UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

CARLO SULZBACH SARTORI

Optimizing Solutions for the Pickup and Delivery Problem

Work presented in partial fulfillment
of the requirements for the degree of
Bachelor in Computer Science

Advisor: Prof. Dr. Luciana Salete Buriol
Coadvisor: MSc. Marcelo Wuttig Friske

Porto Alegre
December 2016

ABSTRACT

Pickup and Delivery Problems are a variation of Vehicle Routing Problems that arise in many real-world transportation scenarios, such as product delivery and courier services. This work studies the Pickup and Delivery Problem with Time Windows, in which goods have to be transported from one location to another, respecting certain time restrictions and the capacity of the vehicles. It aims at minimizing the number of vehicles used, as well as the operational costs to perform all routes. To solve this problem, a mathematical formulation is used and an algorithm is proposed by embedding a Variable Neighborhood Descent method into an Iterated Local Search metaheuristic. Experiments are carried out with standard literature instances, showing that the algorithm is able to deliver good solutions in reasonable time up to a certain number of locations and vehicles. A real-world case study is conducted together with a partner company, which provided the data to generate a set of new instances based on the case. Experiments are done to evaluate how well the proposed algorithm can handle the different scenario. Results show the proposed algorithm produces good solutions for most instances of the real-world case.

Keywords: Vehicle routing problem. pickup and delivery. optimization. heuristic. iterated local search.

Otimizando Soluções para o Problema de Coleta e Entrega de Produtos

RESUMO

Problemas de Coleta e Entrega de Produtos são uma variação dos Problemas de Roteamento de Veículos, os quais têm um amplo número de aplicações reais em transportes, como entrega de produtos e serviços de correios. Este trabalho estuda o Problema de Coleta e Entrega de Produtos com Janelas de Tempo, no qual produtos devem ser transportados de um local para o outro, respeitando certas restrições de tempo e a capacidade de cada veículo. O objetivo é minimizar o número de veículos usados, assim como o custo total de operação das rotas. Para resolver este problema, uma formulação matemática é utilizada e um algoritmo é proposto através da implementação de um método de Descida com Variação de Vizinhança dentro de uma Busca Local Iterada. Experimentos foram realizados com instâncias padrões da literatura, demonstrando que o algoritmo é capaz de encontrar boas soluções em tempo razoável, para até um número limite de locais e veículos. Um estudo sbore um caso real é feito juntamente com uma empresa parceira, a qual forneceu dados para a geração de um novo conjunto de instâncias baseado nesse caso de estudo. Experimentos foram realizados para avaliar como o algoritmo proposto se comportaria neste cenário diferente. Os resultados demonstram que o método produz boas soluções para grande parte das instâncias dos casos reais.

Palavras-chave: problema de roteamento de veículos. coleta e entrega. otimização. heurística. busca local iterada.

LIST OF FIGURES

Figure 2.1 Example of PDPTW solutions. (a) Locations with demands and time windows [min, max], square node is the depot and circles are requests' locations; (b) and (c) possible solutions, with node labels being the exact time a vehicle reaches the node, and arc labels being the load being carried; (c), the dark node is responsible for the infeasibility. 16

Figure 3.1 Pictorial representation of iterated local search. 24

Figure 4.1 Inter-route neighborhood movements 31
Figure 4.2 Intra-route neighborhood movement 32
Figure 4.3 Example of a PDPTW solution. Arc labels present the amount of load being carried. Time windows have been omitted for simplicity. 36
Figure 4.4 Example of route representation 36
Figure 4.5 Example of accumulated demand vector 37

Figure 5.1 Graphic of average running times for instances type 1 and 2 47

LIST OF TABLES

Table 5.1 Summary of instances characteristics 41
Table 5.2 Details of the five *toy* instances 41
Table 5.3 Training instances for irace 42
Table 5.4 Parameter tuning ranges for irace 43
Table 5.5 Comparison between CPLEX and IVND for the toy instances 44
Table 5.6 Average Initial and Final Results of IVND for standard instances 45
Table 5.7 Computational Enviroments 48
Table 5.8 Comparison of IVND average results with the main literature methods 49

Table 6.1 Results for the *Fast Deliveries* instances 52
Table 6.2 Results for the *Programmed Deliveries* instances 53

Table A.1 Average results for 100 locations instances 61
Table A.2 Average results for 200 locations instances 62
Table A.3 Average results for 400 locations instances 63
Table A.4 Average results for 600 locations instances 64
Table A.5 Average results for 800 locations instances 65
Table A.6 Average results for 1000 locations instances 66

LIST OF ABBREVIATIONS AND ACRONYMS

API	Application Program Interface
CPLEX	IBM ILOG CPLEX Optimization Studio
CPU	Central Processing Unit
CVRP	Capacitated Vehicle Routing Problem
CVRPTW	Capacitated Vehicle Routing Problem with Time Windows
DARP	Dial-a-Ride Problem
ILS	Iterated Local Search
IVND	Iterated Variable Neighborhood Descent
MILP	Mixed Integer Linear Programming
OR	Operations Research
PDPTW	Pickup and Delivery Problem with Time Windows
TSP	Travelling Salesman Problem
VNS	Variable Neighborhood Search
VND	Variable Neighborhood Descent
VRP	Vehicle Routing Problem
VRPTW	Vehicle Routing Problem with Time Windows

CONTENTS

1 INTRODUCTION 10
1.1 Motivation 10
1.2 Vehicle Routing Problems 12
1.3 Solution Methods 13
1.4 Overview 14
2 PROBLEM DEFINITION 15
2.1 Mathematical Model 16
3 LITERATURE REVIEW 20
3.1 Methods for solving the PDPTW 20
3.1.1 Heuristic methods 20
3.1.2 Exact methods 22
3.2 Metaheuristics 23
3.2.1 Iterated Local Search 23
3.2.1.1 Iterated Local Search Applications 25
3.2.2 Variable Neighborhood Descent 26
4 PROPOSED ITERATED VARIABLE NEIGHBORHOOD DESCENT 27
4.1 Algorithm 27
4.1.1 Initial Solution 28
4.1.1.1 Route Initialization 29
4.1.1.2 Request Insertion 29
4.1.2 Local Search 30
4.1.2.1 Shift Request 31
4.1.2.2 Exchange Request 31
4.1.2.3 Rearrange Request 32
4.1.2.4 Unbalanced Shift Request 32
4.1.3 Perturbation 33
4.1.3.1 Multiple Reinsertions 33
4.1.3.2 Multiple Exchanges 33
4.1.4 Acceptance Criterion 34
4.1.5 Parameters 34
4.2 Data Structures and Speedup 35
4.2.1 Solution Representation 35
4.2.1.1 Forward and Backward Vectors 35
4.2.1.2 Auxiliary Vectors 37
4.2.2 Forward Time Slack 37
4.2.3 Movement Memory 38
5 RESULTS 40
5.1 Benchmark Instances 40
5.1.1 Li & Lim Instances 40
5.1.2 Toy Instances 41
5.2 Configurations and Parameters 42
5.3 Evaluation 43
5.3.1 Toy Instances 43
5.3.2 Li & Lim Instances 44
5.3.2.1 General Performance of IVND 45
5.3.2.2 IVND and Literature Methods 48

6 REAL-WORLD CASE STUDY 50
6.1 Data Sets 50
6.1.1 Experiment analysis 51
7 CONCLUSION 55
REFERENCES.....57
APPENDIX A — RESULT TABLES FOR THE STANDARD INSTANCES.....60

1 INTRODUCTION

Transportation and mobility in modern societies are seen as major concerns by people, companies and the public services. These problems have been studied with much interest by the scientific community for years, contributing to a better logistic network, cost reduction, and the improvement of service quality and urban mobility.

In the field of combinatorial optimization and operations research, the problem related to transportation and mobility that has received a lot of attention is the *Vehicle Routing Problem* (VRP). The VRP aims at building a set of vehicle routes to attend a set of customers, so that operational costs are minimized. It is used to model several real-world situations, and has many variations, each one considering different constraints and scenarios.

This work focuses on the case where goods should be transported from one location to another, while respecting the capacity of the vehicles, as well as the specific periods of time when goods can be picked up and delivered at each location. In the scientific literature this problem is modeled as a variation of the VRP, known as *Pickup and Delivery Problem with Time Windows* (PDPTW), which has a great applicability in the transportation field.

1.1 Motivation

There is a wide range of practical applications for the PDPTW, including product delivery, courier services, dial-a-ride problems, bus routing, bulk product transportation and pickup and delivery for overnight carriers. According to Trego and Murray (2010), all of them spend enormous amounts of money on a daily basis, mainly due to fuel, equipment, maintenance and wages. Moreover, studies of King and Mast (1987) and Rodrigues, Comtois and Slack (2013) show 10% to 15% of a product's cost comes just from transportation, so it seems fair to optimize the process involving such activity.

