber-n400-5	27	5714	15
ber-n100-7	7	1935	5	ber-n200-6	10	3002	15	ber-n400-6	19	6358	15
nyc-n100-1	6	635	5	nyc-n200-1	7	943	15	ber-n400-7	20	6513	15
nyc-n100-2	4	567	5	nyc-n200-2	8	1104	15	nyc-n400-1	13	1977	15
nyc-n100-3	3	492	5	nyc-n200-3	7	1030	15	nyc-n400-2	14	1975	15
nyc-n100-4	2	556	5	nyc-n200-4	4	1042	15	nyc-n400-3	7	1861	15
nyc-n100-5	2	671	5	nyc-n200-5	5	1193	15	nyc-n400-4	7	1982	15
poa-n100-1	12	1590	5	poa-n200-1	25	2433	15	nyc-n400-5	7	1922	15
poa-n100-2	15	1539	5	poa-n200-2	13	2361	15	poa-n400-1	25	4550	15
poa-n100-3	10	1301	5	poa-n200-3	22	1851	15	poa-n400-2	42	3109	15
poa-n100-4	7	1668	5	poa-n200-4	10	1164	15	poa-n400-3	40	2844	15
poa-n100-5	6	624	5	poa-n200-5	15	2323	15	poa-n400-4	19	2187	15
poa-n100-6	3	562	5	poa-n200-6	28	3081	15	poa-n400-5	15	2303	15
poa-n100-7	5	779	5	poa-n200-7	11	2463	15	poa-n400-6	42	5428	15

Table C.2: Results of algorithm A1 for the new proposed instances in sizes 600, 800, and 1000

Instance	Veh.	Cost	Time (min.)	Instance	Veh.	Cost	Time (min.)	Instance	Veh.	Cost	Time (min.)
bar-n600-1	43	3746	30	bar-n800-1	80	5658	60	bar-n1000-1	54	7940	60
bar-n600-2	24	3947	30	bar-n800-2	31	5115	60	bar-n1000-2	38	3416	60
bar-n600-3	24	3931	30	bar-n800-3	23	5864	60	bar-n1000-3	89	4849	60
bar-n600-4	53	2880	30	bar-n800-4	25	2791	60	bar-n1000-4	19	3466	60
bar-n600-5	13	2631	30	bar-n800-5	81	6099	60	bar-n1000-5	27	6161	60
bar-n600-6	33	4952	30	bar-n800-6	82	6535	60	bar-n1000-6	28	6744	60
bar-n600-7	33	4888	30	bar-n800-7	31	5582	60	ber-n1000-1	89	15067	60
ber-n600-1	49	7699	30	ber-n800-1	59	5386	60	ber-n1000-2	118	16591	60
ber-n600-2	32	3882	30	ber-n800-2	63	6404	60	ber-n1000-3	53	13462	60
ber-n600-3	29	3987	30	ber-n800-3	18	3698	60	ber-n1000-4	56	14520	60
ber-n600-4	76	11120	30	ber-n800-4	108	16125	60	ber-n1000-5	111	15389	60
ber-n600-5	34	8567	30	ber-n800-5	35	10966	60	ber-n1000-6	155	18969	60
ber-n600-6	38	10371	30	ber-n800-6	49	13245	60	ber-n1000-7	74	17597	60
nyc-n600-1	21	3012	30	nyc-n800-1	23	3140	60	nyc-n1000-1	28	4101	60
nyc-n600-2	19	2713	30	nyc-n800-2	27	3744	60	nyc-n1000-2	33	4837	60
nyc-n600-3	20	2728	30	nyc-n800-3	28	3760	60	nyc-n1000-3	34	4558	60
nyc-n600-4	9	2526	30	nyc-n800-4	12	3194	60	nyc-n1000-4	18	4875	60
nyc-n600-5	11	2946	30	nyc-n800-5	14	3703	60	nyc-n1000-5	18	4280	60
poa-n600-1	55	6257	30	poa-n800-1	60	9570	60	poa-n1000-1	31	8277	60
poa-n600-2	27	5271	30	poa-n800-2	73	8087	60	poa-n1000-2	48	10702	60
poa-n600-3	24	2220	30	poa-n800-3	50	9846	60	poa-n1000-3	69	5570	60
poa-n600-4	27	3153	30	poa-n800-4	46	7991	60	poa-n1000-4	23	4598	60
poa-n600-5	20	2586	30	poa-n800-5	73	4269	60	poa-n1000-5	48	5822	60
poa-n600-6	77	8059	30	poa-n800-6	38	4130	60	poa-n1000-6	95	11514	60
poa-n600-7	63	7683	30	poa-n800-7	38	8005	60	poa-n1000-7	75	11661	60

Table C.3: Results of algorithm A1 for the new proposed instances in sizes 1500, 2000, and 2500

Instance	Veh.	Cost	Time (min.)	Instance	Veh.	Cost	Time (min.)	Instance	Veh.	Cost	Time (min.)
bar-n1500-1	77	9088	60	bar-n2000-1	99	11800	60	bar-n2500-1	81	10075	60
bar-n1500-2	63	11782	60	bar-n2000-2	99	11730	60	bar-n2500-2	123	15032	60
bar-n1500-3	95	6009	60	bar-n2000-3	146	13212	60	bar-n2500-3	66	15411	60
bar-n1500-4	62	5023	60	bar-n2000-4	75	11666	60	bar-n2500-4	67	15435	60
bar-n1500-5	76	9577	60	bar-n2000-5	77	12915	60	bar-n2500-5	131	23474	60
bar-n1500-6	161	12949	60	bar-n2000-6	176	9479	60	bar-n2500-6	102	17882	60
bar-n1500-7	41	10025	60	bar-n2000-7	69	9365	60	ber-n2500-1	207	33094	60
ber-n1500-1	174	23759	60	ber-n2000-1	76	13002	60	ber-n2500-2	142	37802	60
ber-n1500-2	69	8628	60	ber-n2000-2	280	32583	60	ber-n2500-3	248	18483	60
ber-n1500-3	71	9209	60	ber-n2000-3	167	26951	60	ber-n2500-4	181	16019	60
ber-n1500-4	37	8729	60	ber-n2000-4	252	35823	60	ber-n2500-5	262	21441	60
ber-n1500-5	181	25548	60	ber-n2000-5	143	32097	60	ber-n2500-6	306	44048	60
ber-n1500-6	99	21151	60	ber-n2000-6	115	29785	60	ber-n2500-7	177	40220	60
ber-n1500-7	101	21683	60	ber-n2000-7	139	30034	60	nyc-n2500-1	71	10264	60
nyc-n1500-1	47	6784	60	nyc-n2000-1	56	7750	60	nyc-n2500-2	79	9369	60
nyc-n1500-2	52	6535	60	nyc-n2000-2	59	7225	60	nyc-n2500-3	37	8983	60
nyc-n1500-3	47	6109	60	nyc-n2000-3	33	9153	60	nyc-n2500-4	46	12253	60
nyc-n1500-4	28	7279	60	nyc-n2000-4	29	7333	60	nyc-n2500-5	47	11308	60
nyc-n1500-5	23	5846	60	nyc-n2000-5	36	9166	60	poa-n2500-1	301	29790	60
poa-n1500-1	145	17392	60	poa-n2000-1	236	22850	60	poa-n2500-2	162	23972	60
poa-n1500-2	203	23869	60	poa-n2000-2	161	16678	60	poa-n2500-3	83	22680	60
poa-n1500-3	69	15099	60	poa-n2000-3	129	9451	60	poa-n2500-4	84	23413	60
poa-n1500-4	64	6591	60	poa-n2000-4	146	12605	60	poa-n2500-5	74	19303	60
poa-n1500-5	33	6237	60	poa-n2000-5	99	12835	60	poa-n2500-6	109	11324	60
poa-n1500-6	142	16893	60	poa-n2000-6	66	18959	60	poa-n2500-7	84	11234	60

Table C.4: Results of algorithm A1 for the new proposed instances in sizes 3000, 4000, and 5000

Instance	Veh.	Cost	Time (min.)	Instance	Veh.	Cost	Time (min.)	Instance	Veh.	Cost	Time (min.)
bar-n3000-1	160	22887	60	bar-n4000-1	158	26524	60	bar-n5000-1	236	23468	60
bar-n3000-2	151	21966	60	bar-n4000-2	108	22376	60	bar-n5000-2	96	15261	60
bar-n3000-3	59	10422	60	bar-n4000-3	104	24011	60	bar-n5000-3	266	38306	60
bar-n3000-4	261	27374	60	bar-n4000-4	160	14441	60	bar-n5000-4	601	50344	60
bar-n3000-5	164	21003	60	bar-n4000-5	158	13538	60	bar-n5000-5	421	49060	60
bar-n3000-6	86	19433	60	bar-n4000-6	157	26454	60	bar-n5000-6	271	39249	60
bar-n3000-7	81	19930	60	ber-n4000-1	577	73677	60	ber-n5000-1	767	103564	60
ber-n3000-1	303	38127	60	ber-n4000-2	416	29743	60	ber-n5000-2	412	71613	60
ber-n3000-2	224	34497	60	ber-n4000-3	139	21317	60	ber-n5000-3	185	63113	60
ber-n3000-3	197	38845	60	ber-n4000-4	181	18695	60	ber-n5000-4	324	77936	60
ber-n3000-4	240	22502	60	ber-n4000-5	149	23923	60	ber-n5000-5	498	31456	60
ber-n3000-5	137	17138	60	ber-n4000-6	324	56965	60	ber-n5000-6	167	21786	60
ber-n3000-6	100	14274	60	ber-n4000-7	156	49730	60	ber-n5000-7	420	74166	60
ber-n3000-7	471	59538	60	nyc-n4000-1	138	15468	60	nyc-n5000-1	131	17546	60
nyc-n3000-1	80	10596	60	nyc-n4000-2	113	14704	60	nyc-n5000-2	149	19879	60
nyc-n3000-2	84	10828	60	nyc-n4000-3	127	15264	60	nyc-n5000-3	74	24742	60
nyc-n3000-3	46	12075	60	nyc-n4000-4	64	17435	60	nyc-n5000-4	88	20945	60
nyc-n3000-4	53	12966	60	nyc-n4000-5	71	16335	60	nyc-n5000-5	81	18101	60
nyc-n3000-5	45	11638	60	poa-n4000-1	517	60472	60	poa-n5000-1	297	65602	60
poa-n3000-1	388	40771	60	poa-n4000-2	513	59115	60	poa-n5000-2	152	44567	60
poa-n3000-2	172	28167	60	poa-n4000-3	196	23772	60	poa-n5000-3	289	62040	60
poa-n3000-3	293	20039	60	poa-n4000-4	379	49664	60	poa-n5000-4	225	28593	60
poa-n3000-4	153	22207	60	poa-n4000-5	404	54647	60	poa-n5000-5	372	38020	60
poa-n3000-5	198	28035	60	poa-n4000-6	543	65615	60	poa-n5000-6	115	24700	60
poa-n3000-6	209	33708	60	poa-n4000-7	139	40480	60	poa-n5000-7	227	45868	60