A related issue is the amount of time customers spend waiting the delivery of an order or service, usually missing an entire shift of day at home and possibly having to deal with delays. In order to maximize customers' convenience, companies have to carefully schedule the order of the visits to minimize delays and perform deliveries within the promised time to the customer.

When considering municipal laws in Brazil, there are cities where delayed deliv-

eries may also incur penalties to companies. In São Paulo city, for example, this happens since 2009[1], being stated that a company must clearly define shift hours (morning, afternoon, or night), and perform the delivery within the shift chosen by the customer. In Porto Alegre city a similar law is being discussed[2], which stipulates at most one hour of delay to perform a delivery, from the time agreed with the customer. It is expected that cases like these will become more common in the next years.

Additionally, the transportation sector accounts for a great percentage of greenhouse gas emissions in world today. According to Velazquez et al. (2015), 28% of the total emission in the United States is due to this sector, and 25% in the European Union. As the number of vehicles grows, those numbers are expected to reach higher levels as well. Thus, there is also an environmental concern to ease.

To solve the referred problems, it is possible to make use of operations research (OR) techniques. In fact, Toth and Vigo (2001) and Hasle, Lie and Quak (2007) estimate the use of computerized procedures for planning the distribution processes can offer savings of up to 20% to companies. Also, OR techniques can help avoiding delayed services to customers by better scheduling the order of visits, and reducing the number of vehicles used, hence lowering both the CO_2 emissions and the operational cost.

Operations research has been successfully applied in the transportation area. There are companies that exist solely or primarily developing optimization software to this sector. Some examples are the Dutch *Quintiq*[3] that develops general optimization and planning software for companies; the Norse *SINTEF*[4] that develops general optimization software, including for transportation and planning; and the Danish *TetraSoft S/A*[5] that develops software for routing and planning. These companies also hold some of the best known solutions for the benchmark instances used to compare PDPTW solution methods.

As far as we know, there are no OR companies working in Porto Alegre, which is another fact that motivated this research. This work has been done in collaboration with a software service company, uMov.me[6] (located in Porto Alegre), who provided both the data and the scenario description for real-world cases. We are also using this opportunity to approximate our research group to the industry, and helping solving problems of local need. A tool is already being developed with this partner.

[1]Municipal law of São Paulo city - 13.747/09, known as "Right Time Law".
[2]Law Project 195/2015 of Porto Alegre city, also known as "Right Time Law".
[3]http://www.quintiq.com/
[4]http://www.sintef.no/en/
[5]http://www.tetrasoft.dk/english-info/
[6]https://umov.me

At last, the PDPTW is a $\mathcal{NP}$-Hard combinatorial optimization problem, which means there is no polynomial time algorithm capable of solving it, unless $\mathcal{P} = \mathcal{NP}$. So, there is also a theoretical interest in the study of such problem and in the development of efficient algorithms to solve it.

It is safe to say, then, that the PDPTW has a wide range of scientific and real-world applications, and studying it is worthwhile. In this context, we hope with this work to bring some contributions to the OR area, especially to the optimization of transportation.

1.2 Vehicle Routing Problems

The *Pickup and Delivery Problem with Time Windows* is part of a wider class of problems, the so-called *Vehicle Routing Problems*. The VRP has more than fifty years of scientific studies, with the first work dating from the end of the 1950s (DANTZIG; RAMSER, 1959).

In the VRP, there is a set of requests, or customers, with demands to be supplied by a fleet of vehicles located in a common location, the depot. The goal is to build a route for each vehicle so that all requests are attended and that costs are minimized. The definition of how a route is constructed and costs are minimized are both linked to the variant considered, more specifically, to its restrictions.

The VRP generalizes the classical $\mathcal{NP}$-Hard *Travelling Salesman Problem* (TSP), so it is $\mathcal{NP}$-Hard as well. In fact, the TSP can be thought as a special case of the VRP, where the requests are the locations, or cities, which should be visited only once, and there is only one route that starts and ends at the same location. The cost function to be minimized is the total distance travelled. However, the TSP is usually not classified as a VRP variation, having its own set of variations.

The classical and most studied variation of VRP is actually the *Capacitated Vehicle Routing Problem* (CVRP). In it, just as in the TSP, all requests should be visited only once, but they have a certain *demand*, while all the vehicles have a *maximum capacity* to attend all demands. This capacity should never be exceeded during a route. All vehicles start and end their routes at a single common depot, and a vehicle can have at most one route. The cost function to be minimized is the total cost of all routes.

Further, another commonly studied VRP variation is the *Vehicle Routing Problem with Time Windows* (VRPTW). In this case, the requests have a defined time interval in which service can occur. Other restrictions are usually very close to the ones of the CVRP,

and in fact most studies of the VRPTW actually consider the *Capacitated Vehicle Routing Problem with Time Windows* (CVRPTW), being a generalization of the CVRP. The two most common cost functions to be minimized are the total cost of the routes, and the total travel time by all vehicles.

The last variation to be referred and later detailed is the *Pickup and Delivery Problem with Time Windows*, which generalizes the VRPTW. In the PDPTW, the vehicles no longer deliver goods from a depot to the customers, but instead the customers need goods to be transported from a *pickup* location to a *delivery* location. These visits should also respect a time interval to happen at each location, just as in the VRPTW. The cost function to be minimized is usually the total cost of all routes, the number of vehicles used, or even a combination of both.

1.3 Solution Methods

Ropke (2005) makes the following statement about the vehicle routing problem addressed by this work:

> For many of the problems [...] the set of feasible solutions is so large that even if we had a computer that in a systematic way could construct and evaluate the cost of a trillion (10^{12}) solutions per second, and we had started that computer right after the big bang, 14 billion years ago, it would still not have evaluated all the feasible solutions today.

Even though the computational power has greatly increased in the past years, there is still no possible way of solving these combinatorial optimization problems purely by enumeration. Particularly, when considering real-world situations, the best solution is not necessarily the one with the least cost, but the one that can be obtained fast *and* with reasonable cost. Usually, a fast procedure should run in a matter of seconds, or at most a couple of minutes, depending on the application.

The case mentioned is exactly of the same type as in the PDPTW, where good solutions should be found as fast as possible for a big number of requests. In such situations, it is common to turn to heuristic methods, which can return good solutions quickly if well planned.

Taking all of this into account, this work proposes a study in the application of the Iterated Local Search metaheuristic (LOURENÇO; MARTIN; STÜTZLE, 2010) for solving the PDPTW. Furthermore, the mathematical formulation of the problem is presented, and the model is solved by the CPLEX commercial solver. Both methods are compared to

other works in the literature through the use of well known benchmark instances. Moreover, tests are carried out with instances built from real-world data provided by our partner company. The heuristic and CPLEX are compared in this scenario to further analyse their application in a real-world case.

1.4 Overview

This work is organized as follows. In Chapter 2 the problem is completely detailed and a mathematical formulation is given. In Chapter 3 a literature review is presented considering the main methods for PDPTW, as well as metaheuristics used in this work. The proposed algorithm is described in Chapter 4. Results for the standard benchmark instances of the literature are given in Chapter 5. In Chapter 6 a set of real-world instances is evaluated. The work is concluded in Chapter 7.

2 PROBLEM DEFINITION

In the studied *Pickup and Delivery Problem with Time Windows*, a set of routes has to be constructed in order to satisfy transportation requests. The transportation requests specify an origin location, referred as *pickup*, and a destination location, referred as *delivery*. A delivery may only happen after its corresponding pickup (so called *precedence constraint*). A fleet of identical vehicles is available to attend such requests with a given maximum capacity. Each request should be transported by only one of these vehicles, that is, there is no transhipment.

Cordeau, Laporte and Ropke (2008) define the same version of PDPTW as a *Multi-vehicle one-to-one static Pickup and Delivery Problem with Time Windows*. It is said to be *multi-vehicle* because allows more than one route, as opposed to *single-vehicle* variations where only one route is allowed. It is called *one-to-one*, because for each pickup request there is only one corresponding delivery. And it is a *static* version because all requests are known beforehand, while in the *dynamic* requests become available during the optimization process.