APPENDIX D — CHARACTERISTICS OF THE NEW INSTANCES

Table D.1: Detailed characteristics of the new instances with 100 locations

Name	Size	City	Distribution	Clusters	Density	Horizon	Time Windows	Service Time	Capacities	Depot
bar-n100-1	100	Barcelona	random-cluster	7	1.5	240	120	5	300	random
bar-n100-2	100	Barcelona	cluster	7	1.2	480	120	15	100	random
bar-n100-3	100	Barcelona	random	-	-	240	120	5	300	central
bar-n100-4	100	Barcelona	random	-	-	240	60	15	300	random
bar-n100-5	100	Barcelona	random	-	-	480	120	15	100	random
bar-n100-6	100	Barcelona	random	-	-	480	120	5	300	random
ber-n100-1	100	Berlin	random-cluster	3	0.8	240	60	10	300	random
ber-n100-2	100	Berlin	random-cluster	4	1.1	480	120	10	100	random
ber-n100-3	100	Berlin	cluster	7	1.0	480	120	5	100	central
ber-n100-4	100	Berlin	cluster	3	1.6	480	120	5	300	random
ber-n100-5	100	Berlin	cluster	3	0.8	480	60	10	100	random
ber-n100-6	100	Berlin	random	-	-	240	60	10	300	central
ber-n100-7	100	Berlin	random	-	-	480	60	5	100	central
nyc-n100-1	100	New-York	random	-	-	240	30	2	6	central
nyc-n100-2	100	New-York	random	-	-	240	60	2	6	central
nyc-n100-3	100	New-York	random	-	-	240	60	2	6	random
nyc-n100-4	100	New-York	random	-	-	480	120	2	6	central
nyc-n100-5	100	New-York	random	-	-	480	60	2	6	random
poa-n100-1	100	Porto-Alegre	random-cluster	6	1.6	240	60	10	100	central
poa-n100-2	100	Porto-Alegre	random-cluster	4	1.3	240	60	15	300	central
poa-n100-3	100	Porto-Alegre	random-cluster	7	1.5	240	60	10	300	random
poa-n100-4	100	Porto-Alegre	random-cluster	8	0.9	480	60	15	100	random
poa-n100-5	100	Porto-Alegre	cluster	5	1.0	240	60	5	300	central
poa-n100-6	100	Porto-Alegre	cluster	3	1.1	480	120	5	100	central
poa-n100-7	100	Porto-Alegre	cluster	6	1.5	480	60	15	100	central

Table D.2: Detailed characteristics of the new instances with 200 locations

Name	Size	City	Distribution	Clusters	Density	Horizon	Time Windows	Service Time	Capacities	Depot
bar-n200-1	200	Barcelona	random-cluster	7	1.0	240	60	15	100	random
bar-n200-2	200	Barcelona	random-cluster	7	1.5	240	60	15	300	random
bar-n200-3	200	Barcelona	random-cluster	3	1.5	480	60	10	100	random
bar-n200-4	200	Barcelona	cluster	4	0.9	240	120	10	300	central
bar-n200-5	200	Barcelona	cluster	3	0.9	480	120	5	100	central
bar-n200-6	200	Barcelona	cluster	3	1.3	480	120	15	100	random
bar-n200-7	200	Barcelona	random	-	-	480	60	15	300	central
ber-n200-1	200	Berlin	random-cluster	6	1.5	240	120	15	300	central
ber-n200-2	200	Berlin	random-cluster	8	1.2	480	60	10	300	central
ber-n200-3	200	Berlin	cluster	6	1.2	240	120	5	300	central
ber-n200-4	200	Berlin	cluster	5	1.5	480	120	5	300	central
ber-n200-5	200	Berlin	random	-	-	240	60	10	300	central
ber-n200-6	200	Berlin	random	-	-	480	120	5	100	central
nyc-n200-1	200	New-York	random	-	-	240	120	2	6	random
nyc-n200-2	200	New-York	random	-	-	240	30	2	6	central
nyc-n200-3	200	New-York	random	-	-	240	60	2	6	random
nyc-n200-4	200	New-York	random	-	-	480	120	2	6	random
nyc-n200-5	200	New-York	random	-	-	480	30	2	6	central
poa-n200-1	200	Porto-Alegre	random-cluster	3	0.8	240	60	15	300	random
poa-n200-2	200	Porto-Alegre	random-cluster	4	1.1	480	120	15	100	random
poa-n200-3	200	Porto-Alegre	cluster	6	1.1	240	120	15	100	random
poa-n200-4	200	Porto-Alegre	cluster	7	1.0	240	60	5	300	random
poa-n200-5	200	Porto-Alegre	random	-	-	240	120	5	300	central
poa-n200-6	200	Porto-Alegre	random	-	-	240	60	15	100	random
poa-n200-7	200	Porto-Alegre	random	-	-	480	60	10	100	random