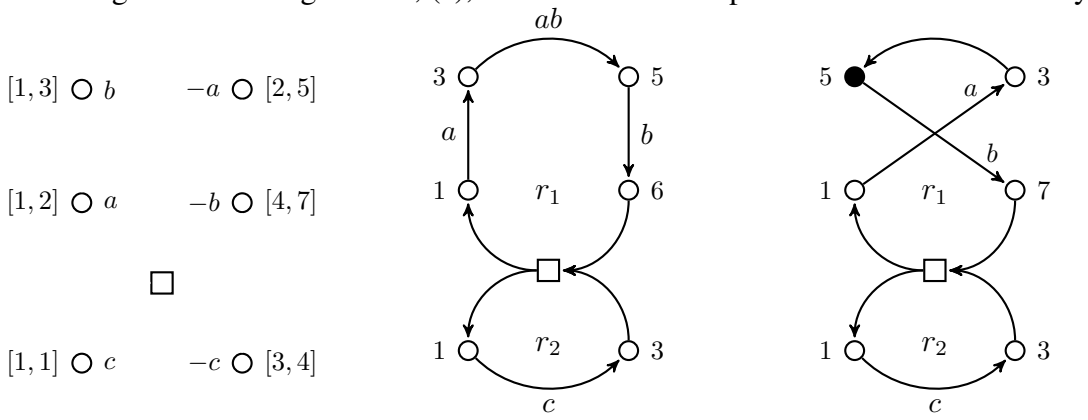
The PDPTW has a set of n transportation requests and all of them should be attended. Each request has: (i) pickup location; (ii) delivery location; (iii) time window for pickup, indicating the earliest and latest time the pickup may be performed; (iv) time window for delivery; (v) service time, or how much time a vehicle takes to complete the service; and (vi) demand, how many units of goods the vehicle should pickup and deliver. The demand of the delivery location should be strictly complementary to the one of the pickup location, i.e., if the pickup location has demand a, the delivery should have demand $-a$.

It is important to distinguish between the types of time windows restrictions. These can be *soft* time windows, or *hard* time windows. The former considers a scenario where time windows can be violated, in order to perform all the deliveries. The latter considers the opposite scenario, where time windows cannot be violated, and a violation leads to an infeasible solution. The problem being studied considers only *hard time windows*.

The fleet of m vehicles is located at a common starting location, referred as *depot*, from where vehicles start and end their routes. The problem considers that there is only one depot, and it has its own time window, defining the size of the planning horizon, or the maximum time a vehicle route can have.

A solution to the PDPTW can be given in a graph as a set of vehicle routes, and a vehicle route is a set of ordered locations, or nodes, to be visited. Figure 2.1 shows an example of PDPTW solutions in a graph. In Figure 2.1(a) the locations are presented, together with their time windows [min, max] and demands. The time window of the depot has been omitted for simplicity. In Figure 2.1(b) a feasible solution is presented, i.e., a solution respecting all constraints of the problem. The solution s has two routes r_1 and r_2, and is denoted by $s = \{r_1, r_2\}$. Finally, solution in Figure 2.1(c) is infeasible because the dark node is reached by the vehicle at time 5, while its time window states the maximum time of 3 ($[1, 3]$).

Figure 2.1: Example of PDPTW solutions. (a) Locations with demands and time windows [min, max], square node is the depot and circles are requests' locations; (b) and (c) possible solutions, with node labels being the exact time a vehicle reaches the node, and arc labels being the load being carried; (c), the dark node is responsible for the infeasibility.



(a) Locations in a graph (b) A feasible solution (c) An infeasible solution

Source: From the Author

The aim of the PDPTW is to find a feasible solution, so that the number of vehicles ($|s|$) is minimized, and the total cost of the routes is also minimized. This defines and hierarchical order of minimization, the first being the number of routes, and the second the total operational cost of the routes.

A mathematical formulation can be given to this problem so that it can be better understood. Also, this allows it to be solved exaclty by a generic solver, such as CPLEX. The formulation is given in Section 2.1.

2.1 Mathematical Model

This section presents a formal description of the PDPTW through a mathematical model in Mixed Integer Linear Programming (MILP) form, based on the work of

Grandinetti et al. (2014). The formulation considers the same objective function of Li and Lim (2003), minimizing first the number of vehicles, and second the cost of all routes.

As in other variations of VRP, the problem is defined on a graph $G = (V, A)$, where V is the set of all nodes and A the set of all arcs connecting two nodes. An usual PDPTW scenario has n requests and a maximum of m vehicles to be used. This defines the following sets: $P \subset V$ the set of all pickup locations, $D \subset V$ the set of all delivery locations and K the set of all vehicles, such that $|P| = |D| = n$ and $|K| = m$. Each node $p \in P$ has strictly one corresponding delivery pair, denoted as $dev(p)$; analogously, each node $d \in D$ has an unique pickup location. Two depots are considered in this model: the departure $\delta_0 = 0$, and the arrival $\delta_1 = 2n + 1$, which could be physically the same. Then, the set of all nodes is given by $V = P \cup D \cup \{\delta_0, \delta_1\}$. Also, each arc $a \in A$ connecting two nodes $i, j \in V$ has a time t_{ij}, and a cost c_{ij} of using this arc. It is assumed the times and costs are non-negative, and that arc times satisfy the triangular inequality.

Each node $i \in V$ has a service time s_i and a time window $[e_i, l_i]$. A vehicle is allowed to arrive at a location before service can start (before e_i), but in this case it must wait until the start of time window to perform the visit. Though, a vehicle is never allowed to arrive after the maximum time l_i. Additionally, every node has a demand Q_i associated, being $Q_i > 0$ when $i \in P$, $Q_i < 0$ when $i \in D$ and $Q_i = 0$ when $i \in \{\delta_0, \delta_1\}$. This demand corresponds to the amount of goods a vehicle must pickup or deliver at the given location.

The homogeneous fleet of vehicles has a maximum capacity U per vehicle. There is also a cost associated for allocating a vehicle to a route, given as a weight parameter ω. This weight should be big enough in order to dominate the objective function's value and drive the search to solutions with fewer vehicles. In practice, ω is defined according to the number of locations in the problem, so that it is able to dominate the value.

Four sets of decision variables are used in this model: $x_{ijk}, i, j \in V, k \in K$, a binary variable which assumes *one* if the arc (i, j) is traversed by vehicle k, and *zero* otherwise; $y_k, k \in K$, a binary variable which takes value *one* if vehicle k is used, and *zero* otherwise; $h_{ik}, i \in V, k \in K$, a real variable, which indicates the time vehicle k starts service at node i; and $q_{ik}, i \in V, k \in K$, a real variable indicating the remaining capacity of vehicle k before leaving node i. Both variables h_{ik} and q_{ik} are only well defined when vehicle k is used.

Then, the mathematical model is given as follows:

$$\textbf{minimize } \omega \sum_{k \in K} y_k + \sum_{i \in V} \sum_{j \in V} \sum_{k \in K} c_{ij} x_{ijk}, \tag{2.1}$$

subject to

$$\sum_{k \in K} \sum_{j \in V | j \neq i} x_{ijk} = 1 \qquad \forall i \in P \tag{2.2}$$

$$\sum_{j \in V | j \neq i} x_{ijk} - \sum_{j \in V | j \neq i} x_{dev(i)jk} = 0 \qquad \forall i \in P, k \in K \tag{2.3}$$

$$\sum_{j \in V | j \neq \delta_0} x_{\delta_0 jk} = y_k \qquad \forall k \in K \tag{2.4}$$

$$\sum_{j \in V | j \neq \delta_1} x_{j \delta_1 k} = y_k \qquad \forall k \in K \tag{2.5}$$

$$\sum_{i \in V} \sum_{j \in V | j \neq i} x_{ijk} \leq W y_k \qquad \forall k \in K \tag{2.6}$$

$$\sum_{j \in V | j \neq i} x_{ijk} - \sum_{j \in V | j \neq i} x_{jik} = 0 \qquad \forall i \in P \cup D, k \in K \tag{2.7}$$

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

$$h_{dev(i)k} \geq h_{ik} + t_{idev(i)} \sum_{j \in V | j \neq i} x_{ijk} \qquad \forall i \in P, k \in K \tag{2.9}$$

$$h_{ik} \geq e_i \qquad \forall i \in V, k \in K \tag{2.10}$$

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

$$q_{jk} \geq q_{ik} + Q_j - W(1 - x_{ijk}) \qquad \forall i, j \in V | i \neq j, k \in K \tag{2.12}$$

$$\max(0, Q_i) \sum_{j \in V | j \neq i} x_{ijk} \leq q_{ik} \qquad \forall i \in V, k \in K \tag{2.13}$$

$$\min(U, U + Q_i) \sum_{j \in V | j \neq i} x_{ijk} \geq q_{ik} \qquad \forall i \in V, k \in K \tag{2.14}$$

$$x_{ijk}, y_k \in \{0, 1\} \qquad \forall i, j \in V, k \in K \tag{2.15}$$

$$h_{ik}, q_{ik} \in \mathbb{R} \qquad \forall i \in V, k \in K \tag{2.16}$$

As stated previously, the objective function (2.1) minimizes the accumulated costs. Parameter ω is responsible for relating the cost of each route in solution to the number of vehicles used, resulting in the final total operational cost.

Constraint (2.2) ensures that all requests are attended, while (2.3) ensures the pairing condition, that is, if a vehicle k serves a pickup p, then k must also serve the delivery pair of p, $dev(p)$. Constraints (2.4) and (2.5) assure that for each route, only one arc

leaves depot δ_0 and only one arc arrives at depot δ_1, respectively. Constraints (2.6) link the x-variables to the y-variables: if an arc (i, j) is traversed by vehicle k, then k must be considered used ($y_k = 1$); on the other hand, if a vehicle k does not traverse arc (i, j), the corresponding x must be zero. Here, W is defined as a large non-negative scalar. Next, constraint (2.7) is the flow conservation constraint, stating that the number of incoming arcs must be equal to the number of outgoing arcs for all nodes, except the depots.

Constraint (2.8) assures all nodes in a route are served after their predecessor, and constraint (2.9) ensures the precedence constraint, that is, a delivery node can only be served after its corresponding pickup node. Constraints (2.10) and (2.11) impose the time window limits for service to occur in node i, being the earliest time and latest time, respectively. Constraint (2.12) updates the remaining capacity of a vehicle before leaving node i. Constraints (2.13) and (2.14) assure that a vehicle's transportation will neither become negative nor exceed its maximum capacity U. Finally, constraint (2.15) ensures the binary condition of the x and y-variables, as well as, (2.16) set variables h and q to be real.

3 LITERATURE REVIEW

This chapter presents a literature review on the main existing methods to solve the PDPTW, as well on the metaheuristics later used to create the proposed algorithm in this work. The first section (3.1) reviews the heuristic and exact methods to solve the PDPTW, and the second section (3.2) reviews the two metaheuristics used in this work.

3.1 Methods for solving the PDPTW

The literature about the PDPTW is not as extensive as about the CVRP or the VRPTW, but has many overlaps with the one of the Dial-a-ride Problem. Two main surveys about the problem can be considered, one by Savelsbergh and Sol (1995), and another by Cordeau, Laporte and Ropke (2008).

As it is usual, the methods for solving the problem can be divided into two classes: the heuristic methods, and the exact methods. This work presents a review on both types, but focusing on heuristics.

3.1.1 Heuristic methods

The first work using metaheuristics is by Bruggen, Lenstra and Schuur (1993). The authors proposed two different approaches to solve only the *single vehicle* version of PDPTW. First a *two-phase local search* algorithm, and a second method using a *Penalized Simulated Annealing*. The methods were not directly compared, being tested against different sets of instances according to the aim each algorithm had.

It was only with the work of Nanry and Barnes (2000) that a metaheuristic was applied to the *multi-vehicle* version of the PDPTW. The proposed method used a *Reactive Tabu Search* with a neighborhood consisting of three simple movements: 1) moving a request from one route to another; 2) exchanging a request in one route with a request in another; and 3) relocating a request to another position within its original route. The method uses a sort of *shaking* to escape local minima. This method was tested in instances with up to 50 requests, however these instances were later considered too easy and ended up unused.

Based on the previous work, Li and Lim (2003) proposed a *Tabu-embedded Sim-*

ulated Annealing approach for solving the PDPTW. The method uses the same neighborhoods of Nanry and Barnes (2000) inside the embedded tabu search. The authors have also introduced a set of new benchmark instances, which became the standard set for the PDPTW, being still used nowadays. The objective function first minimizes the number of vehicles used, and second the total cost of all routes.

Bent and Hentenryck (2006) have applied a *Two-stage Hybrid Algorithm* for the PDPTW, and were able to find good results for the Li and Lim (2003) benchmark instances. The first stage of the algorithm aims at minimizing the number of vehicles used by means of a *Simulated Annealing* that uses a modified objective function to create routes with few requests and routes with many requests, i.e., unbalanced routes. The second stage minimizes the travel distance by means of a *Large Neighborhood Search*, which uses an approach of removing and reinserting requests.

Ropke and Pisinger (2006) developed an *Adaptive Large Neighborhood Search* for the PDPTW, being able to outperform results of the previous methods. In fact, it can still be considered the *state-of-the-art* method for the PDPTW. The method also uses a two-stage approach, but the heuristics used in both stages are exactly the same, only varying the objective function's weights. In the first stage, vehicle minimization is considered, while in the second is the total distance minimization. The method also uses a remove and reinsert approach, but with more types of removal and reinsertion methods (three for removal and two for reinsertion). It is named *adaptive* because the method favors the choice of the removal and reinsertion methods based on how well they behave in run time. This approach was able to improve on more than half of all instances of Li and Lim (2003) data set.

There are also three commercial solvers that hold some of the best known solutions for the standard set of instances. One proposed by SINTEF, another by TetraSoft A/S and another by Quintiq. Details about the methods are not available, but the results obtained are available at SINTEF's website (SINTEF, 2008). Even though they have not been published in a scientific paper, it is important to confirm that there is so much application for the PDPTW that even commercial softwares are testing their quality against benchmark instances used in the academia.

Finally, some of the recorded best known solutions for the Li and Lim (2003) data set are from works in progress by the time this work was published. This is another important fact, because it shows that there is active research on the PDPTW, and that the benchmark instances proposed a decade ago are still challenging.

3.1.2 Exact methods

The first work on exact methods was developed by Desrosiers, Dumas and Soumis (1986). It was a dynamic programming approach for the *single vehicle* version of PDPTW, with rules for eliminating dominated labels. It could handle instances with up to 40 requests.

Dumas, Desrosiers and Soumis (1991) proposed the first column generation approach to the *multi-vehicle* version of the PDPTW. It was embedded into a *branch-and-bound* algorithm, able to handle instances with up to 50 requests. Sol (1994) proposed another column generation with a different pricing problem and branching rules, as well as a procedure for reducing the number of variables in the set partitioning problem. An updated version of this method was presented in the work of Savelsbergh and Sol (1998).

Sigurd, Pisinger and Sig (2004) proposed a column generation embedded into a *branch-and-bound* approach for a closely related variant of the PDPTW. The research was motivated by a real-world problem of transporting live animals, in this case pigs, from one farm to another, avoiding the spread of animal diseases. For example, trucks that have transported sick pigs were not allowed to pickup healthy pigs. This is an extra precedence constraint, and it allowed to evaluate the pricing problem much faster. Tests were carried out with instances containing up to 108 requests with time windows, and 320 without time windows. The instances tested were not from the Li and Lim (2003) data set.

Ropke, Cordeau and Laporte (2007) developed a *Branch-and-cut* algorithm to solve the PDPTW and the *Dial-a-ride Problem* (DARP). The authors also proposed a new set of benchmark instances with up to 96 requests, though they are not widely used. It was only with the work of Ropke and Cordeau (2009) that an exact method was tested against the standard data set, even though only a small set of instances could be solved. The proposed algorithm was a *Branch-and-cut-and-price*, that outperformed the previous algorithm.

More recently, Koning (2011) proposed a Column Generation approach for the PDPTW. In the master problem, it selects the routes that result in the best overall solution, while the subproblem searches for good routes. The standard benchmark set was used for testing, with the method performing better in instances with short planning horizon and 400 customers, because the development was mainly focused on these.

3.2 Metaheuristics

Before describing the proposed method to solve the PDPTW, it is important to be familiar with the metaheuristics employed. Two are combined in our proposed algorithm, namely the *Iterated Local Search* (ILS) and the *Variable Neighborhood Descent* (VND). A review on both is presented in the following subsections.

3.2.1 Iterated Local Search

The ILS is an *stochastic local search* method used for solving hard combinatorial optimization problems. It holds some resemblance to the Random Restart Search, performing a biased walk through the solution space. Although, according to Lourenço, Martin and Stützle (2010) it can outperform Random Restart methods in most cases, both in solution quality and speed. The method's main characteristics rely on its modularity and simplicity, both conceptually and in practice.

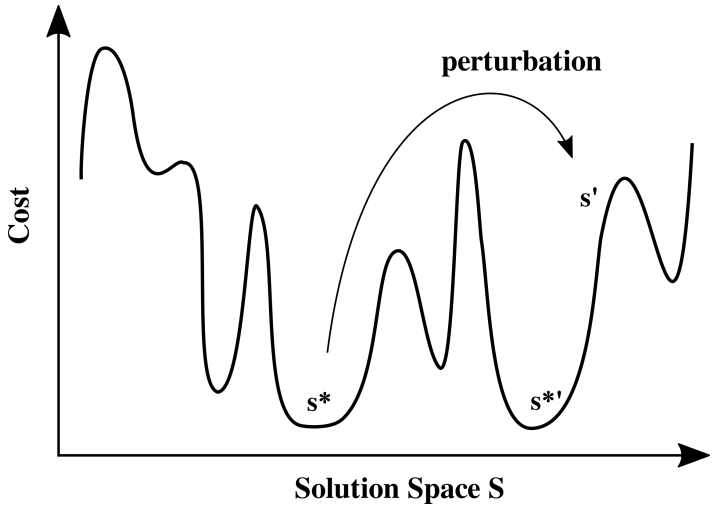
Iterated Local Search works by iterating an embedded heuristic usually called *local search*, even if in fact it has not to be a true local search. This procedure is responsible for reaching a local optimum in the solution space. Because this locally optimal solution is not guaranteed to be the global optimum, ILS applies a *perturbation* to escape this solution and continue the exploration.