Table D.3: Detailed characteristics of the new instances with 400 locations

Name	Size	City	Distribution	Clusters	Density	Horizon	Time Windows	Service Time	Capacities	Depot
bar-n400-1	400	Barcelona	random-cluster	5	0.7	240	120	10	100	random
bar-n400-2	400	Barcelona	random-cluster	3	1.0	240	120	10	300	random
bar-n400-3	400	Barcelona	random-cluster	8	0.9	480	60	5	100	central
bar-n400-4	400	Barcelona	cluster	3	1.0	240	60	5	100	random
bar-n400-5	400	Barcelona	random	-	-	240	120	15	100	random
bar-n400-6	400	Barcelona	random	-	-	480	120	15	300	central
bar-n400-7	400	Barcelona	random	-	-	480	60	5	300	central
ber-n400-1	400	Berlin	random-cluster	3	1.0	240	120	5	100	random
ber-n400-2	400	Berlin	random-cluster	5	1.0	240	120	5	300	random
ber-n400-3	400	Berlin	cluster	4	1.6	240	60	15	100	random
ber-n400-4	400	Berlin	cluster	3	1.1	480	60	15	100	central
ber-n400-5	400	Berlin	random	-	-	480	120	15	300	central
ber-n400-6	400	Berlin	random	-	-	480	60	5	100	central
ber-n400-7	400	Berlin	random	-	-	480	60	5	300	central
nyc-n400-1	400	New-York	random	-	-	240	30	2	6	central
nyc-n400-2	400	New-York	random	-	-	240	30	2	6	random
nyc-n400-3	400	New-York	random	-	-	480	120	2	6	central
nyc-n400-4	400	New-York	random	-	-	480	60	2	6	central
nyc-n400-5	400	New-York	random	-	-	480	60	2	6	random
poa-n400-1	400	Porto-Alegre	random-cluster	4	1.0	480	120	15	300	central
poa-n400-2	400	Porto-Alegre	cluster	3	0.7	240	120	15	100	random
poa-n400-3	400	Porto-Alegre	cluster	5	1.3	240	120	15	300	random
poa-n400-4	400	Porto-Alegre	cluster	7	1.2	480	60	15	100	central
poa-n400-5	400	Porto-Alegre	cluster	4	1.1	480	60	10	300	central
poa-n400-6	400	Porto-Alegre	random	-	-	240	120	10	300	random

Table D.4: Detailed characteristics of the new instances with 600 locations

Name	Size	City	Distribution	Clusters	Density	Horizon	Time Windows	Service Time	Capacities	Depot
bar-n600-1	600	Barcelona	random-cluster	5	1.1	240	120	10	300	central
bar-n600-2	600	Barcelona	random-cluster	8	0.9	480	120	10	100	central
bar-n600-3	600	Barcelona	random-cluster	4	1.4	480	120	10	300	central
bar-n600-4	600	Barcelona	cluster	8	1.3	240	60	15	300	random
bar-n600-5	600	Barcelona	cluster	4	1.2	480	60	5	300	central
bar-n600-6	600	Barcelona	random	-	-	480	60	15	100	random
bar-n600-7	600	Barcelona	random	-	-	480	60	15	300	central
ber-n600-1	600	Berlin	random-cluster	7	0.8	240	120	5	100	random
ber-n600-2	600	Berlin	cluster	3	1.4	240	60	5	100	random
ber-n600-3	600	Berlin	cluster	4	1.4	480	60	15	100	random
ber-n600-4	600	Berlin	random	-	-	240	60	10	300	random
ber-n600-5	600	Berlin	random	-	-	480	120	10	100	central
ber-n600-6	600	Berlin	random	-	-	480	60	10	100	random
nyc-n600-1	600	New-York	random	-	-	240	30	2	6	central
nyc-n600-2	600	New-York	random	-	-	240	30	2	6	random
nyc-n600-3	600	New-York	random	-	-	240	60	2	6	random
nyc-n600-4	600	New-York	random	-	-	480	120	2	6	central
nyc-n600-5	600	New-York	random	-	-	480	60	2	6	random
poa-n600-1	600	Porto-Alegre	random-cluster	6	0.8	240	120	10	300	central
poa-n600-2	600	Porto-Alegre	random-cluster	4	0.9	480	120	10	100	central
poa-n600-3	600	Porto-Alegre	cluster	4	1.5	240	120	5	300	central
poa-n600-4	600	Porto-Alegre	cluster	7	0.9	480	120	15	100	central
poa-n600-5	600	Porto-Alegre	cluster	6	1.2	480	60	10	300	central
poa-n600-6	600	Porto-Alegre	random	-	-	240	120	15	100	random
poa-n600-7	600	Porto-Alegre	random	-	-	240	60	10	100	random