The *perturbation* is a slight modification on the current local optimum, just enough to reach a "nearby" solution. Local search is applied to this new starting point to reach another local optimum. This differs from the approach of Random Restart Search, because ILS keeps a certain amount of information about the previously visited local optimum, reducing the actual search space (LOURENÇO; MARTIN; STÜTZLE, 2010).

In Figure 3.1, a pictorial view of ILS is presented. A combinatorial optimization problem with cost function to be minimized is considered, with its solution space denoted by $\mathcal{S}$. Solution s^* is the current local minimum, and the *perturbation* takes it to an intermediate solution s'. Next, a *local search* is applied to s' and another local minimum is reached, denoted by $s^{*\prime}$.

In order to continue the search, one has to decide whether to perturb the new $s^{*\prime}$ or the incumbent s^*. This decision is done by a so-called *acceptance criterion*. The whole process is then repeated, with new local minimum and perturbations, until some *stopping criterion* is met.

Figure 3.1: Pictorial representation of iterated local search.



Source: (LOURENÇO; MARTIN; STÜTZLE, 2010)

A basic framework is presented in Algorithm 1. When designing an ILS approach, only four modules must be considered: GenerateInitialSolution, LocalSearch, Perturbation and AcceptanceCriterion. Those modules can have no interactions whatsoever, but considering their interactions does lead to much better results. A further analysis in each module is given next.

Algorithm 1 Iterated Local Search

1: $s_0 =$ GenerateInitialSolution( )
2: $s^* =$ LocalSearch(s_0)
3: **repeat**
4: $\quad s' =$ Perturbation$(s^*, history)$
5: $\quad s^{*\prime} =$ LocalSearch(s')
6: $\quad s^* =$ AcceptanceCriterion$(s^*, s^{*\prime}, history)$
7: **until** termination condition met

The quality needed for the initial solution s_0 depends on the type of application. If a good solution is to be found *as fast as possible*, then a greedy construction heuristic is generally the best approach. If time is not an issue, a random initial solution can be used, as they tend to converge to the same point when large periods of time are considered.

For most of the combinatorial optimization problems studied, there are already several local search procedures available. Most of them will perform well when embedded in the ILS framework. Another approach that has been used recently is the embedding of metaheuristics as a local search, such as *Tabu Search*, *Variable Neighborhood Search*, and *Simulated Annealing*.

The perturbation mechanism has a great responsibility in the search process. It is said that the perturbation has a *strength*, usually measured by how much of the solution it changes. When the perturbation is *too weak*, the local search may undo the changes and the ILS is kept trapped in the basin of attraction of that local optimum. On the other hand, when the perturbation is *too strong*, only a few, or any information about the current solution is kept, and the search becomes a Random Restart Search. A certain balance has to be taken in order to avoid such cases. The *history* parameter works much like a memory, and can be used to help guiding the perturbation based on previous moves.

Finally, the AcceptanceCriterion module chooses between the current solution $s^{*\prime}$ and the best found so far s^* to continue the search. This module tells if the search will favor intensification or diversification. Several methods can be considered. For example, one can choose to always keep the best solution found so far, thus favoring intensification; analogously, one can always keep the current solution, thus favoring diversification. Many other methods lie in between those two, usually accepting a solution probabilistically, or based on the *history* parameter, so that diversification and intensification are alternated during the search.

3.2.1.1 Iterated Local Search Applications

Iterated Local Search has been successfully applied to many $\mathcal{NP}$-Hard problems, such as the TSP (MERZ; HUHSE, 2008), *Quadratic Assignment Problem* (STÜTZLE, 2006), MAX-SAT (BATTITI; PROTASI, 1997) and *Job Shop Scheduling* (LOURENCO, 1995). It has also several applications to VRP variants, such as the VRPTW (RIBAS et al., 2011), the *Heterogeneous Fleet Vehicle Routing Problem* (PENNA; SUBRAMANIAN; OCHI, 2013), the *Vehicle Routing Problem with Cross-Docking* (MORAIS; MATEUS; NORONHA, 2014), and the *Multi-compartment Vehicle Routing Problem* (SILVESTRIN, 2016).

However, there are no known direct applications of ILS to solve the PDPTW. The work of Nanry and Barnes (2000) that applied *Reactive Tabu Search* with a procedure similar to a *perturbation* holds some resemblance to ILS, but it was never tested against the data set proposed by Li and Lim (2003) to better compare its performance with current methods.

3.2.2 Variable Neighborhood Descent

Variable Neighborhood Descent was first proposed by Mladenović and Hansen (1997) as a variation of the Variable Neighborhood Search (VNS). Both methods are based on the systematic change of neighborhoods within the search, so that the algorithm does not get stuck on a neighborhood's local minima.

To better understand the process, let $\mathcal{N}$ be a set of predefined neighborhood structures given by $\{\mathcal{N}_1, ..., \mathcal{N}_{kmax}\}$, where $\mathcal{N}_k$ is a structure in this set. Then, when a local optimum in neighborhood $\mathcal{N}_k$ is reached, one moves the search to another neighborhood. This way, a different area of the search space is considered, allowing further optimizations.

The main difference between VNS's variations is how neighborhoods are changed. In VND, this change is deterministic, and the stopping criterion is when no further improvements can be done, i.e., a local optimum has been reached with regards to all neighborhoods. This is illustrated in Algorithm 2. For more variations the reader is referred to (MLADENOVIĆ; HANSEN, 1997).

Algorithm 2 VND

Input: Neighborhood set $\mathcal{N} = \{\mathcal{N}_1, ..., \mathcal{N}_{kmax}\}$

```
1:  k = 1
2:  repeat
3:      s*' = N_k(s*)
4:      if s*' < s* then
5:          s* = s*'
6:          k = 1
7:      else
8:          k = k + 1
9:      end if
10: until k > kmax
```

4 PROPOSED ITERATED VARIABLE NEIGHBORHOOD DESCENT

This chapter describes the *Iterated Variable Neighborhood Descent* (IVND) method proposed to solve the Pickup and Delivery Problem with Time Windows. It is assumed, to this end, that the problem is defined on a graph, just as in Chapter 2, and so the algorithm makes use of node and edge manipulations to search in the solution space.

A solution to the problem is taken to be a set of routes in that graph. A route is an ordered set of requests to be visited, with regards to the PDPTW side constraints. Further, for simplicity, a pickup-delivery pair is denoted as a request, and any request p can be decomposed into its two locations, namely p_u, the pickup location, and p_v, the delivery location.

In fact, different from other VRP variations in which one request corresponds to one node in the graph (such as CVRP and VRPTW), the PDPTW considers always two nodes for each request. Thus, when moving a request in the solution's graph, the algorithm has to actually move two nodes, increasing its complexity.

The chapter goes as follows. In Section 4.1 the algorithm is detailed, and parameters are discussed; after, in Section 4.2, an analysis of data structures and speedup techniques used is conducted.

4.1 Algorithm

An Iterated Variable Neighborhood Descent algorithm can be seen as an Iterated Local Search, where the local search is actually a VND. It starts from a local minimum and repeatedly applies a perturbation to escape from it, followed by a Variable Neighborhood Descent to find another local minimum, until a stopping criterion is satisfied. A criterion decides if the newly found local minimum is accepted to continue the search, or if a previously visited solution is to be taken.

The proposed IVND is presented in Algorithm 3. In line 1, the initial solution s_0 is generated, and improved in line 2 by the VND metaheuristic. Then, the main loop (lines 3-7) is repeated until the number of *iterations without improvement*, i_b, reaches a maximum value I, the *stopping criterion*. This loop is responsible for alternating between perturbations, improvements and the acceptance of new solutions to continue the search.

Unless otherwise stated, a solution is evaluated by the function in Equation 4.1, which is a more explicit description of the objective function presented in Chapter 2. It

Algorithm 3 IVND

1: $s_0 = \mathsf{ModifiedInsertionHeuristic}(\ )$
2: $s^* = s^{*\prime} = \mathsf{VND}(s_0)$
3: **repeat**
4: $\quad s' = \mathsf{Perturbation}(s^{*\prime})$
5: $\quad s^{*\prime} = \mathsf{VND}(s')$
6: $\quad s^{*\prime} = \mathsf{AcceptanceCriterion}(s^*, s^{*\prime})$
7: **until** $i_b > I$
8: **return** s^*

is supposed to minimize the values of its terms in hierarchical order. The first term is the number of routes in solution s, and the second term is the total distance travelled by all routes. Note that the second term is usually referred as the total cost, though, in this case distance and cost will be used interchangeably.

$$e(s) = (\ |s|, \sum_{r \in s} d(r)\ ) \tag{4.1}$$

In the following sections, all four modules of the IVND are described and analysed.