Table D.5: Detailed characteristics of the new instances with 800 locations

Name	Size	City	Distribution	Clusters	Density	Horizon	Time Windows	Service Time	Capacities	Depot
bar-n800-1	800	Barcelona	random-cluster	3	1.4	240	60	15	300	central
bar-n800-2	800	Barcelona	random-cluster	6	0.7	480	120	10	100	central
bar-n800-3	800	Barcelona	random-cluster	3	0.8	480	60	5	100	random
bar-n800-4	800	Barcelona	cluster	4	1.6	480	120	10	100	central
bar-n800-5	800	Barcelona	random	-	-	240	120	15	100	central
bar-n800-6	800	Barcelona	random	-	-	240	120	15	300	random
bar-n800-7	800	Barcelona	random	-	-	480	120	10	100	random
ber-n800-1	800	Berlin	cluster	4	1.5	240	120	10	100	central
ber-n800-2	800	Berlin	cluster	5	0.7	240	120	10	300	random
ber-n800-3	800	Berlin	cluster	8	1.2	480	120	5	100	random
ber-n800-4	800	Berlin	random	-	-	240	60	10	300	random
ber-n800-5	800	Berlin	random	-	-	480	120	5	100	central
ber-n800-6	800	Berlin	random	-	-	480	60	10	100	random
nyc-n800-1	800	New-York	random	-	-	240	120	2	6	random
nyc-n800-2	800	New-York	random	-	-	240	30	2	6	central
nyc-n800-3	800	New-York	random	-	-	240	30	2	6	random
nyc-n800-4	800	New-York	random	-	-	480	120	2	6	central
nyc-n800-5	800	New-York	random	-	-	480	60	2	6	random
poa-n800-1	800	Porto-Alegre	random-cluster	7	1.5	240	60	5	300	central
poa-n800-2	800	Porto-Alegre	random-cluster	5	1.0	240	60	10	300	random
poa-n800-3	800	Porto-Alegre	random-cluster	3	1.5	480	60	15	100	central
poa-n800-4	800	Porto-Alegre	random-cluster	8	1.4	480	60	15	300	random
poa-n800-5	800	Porto-Alegre	cluster	7	1.2	240	120	15	300	central
poa-n800-6	800	Porto-Alegre	cluster	5	0.9	240	60	5	300	random
poa-n800-7	800	Porto-Alegre	random	-	-	480	120	10	100	random

Table D.6: Detailed characteristics of the new instances with 1000 locations

Name	Size	City	Distribution	Clusters	Density	Horizon	Time Windows	Service Time	Capacities	Depot
bar-n1000-1	1000	Barcelona	random-cluster	7	1.3	480	60	15	300	random
bar-n1000-2	1000	Barcelona	cluster	4	1.3	240	120	5	100	central
bar-n1000-3	1000	Barcelona	cluster	7	1.1	240	60	15	100	central
bar-n1000-4	1000	Barcelona	cluster	5	1.0	480	120	5	300	random
bar-n1000-5	1000	Barcelona	random	-	-	480	120	5	100	central
bar-n1000-6	1000	Barcelona	random	-	-	480	60	5	300	random
ber-n1000-1	1000	Berlin	random-cluster	3	0.9	240	120	5	100	central
ber-n1000-2	1000	Berlin	random-cluster	5	0.8	240	60	10	300	random
ber-n1000-3	1000	Berlin	random-cluster	6	1.0	480	120	10	300	random
ber-n1000-4	1000	Berlin	random-cluster	5	1.3	480	60	10	300	random
ber-n1000-5	1000	Berlin	random	-	-	240	120	10	300	random
ber-n1000-6	1000	Berlin	random	-	-	240	60	15	100	central
ber-n1000-7	1000	Berlin	random	-	-	480	60	15	300	random
nyc-n1000-1	1000	New-York	random	-	-	240	120	2	6	central
nyc-n1000-2	1000	New-York	random	-	-	240	30	2	6	central
nyc-n1000-3	1000	New-York	random	-	-	240	30	2	6	random
nyc-n1000-4	1000	New-York	random	-	-	480	30	2	6	random
nyc-n1000-5	1000	New-York	random	-	-	480	60	2	6	central
poa-n1000-1	1000	Porto-Alegre	random-cluster	7	1.3	480	120	5	300	random
poa-n1000-2	1000	Porto-Alegre	random-cluster	7	1.5	480	60	10	100	central
poa-n1000-3	1000	Porto-Alegre	cluster	4	0.6	240	120	10	100	central
poa-n1000-4	1000	Porto-Alegre	cluster	5	1.1	480	120	5	100	central
poa-n1000-5	1000	Porto-Alegre	cluster	5	0.9	480	60	15	300	random
poa-n1000-6	1000	Porto-Alegre	random	-	-	240	120	10	100	central
poa-n1000-7	1000	Porto-Alegre	random	-	-	240	60	5	100	central