4.1.1 Initial Solution

In order to generate initial solutions, a modified version of the *Insertion Heuristic* by Solomon (1987) is proposed to work with pairs of locations. This greedy algorithm was chosen because it is able to produce good solutions quickly, which is an important feature when dealing with vehicle routing problems such as the PDPTW.

The Insertion Heuristic is a *sequential* route construction algorithm, meaning it builds a solution one route at a time. Requests are progressively inserted into their best position in the current route. This is done until no more feasible insertions are possible, when a new route is taken to continue the procedure.

Algorithm 4 describes the *Modified Insertion Heuristic*. It takes as input a set P containing all the n requests to be routed in the problem, and an integer K_m, the number of vehicles available. Then, starting with $k = 1$, it initializes an empty route r_k with one request $p \in P$, a process named *Route Initialization* (detailed in Section 4.1.1.1). Once a route has been initialized, the method proceeds by adding more requests to this route, as long as feasible insertions are possible. The request $p \in P$ that minimizes the insertion cost is chosen to be inserted. This process is named *Request Insertion* (detailed

in Section 4.1.1.2). Every time a request is added to route r_k, it is removed from P.

When no more feasible insertions are possible in r_k, this route is added to solution s. If there are still requests in P, a new empty route is initialized and the procedure is repeated. In case there are no more vehicles available, the algorithm terminates with no feasible solution. Otherwise, if set $P = \emptyset$, a feasible solution s is returned.

Algorithm 4 Modified Insertion Heuristic

Input: set P with n requests to be routed
Input: number of vehicles available K_m
1: $s = \emptyset$
2: $k = 1$
3: **while** $k \leq K_m$ and $P \neq \emptyset$ **do**
4: $r_k =$ initialize route r_k with a request $p \in P$
5: $P =$ remove p from P
6: **while** there is feasible insertion in r_k **do**
7: $p =$ choose request p that minimizes $c_i(b, p, f)$
8: $r_k =$ insert p in r_k
9: $P =$ remove p from P
10: **end while**
11: $s =$ insert r_k in s
12: $k = k + 1$
13: **end while**

4.1.1.1 Route Initialization

Two criteria are taken into account when selecting the first request to be inserted in an empty route. First, from the set P, those requests that have a feasible insertion and the pickup location with minimum value of starting time window are chosen. In cases where more than one request meets the previous criterion, the one with the pickup location closest to the depot (*minimum distance*) is chosen. If there are any ties left, the pickup location with highest index is selected.

This combination of criteria was chosen because resulted in the best average solutions. A similar method was proposed by Li and Lim (2003).

4.1.1.2 Request Insertion

After a route has been initialized with one pickup-delivery pair, the heuristic continues by inserting other requests according to a cost function $c_i(b, p, f)$. This function evaluates the cost of inserting a request p in the current route. Equation 4.3 details the function. Here, request p is decomposed into its pickup location p_u and its delivery lo-

cation p_v. Locations b_u and f_u are adjacent in the current route, and location p_u is to be inserted between them. The same applies to locations b_v, f_v and p_v.

$$c_i(b, p, f) = c_1(b_u, p_u, f_u) + c_1(b_v, p_v, f_v) \tag{4.2}$$

To evaluate the insertion of p_u and p_v, function $c_1(i, x, j)$ is used. It evaluates the cost of inserting location x between locations i and j. Equation 4.3 presents this function, where values of type d_{ij} are the distance from location i to location j.

$$c_1(i, x, j) = d_{ix} + d_{xj} - d_{ij} \tag{4.3}$$

Given that, the request p and adjacent locations b and f which minimize function $c_i(b, p, f)$ are chosen. This is described in Equation 4.4, where p^* is the selected request with least insertion cost between b^* and f^* in route.

$$c_i(b^*, p^*, f^*) = \min[c_i(b, p, f)], \quad \forall p \in P \tag{4.4}$$

It is important to note that only feasible insertions are considered when selecting a position to insert request p. If no feasible insertions are found, it is said p cannot be inserted in this route, and another request is to be tried. This results in only feasible routes being constructed by this procedure.

4.1.2 Local Search

A Variable Neighborhood Descent metaheuristic is employed as the local search module to improve a given solution. The VND has the major advantages of being overall simple and needing only a few parameters. In fact, given only a set of neighborhoods, the method will stop when a local minimum is reached with regards to all neighborhoods.

Recalling Chapter 3, VND needs a set $\mathcal{N}$ of neighborhoods to explore the solution space. The proposed IVND uses four neighborhoods: 1) *Shift Request*, 2) *Exchange Request*, 3) *Rearrange Request*, and 4) *Unbalanced Shift Request*. The first three were proposed and used by Nanry and Barnes (2000), and Li and Lim (2003), while the last is inspired by a method of Bent and Hentenryck (2006).

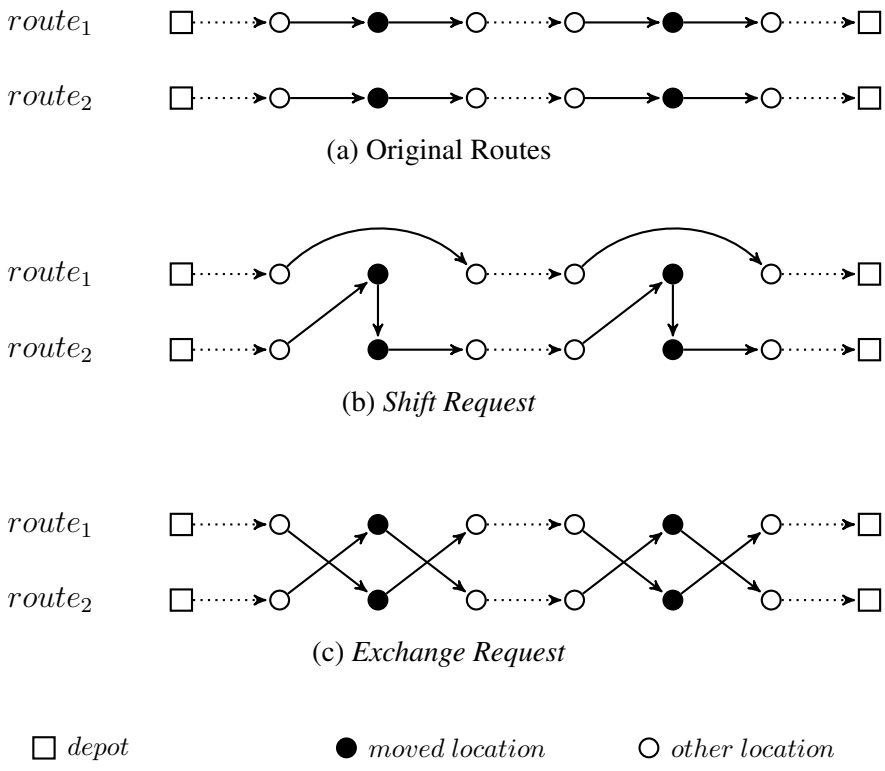
All neighborhoods are of the *best improvement* type, that is, from all possibilities in a given neighborhood $\mathcal{N}_k$, the best is performed. Each one is described next.

4.1.2.1 Shift Request

The first neighborhood attempts to move a request from one route to another in solution. For every pair of routes $r_1, r_2 \mid r_1 \neq r_2$, a request in route r_1 is removed and inserted into all feasible positions in r_2. From all possible pairs and positions, the one that minimizes the cost is chosen. The basic idea is pictured in Figure 4.1(b). Infeasible shifts are forbidden with regards to the PDPTW constraints.

This neighborhood, together with the *Unbalanced Shift Request* (in Section 4.1.2.4), is important for the search, because it is able to reduce the number of routes, i.e, minimize the number of vehicles used.