Table D.7: Detailed characteristics of the new instances with 1500 locations

Name	Size	City	Distribution	Clusters	Density	Horizon	Time Windows	Service Time	Capacities	Depot
bar-n1500-1	1500	Barcelona	random-cluster	6	1.2	240	60	5	300	random
bar-n1500-2	1500	Barcelona	random-cluster	3	1.1	480	60	10	100	random
bar-n1500-3	1500	Barcelona	cluster	5	1.2	240	60	10	300	central
bar-n1500-4	1500	Barcelona	cluster	3	1.2	480	120	15	300	central
bar-n1500-5	1500	Barcelona	random	-	-	240	120	5	100	random
bar-n1500-6	1500	Barcelona	random	-	-	240	60	15	300	central
bar-n1500-7	1500	Barcelona	random	-	-	480	60	5	300	central
ber-n1500-1	1500	Berlin	random-cluster	5	0.9	240	120	10	100	random
ber-n1500-2	1500	Berlin	cluster	7	1.6	480	120	15	100	random
ber-n1500-3	1500	Berlin	cluster	3	1.5	480	60	15	100	random
ber-n1500-4	1500	Berlin	cluster	7	0.7	480	60	5	300	central
ber-n1500-5	1500	Berlin	random	-	-	240	60	10	300	central
ber-n1500-6	1500	Berlin	random	-	-	480	120	15	100	central
ber-n1500-7	1500	Berlin	random	-	-	480	120	15	300	central
nyc-n1500-1	1500	New-York	random	-	-	240	120	2	6	central
nyc-n1500-2	1500	New-York	random	-	-	240	30	2	6	central
nyc-n1500-3	1500	New-York	random	-	-	240	60	2	6	central
nyc-n1500-4	1500	New-York	random	-	-	480	30	2	6	random
nyc-n1500-5	1500	New-York	random	-	-	480	60	2	6	random
poa-n1500-1	1500	Porto-Alegre	random-cluster	4	0.8	240	120	10	100	random
poa-n1500-2	1500	Porto-Alegre	random-cluster	3	0.6	240	60	15	100	random
poa-n1500-3	1500	Porto-Alegre	random-cluster	5	1.1	480	120	10	100	random
poa-n1500-4	1500	Porto-Alegre	cluster	5	1.0	240	60	5	300	random
poa-n1500-5	1500	Porto-Alegre	cluster	6	0.7	480	60	5	300	central
poa-n1500-6	1500	Porto-Alegre	random	-	-	240	120	10	300	central

Table D.8: Detailed characteristics of the new instances with 2000 locations

Name	Size	City	Distribution	Clusters	Density	Horizon	Time Windows	Service Time	Capacities	Depot
bar-n2000-1	2000	Barcelona	random-cluster	7	1.2	240	120	5	100	central
bar-n2000-2	2000	Barcelona	random-cluster	5	1.3	240	120	5	100	random
bar-n2000-3	2000	Barcelona	random-cluster	8	1.6	240	120	10	300	central
bar-n2000-4	2000	Barcelona	random-cluster	5	1.4	480	120	10	300	central
bar-n2000-5	2000	Barcelona	random-cluster	4	1.6	480	60	10	300	central
bar-n2000-6	2000	Barcelona	cluster	6	1.4	240	60	15	300	central
bar-n2000-7	2000	Barcelona	cluster	7	0.7	480	60	10	300	random
ber-n2000-1	2000	Berlin	cluster	3	0.8	480	60	10	300	random
ber-n2000-2	2000	Berlin	random	-	-	240	120	15	100	central
ber-n2000-3	2000	Berlin	random	-	-	240	120	5	100	random
ber-n2000-4	2000	Berlin	random	-	-	240	60	10	100	random
ber-n2000-5	2000	Berlin	random	-	-	480	120	15	100	random
ber-n2000-6	2000	Berlin	random	-	-	480	60	10	100	random
ber-n2000-7	2000	Berlin	random	-	-	480	60	15	300	central
nyc-n2000-1	2000	New-York	random	-	-	240	60	2	6	central
nyc-n2000-2	2000	New-York	random	-	-	240	60	2	6	random
nyc-n2000-3	2000	New-York	random	-	-	480	120	2	6	central
nyc-n2000-4	2000	New-York	random	-	-	480	120	2	6	random
nyc-n2000-5	2000	New-York	random	-	-	480	30	2	6	central
poa-n2000-1	2000	Porto-Alegre	random-cluster	6	1.1	240	120	15	100	central
poa-n2000-2	2000	Porto-Alegre	cluster	7	1.0	240	120	10	100	random
poa-n2000-3	2000	Porto-Alegre	cluster	7	1.2	240	120	10	300	central
poa-n2000-4	2000	Porto-Alegre	cluster	3	1.1	240	60	10	100	random
poa-n2000-5	2000	Porto-Alegre	cluster	7	0.6	480	60	15	300	central
poa-n2000-6	2000	Porto-Alegre	random	-	-	480	60	5	300	random