Figure 4.1: Inter-route neighborhood movements



(a) Original Routes

(b) *Shift Request*

(c) *Exchange Request*

Source: From the Author

4.1.2.2 Exchange Request

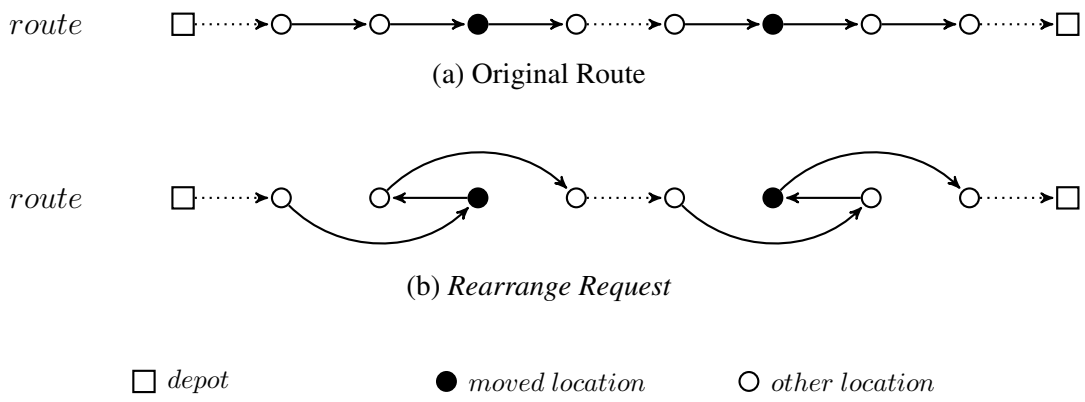
The second neighborhood considered swaps two requests between routes. Again, for every pair of routes $r_1, r_2 \mid r_1 \neq r_2$, a request p_1 is removed from route r_1 and another p_2 from route r_2. Then, request p_1 is inserted into all feasible positions in r_2, as well as p_2 is inserted into all feasible positions in r_1. Again, from all possible pairs and positions,

the one that minimizes the cost is chosen. The basic essence is shown in Figure 4.1(c). Infeasible changes are also forbidden with regards to the PDPTW constraints.

4.1.2.3 Rearrange Request

The third neighborhood is the only *intra-route* neighborhood, meaning its movements only affect a single route. For every route r, a request p is removed and reinserted in another position in r. From all the possible routes and positions, the one that minimizes the cost is chosen. This allows further refinement after other neighborhoods have been applied. Figure 4.2(b) pictures this basic idea. Infeasible changes are forbidden with regards to the PDPTW constraints.

Figure 4.2: Intra-route neighborhood movement



(a) Original Route

(b) *Rearrange Request*

Source: From the Author

4.1.2.4 Unbalanced Shift Request

The last neighborhood is based on the shifting of requests, just like the *Shift Request* in Section 4.1.2.1. Their difference relies on the objective function used to evaluate the movement. In the *Shift Request*, the original evaluation is used (Equation 4.1), while in the *Unbalanced Shift Request* the objective function presented in Equation 4.5 is used.

$$e(s) = (\ |s|, -\sum_{r \in s} |r|^2, \sum_{r \in s} d(r)\ ) \tag{4.5}$$

The terms are also minimized in hierarchical order. The first term is the number of routes in solution s. The second term maximizes $\sum_{r \in s} |r|^2$, which means it favors routes with many locations and with fewer locations, instead of routes with a balanced distribution of them. This can lead the algorithm to remove locations from small routes

and add them to larger routes. The third term minimizes the total distance travelled. This same function was proposed by Bent and Hentenryck (2006) in the first stage of their algorithm to reduce the number of vehicles. As in all the other neighborhoods, infeasible moves are forbidden with regards to the PDPTW constraints.

4.1.3 Perturbation

In order to escape local minimum, ILS employs a perturbation mechanism. The proposed IVND uses two types of perturbation: 1) *Multiple Reinsertions*, 2) *Multiple Exchanges*. They are performed one after the other during the search, so that the perturbation is strong enough to change the solutions visited, but not too much to become a Random Restart Search.

4.1.3.1 Multiple Reinsertions

The first perturbation used is based on the idea of *remove and reinsert*. First, a number $N_r = n_r N$ of requests are randomly removed from solution s, where n_r is a parameter containing the percentage of locations to be removed and N the number of locations in the problem. Then, all N_r requests are reinserted in the same order of removal, but at random positions in the solution, not necessarily belonging to its original route. Infeasible changes are forbidden. If some of the requests could not be reinserted, the perturbation is undone and the original solution s is returned.

4.1.3.2 Multiple Exchanges

This perturbation is based on the previously presented neighborhood *Exchange Request*. This procedure executes a total of n_e exchanges, one by one. For each exchange performed, two routes in solution s are randomly picked, say $r_1,\ r_2 \mid r_1 \neq r_2$. Then, two requests $p_1 \in r_1,\ p_2 \in r_2$ are removed and reinserted at a random position in the other route, i.e., p_1 in r_2, and p_2 in r_1. This is repeated until n_e exchanges, or N^2 unsuccessful tries have been performed, whichever happens first. Infeasible movements are forbidden with regards to the PDPTW constraints.

4.1.4 Acceptance Criterion

The acceptance criterion in ILS is responsible for guiding the search towards solution diversification, or intensification. In other words, it chooses to accept the newly found local minimum $s^{*\prime}$ or the incumbent s^* to continue the search. The proposed IVND employs a criterion based on the amount of *iterations without improvement*.

The method automatically accepts the new $s^{*\prime}$ if it is better than the incumbent s^*. Otherwise, it accepts $s^{*\prime}$ over s^* with probability $\alpha\frac{i_b}{I}$. Here, i_b is the variable containing the number of iterations without improvement, I is the maximum number of iterations without improvement, and α is a parameter responsible for adjusting the acceptance rate. The acceptance criterion is summarized in Algorithm 5.

Such an approach is able to alternate between intensification and diversification of solutions. When the algorithm is successfully improving the incumbent s^*, i_b assumes small values, leading to a higher probability of accepting s^*, and favoring intensification. On the other hand, if the algorithm is trapped in a local minimum, i_b will reach larger values, being more likely to accept $s^{*\prime}$ to continue the search, thus favoring diversification.

Algorithm 5 Acceptance Criterion

Input: Best solution found s^*
Input: Newly found local minimum $s^{*\prime}$
if $s^* < s^{*\prime}$ **then**
    $i_b = i_b + 1$
    if $rand(0,1) > \alpha\frac{i_b}{I}$ **then**
        $s^{*\prime} = s^*$
    end if
else
    $s^* = s^{*\prime}$
    $i_b = 0$
end if

4.1.5 Parameters

The IVND has a set of four parameters to be defined. They are listed below:

1. **Stopping Criterion**: the parameter I, with the maximum number of iterations without improvement, making the algorithm stop when it has converged to a local minimum;

2. **Acceptance rate**: parameter α, with the acceptance rate to choose between s^* and $s^{*\prime}$ to continue the search, given as a percentage;
3. **Perturbation size**: the perturbations sizes, n_e, the number of exchanges (given as an absolute number), and n_r, the number of reinsertions (given as a percentage of the problem size);
4. **Neighborhood order**: the order in which the neighborhoods of VND should be searched, that is, the order of set $\mathcal{N}$.

All parameters can largely impact on the solution's quality and general performance of the method. Because of that, a special tuning is conducted to decide on the best combination of parameters. This process is described in Chapter 5.

4.2 Data Structures and Speedup

An important aspect of metaheuristics is how and which data structures are employed to search for solutions. They have a great impact on an algorithm's performance. Since the proposed algorithm is supposed to be used in the real-world application studied, an algorithm with small response time is needed, thus good data structures and speedup techniques are important.

Those data structure and speedup used in the our algorithm are described in this section. First the representation of a solution is considered, and after specific speedup techniques such as constant time feasibility test and movement memory are detailed.

4.2.1 Solution Representation

As previously mentioned, a solution is a set of routes in a graph, and a route is a set of ordered locations to be visited by a vehicle. Since a solution is trivially defined, let us consider the data structures used to represent a route.

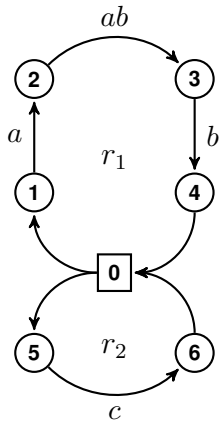
4.2.1.1 Forward and Backward Vectors

In our implementation, each route r_k in solution $s = \{r_1, ..., r_m\}$ is constructed with two main vectors, namely the *forward*, $\vec{f}^k$, and the *backward*, $\vec{b}^k$. For each node in the route, the *forward* vector stores the next node to be visited following the path's order.

Analogously, the *backward* vector stores the incoming node.

In Figure 4.3 a solution to the PDPTW is presented containing two routes. By using the forward and backward structures described, and considering node 0 as the depot, the first location in route r_1 to be visited can be seen as the forward of node 0, $\vec{f}_0^1 = 1$. Similarly, the next node to be visited is denoted by $\vec{f}_1^1 = 2$, while the previous node is $\vec{b}_1^1 = 0$. When looking at route r_2, though, the first node visited after the depot is $\vec{f}_0^2 = 5$.

Figure 4.3: Example of a PDPTW solution. Arc labels present the amount of load being carried. Time windows have been omitted for simplicity.



Source: From the Author

The complete solution in Figure 4.3 is represented by four vectors of constant size $N = 7$, where N is the total number of locations in the problem, including the depot. All four are given in Figure 4.4, the first position is indexed by 0 and is the depot.