Table D.9: Detailed characteristics of the new instances with 2500 locations

Name	Size	City	Distribution	Clusters	Density	Horizon	Time Windows	Service Time	Capacities	Depot
bar-n2500-1	2500	Barcelona	cluster	8	1.0	480	60	10	100	central
bar-n2500-2	2500	Barcelona	random	-	-	240	120	5	300	random
bar-n2500-3	2500	Barcelona	random	-	-	480	120	5	100	central
bar-n2500-4	2500	Barcelona	random	-	-	480	120	5	100	random
bar-n2500-5	2500	Barcelona	random	-	-	480	60	15	100	central
bar-n2500-6	2500	Barcelona	random	-	-	480	60	10	100	random
ber-n2500-1	2500	Berlin	random-cluster	3	0.9	240	60	5	300	random
ber-n2500-2	2500	Berlin	random-cluster	3	0.8	480	60	10	100	central
ber-n2500-3	2500	Berlin	cluster	7	0.9	240	120	15	300	central
ber-n2500-4	2500	Berlin	cluster	7	1.3	240	120	10	300	random
ber-n2500-5	2500	Berlin	cluster	3	1.4	240	60	15	300	central
ber-n2500-6	2500	Berlin	random	-	-	240	120	10	300	random
ber-n2500-7	2500	Berlin	random	-	-	480	60	15	300	random
nyc-n2500-1	2500	New-York	random	-	-	240	60	2	6	central
nyc-n2500-2	2500	New-York	random	-	-	240	60	2	6	random
nyc-n2500-3	2500	New-York	random	-	-	480	120	2	6	random
nyc-n2500-4	2500	New-York	random	-	-	480	30	2	6	central
nyc-n2500-5	2500	New-York	random	-	-	480	30	2	6	random
poa-n2500-1	2500	Porto-Alegre	random-cluster	8	0.9	240	120	15	100	central
poa-n2500-2	2500	Porto-Alegre	random-cluster	4	1.4	240	120	5	300	random
poa-n2500-3	2500	Porto-Alegre	random-cluster	5	1.2	480	120	5	100	random
poa-n2500-4	2500	Porto-Alegre	random-cluster	8	1.3	480	60	5	100	central
poa-n2500-5	2500	Porto-Alegre	random-cluster	3	0.8	480	60	5	100	random
poa-n2500-6	2500	Porto-Alegre	cluster	4	1.5	240	60	5	300	random
poa-n2500-7	2500	Porto-Alegre	cluster	6	1.4	480	60	10	100	central

Table D.10: Detailed characteristics of the new instances with 3000 locations

Name	Size	City	Distribution	Clusters	Density	Horizon	Time Windows	Service Time	Capacities	Depot
bar-n3000-1	3000	Barcelona	random-cluster	3	0.9	480	120	15	100	random
bar-n3000-2	3000	Barcelona	random-cluster	6	1.5	480	60	15	300	central
bar-n3000-3	3000	Barcelona	cluster	8	1.1	480	120	5	300	random
bar-n3000-4	3000	Barcelona	random	-	-	240	60	10	100	random
bar-n3000-5	3000	Barcelona	random	-	-	240	60	5	300	central
bar-n3000-6	3000	Barcelona	random	-	-	480	60	5	100	random
bar-n3000-7	3000	Barcelona	random	-	-	480	60	5	300	random
ber-n3000-1	3000	Berlin	random-cluster	8	1.3	240	120	10	100	central
ber-n3000-2	3000	Berlin	random-cluster	4	1.0	240	120	5	300	central
ber-n3000-3	3000	Berlin	random-cluster	4	1.5	480	120	15	300	central
ber-n3000-4	3000	Berlin	cluster	8	0.7	240	60	10	300	central
ber-n3000-5	3000	Berlin	cluster	6	1.6	480	120	15	100	random
ber-n3000-6	3000	Berlin	cluster	6	1.4	480	60	10	300	central
ber-n3000-7	3000	Berlin	random	-	-	240	60	15	100	central
nyc-n3000-1	3000	New-York	random	-	-	240	120	2	6	central
nyc-n3000-2	3000	New-York	random	-	-	240	60	2	6	random
nyc-n3000-3	3000	New-York	random	-	-	480	120	2	6	random
nyc-n3000-4	3000	New-York	random	-	-	480	30	2	6	central
nyc-n3000-5	3000	New-York	random	-	-	480	60	2	6	central
poa-n3000-1	3000	Porto-Alegre	random-cluster	4	1.4	240	60	15	100	central
poa-n3000-2	3000	Porto-Alegre	random-cluster	5	1.3	480	120	15	100	random
poa-n3000-3	3000	Porto-Alegre	cluster	4	1.1	240	120	15	100	central
poa-n3000-4	3000	Porto-Alegre	cluster	3	0.6	480	60	15	100	random
poa-n3000-5	3000	Porto-Alegre	random	-	-	240	120	5	100	random
poa-n3000-6	3000	Porto-Alegre	random	-	-	240	60	5	300	central

Table D.11: Detailed characteristics of the new instances with 4000 locations

Name	Size	City	Distribution	Clusters	Density	Horizon	Time Windows	Service Time	Capacities	Depot
bar-n4000-1	4000	Barcelona	random-cluster	3	0.7	480	120	10	100	random
bar-n4000-2	4000	Barcelona	random-cluster	7	1.5	480	120	5	300	central
bar-n4000-3	4000	Barcelona	random-cluster	3	1.0	480	60	5	100	central
bar-n4000-4	4000	Barcelona	cluster	3	1.4	240	60	5	100	central
bar-n4000-5	4000	Barcelona	cluster	8	1.0	240	60	5	300	central
bar-n4000-6	4000	Barcelona	random	-	-	480	120	10	100	central
ber-n4000-1	4000	Berlin	random-cluster	6	1.0	240	60	15	300	central
ber-n4000-2	4000	Berlin	cluster	3	1.6	240	60	15	300	random
ber-n4000-3	4000	Berlin	cluster	3	1.2	480	120	10	100	random
ber-n4000-4	4000	Berlin	cluster	3	1.3	480	120	15	300	central
ber-n4000-5	4000	Berlin	cluster	6	0.8	480	60	10	100	central
ber-n4000-6	4000	Berlin	random	-	-	240	120	5	100	central
ber-n4000-7	4000	Berlin	random	-	-	480	120	5	300	random
nyc-n4000-1	4000	New-York	random	-	-	240	30	2	6	central
nyc-n4000-2	4000	New-York	random	-	-	240	60	2	6	central
nyc-n4000-3	4000	New-York	random	-	-	240	60	2	6	random
nyc-n4000-4	4000	New-York	random	-	-	480	120	2	6	central
nyc-n4000-5	4000	New-York	random	-	-	480	30	2	6	random
poa-n4000-1	4000	Porto-Alegre	random-cluster	5	1.5	240	60	15	100	central
poa-n4000-2	4000	Porto-Alegre	random-cluster	5	1.5	240	60	15	300	random
poa-n4000-3	4000	Porto-Alegre	cluster	5	0.9	480	60	15	300	random
poa-n4000-4	4000	Porto-Alegre	random	-	-	240	120	10	300	random
poa-n4000-5	4000	Porto-Alegre	random	-	-	240	60	10	100	random
poa-n4000-6	4000	Porto-Alegre	random	-	-	240	60	15	300	central
poa-n4000-7	4000	Porto-Alegre	random	-	-	480	120	5	100	random

Table D.12: Detailed characteristics of the new instances with 5000 locations

Name	Size	City	Distribution	Clusters	Density	Horizon	Time Windows	Service Time	Capacities	Depot
bar-n5000-1	5000	Barcelona	cluster	3	0.8	240	60	5	100	random
bar-n5000-2	5000	Barcelona	cluster	5	1.6	480	120	5	100	random
bar-n5000-3	5000	Barcelona	random	-	-	240	120	5	100	central
bar-n5000-4	5000	Barcelona	random	-	-	240	60	15	100	random
bar-n5000-5	5000	Barcelona	random	-	-	240	60	10	300	central
bar-n5000-6	5000	Barcelona	random	-	-	240	60	5	300	random
ber-n5000-1	5000	Berlin	random-cluster	3	1.2	240	120	15	100	central
ber-n5000-2	5000	Berlin	random-cluster	5	1.2	240	60	5	100	random
ber-n5000-3	5000	Berlin	random-cluster	8	0.6	480	120	5	100	random
ber-n5000-4	5000	Berlin	random-cluster	3	0.6	480	120	15	300	central
ber-n5000-5	5000	Berlin	cluster	6	1.6	240	60	15	300	random
ber-n5000-6	5000	Berlin	cluster	5	1.3	480	120	10	300	central
ber-n5000-7	5000	Berlin	random	-	-	240	60	5	300	central
nyc-n5000-1	5000	New-York	random	-	-	240	120	2	6	random
nyc-n5000-2	5000	New-York	random	-	-	240	60	2	6	central
nyc-n5000-3	5000	New-York	random	-	-	480	120	2	6	random
nyc-n5000-4	5000	New-York	random	-	-	480	30	2	6	random
nyc-n5000-5	5000	New-York	random	-	-	480	60	2	6	central
poa-n5000-1	5000	Porto-Alegre	random-cluster	7	1.4	480	120	15	100	central
poa-n5000-2	5000	Porto-Alegre	random-cluster	7	1.2	480	120	5	100	random
poa-n5000-3	5000	Porto-Alegre	random-cluster	3	1.4	480	120	15	300	central
poa-n5000-4	5000	Porto-Alegre	cluster	7	1.3	240	60	5	100	random
poa-n5000-5	5000	Porto-Alegre	cluster	8	1.1	240	60	10	300	central
poa-n5000-6	5000	Porto-Alegre	cluster	7	1.3	480	60	5	300	random
poa-n5000-7	5000	Porto-Alegre	random	-	-	480	120	10	300	central