Figure 4.4: Example of route representation

$$\vec{f}^1 = \langle\, 1, 2, 3, 4, 0, -, - \,\rangle$$
$$\vec{b}^1 = \langle\, 4, 0, 1, 2, 3, -, - \,\rangle$$

$$\vec{f}^2 = \langle\, 5, -, -, -, -, 6, 0 \,\rangle$$
$$\vec{b}^2 = \langle\, 6, -, -, -, -, 0, 5 \,\rangle$$

Source: From the Author

A route does not necessarily have all the N locations in the problem, as in Figure 4.3, locations 5 and 6 do not belong to route r_1. In the above vectors, the sign $-$ means

the location is not well defined for the considered route, since it is not part of it. This fact could be seen as a drawback of this representation, because it stores $2N|s|$ total locations, as opposed to only N. However, it allows $\mathcal{O}(1)$ access to any node i, as well as its following and incoming nodes, easing the insertion and removal of nodes at any point in the route. Memory not being an issue, this representation can greatly improve the algorithm's performance.

4.2.1.2 Auxiliary Vectors

Besides the *forward* and *backward* vectors, two other auxiliary vectors are also considered, both of size n as well. The first is a vector of accumulated demand, $\vec{q^k}$, that stores how much load a vehicle is carrying before leaving a given node i. In Figure 4.3, the accumulated demand vectors are given as in Figure 4.5.

Figure 4.5: Example of accumulated demand vector

$$\vec{q^1} = \langle\, 0, a, ab, b, 0, -, - \,\rangle$$

$$\vec{q^2} = \langle\, 0, -, -, -, -, c, 0 \,\rangle$$

Source: From the Author

The second auxiliary vector employed stores the accumulated time to reach a given node i. More precisely, it stores the exact moment in time at which the vehicle arrives at node i, but not the time the vehicle starts service at node i, since it can arrive before the start of time window.

Those two auxiliary structures are particularly important because they allow a faster verification of whether the node can be inserted in the given route or not, so that the solution does not become infeasible. Their main usage is to speedup the perturbation process and the forward time slack calculation explained in the next section.

4.2.2 Forward Time Slack

When a problem considers *time windows*, such as in the VRPTW and PDPTW, it can be hard to create efficient methods to check the feasibility of a node's insertion in a

route. In a naive implementation, one would have to check all the following nodes and their time windows to avoid falling into infeasibility. Although, this is a computationally expensive approach, with worst case complexity $\mathcal{O}(N)$.

To overcome this problem, Savelsbergh (1992) proposed a method capable of making the feasibility test in $\mathcal{O}(1)$ time, through so called *Forward Time Slacks*. For each node i, a forward time slack is defined as F_i, and indicates how far the departure time of this node can be shifted forward in time without causing the route to become infeasible.

Taking the departure time of the depot as early as possible $D_0 = e_0$, the forward time slack is given in Equation 4.6:

$$F_i = \min_{i \leq k \leq n} (l_k - (D_i + \sum_{i \leq p < k} t_{p,p+1})) \quad , \forall i \in r_k \tag{4.6}$$

This is defined for all nodes i in route r_k. The term l_k is the latest time in time window of node k, $[e_k, l_k]$. The departure time at node i is denoted by D_i, while term $\sum_{i \leq p < k}(t_{p,p+1})$ is how much time it takes to reach node k. In a concrete implementation, this F_i is calculated for every node, starting from the last node in route ($\vec{b}_0^k$), and going backwards to the first node (0).

Then, to test the feasibility of inserting node i just before a given node j, it is sufficient to test if the amount of time added to reach node j is less then its F_j. This method was first proposed to the VRPTW and TSPTW, however it is easy to extend it to work with pairs of locations.

Nevertheless, the *forward time slacks* only allow testing in $\mathcal{O}(1)$ time, while the insertion and correct update of data structures are still $\mathcal{O}(N)$. Also, every time a modification is performed in a route, the forward time slacks must be recalculated for this route. The improvement in performance is rewarding, though, especially when applied to a *best improvement* strategy.

4.2.3 Movement Memory

One of the major drawbacks when using *best improvement* type neighborhoods, is the computational time needed to perform only a small modification in the solution. If the neighborhoods are iteratevely explored, many possible movements in these neighborhoods are recomputed over and over, even though they remain the same at every iteration. Though, the neighborhood's structure can be exploited to avoid recalculations and

speedup the process.

In the proposed IVND, the neighborhoods are all based in modifications involving one or two routes, only the best being used. When a modification is performed, at most two routes are changed, all the others staying the same. More precisely, the best movements between the unmodified routes also stay being the same in next iterations of VND.

To better understand the idea, consider a PDPTW solution $s = \{r_1, r_2, r_3, r_4, r_5\}$, with five routes, and the neighborhood, $\mathcal{N}_k$, *Shift Request*. Also, suppose in the first iteration the best improving movement in $\mathcal{N}_k$ is between r_1 and r_2. So, r_1 and r_2 are modified, but the rest remains the same. The best possible movement between r_3 and r_4, or r_4 and r_5 are kept the same in the second iteration, and need not to be computed.

Computations are avoided by storing the best movements in a *movement memory*. Considering the previous example, the best movement between r_3 and r_4 is stored, as well as all the others computed. In the second iteration, when the algorithm is to compute the best movement between r_3 and r_4, it first looks at the memory, and if a movement is stored, it is taken, avoiding a possibly long computation time. However, when a route is modified, its memory is erased, because the best movement may have changed, or become infeasible.

In concrete terms, the memory is implemented as a matrix M, with dimensions $|s| \times |s|$. Each neighborhood $\mathcal{N}_k$ has its memory, denoted as M^k. When a movement between two routes r_a and r_b is to be performed in neighborhood $\mathcal{N}_k$, the algorithm checks the memory by index, as M^k_{ab}, looking for an available entry. If it exists, it is taken, otherwise the algorithm computes an available movement normally.

This data structure is important when large number of requests and vehicles are considered, as the computational time of the neighborhoods grows quickly. In fact, experiments have shown that this *movement memory* can improve the algorithm's performance up to a matter of four times, reducing the computation time by 75%. It is important to note that the fewer the vehicles used in a solution s, the less effective this memory gets.

5 RESULTS

This chapter presents the results of the Iterated Variable Neighborhood Descent method obtained from the standard set of benchmark instances of the PDPTW. For real-world instances and application one is referred to Chapter 6.

The text goes as follows. First it is described the test environment and methodology for the choice of parameters' values. Then, the benchmark instances are detailed. A comparison is done between the proposed IVND, CPLEX and the main algorithms in the literature that solved the same set of instances.

5.1 Benchmark Instances

The benchmark data set used is the same proposed by Li and Lim (2003), available at SINTEF's website (SINTEF, 2008), and taken as the standard for the PDPTW. Even though there are some small and easy instances among this set, the CPLEX solver running the MILP formulation is not able to solve them. Thus a subset with smaller instances has been generated. These instances have been named *toy instances*, and are described next, after the Li and Lim (2003) data set.

5.1.1 Li & Lim Instances

These instances were generated from the VRPTW instances proposed by Solomon (1987) and from the larger VRPTW instances by Gehring and Homberger (1999). The instances were created by first solving the VRPTW with an heuristic method, and then randomly pairing nodes that appeared in the same route in the obtained solution. A total of 354 instances were created, with 100, 200, 400, 600, 800, and 1000 total locations.

For each size, instances are divided into sets according to the locations' spatial distribution: locations clustered (*LC*); locations randomly distributed (*LR*); and locations partially clustered and partially randomly distributed (*LRC*). Additionally, they are divided according to the planning horizon, measured by the maximum arrival time at the depot. Instances of type 1 (*LC1, LR1* and *LRC1*) have short planning horizon, while those of type 2 (*LC2, LR2* and *LRC2*) have a longer planning horizon. Table 5.1 presents a summary of instances characteristics.

Table 5.1: Summary of instances characteristics

Instance Size	**Number of Instances**						**Vehicles Available**
	$LC1$	$LC2$	$LR1$	$LR2$	$LRC1$	$LRC2$	
100	9	8	12	11	8	8	25
200	10	10	10	10	10	10	50
400	10	10	10	10	10	10	100
600	10	10	10	10	10	10	150
800	10	10	10	10	10	10	200
1000	10	10	10	10	10	8	250

This set considers that the locations are in a 2D Cartesian plane, and the distances between each pair of locations are given by the *Euclidean distance*. The time t_{ij} and cost c_{ij} to go from one location i to a location j have the same value of the distance d_{ij} between these locations. Because of that, the terms distance and cost are used interchangeably in the evaluation. Although, this relation only holds for this particular set of instances, not being true for the PDPTW as a whole.

5.1.2 Toy Instances

Because the smallest instances in the standard set have 100 locations, that is considered large for CPLEX, a subset of smaller instances has been generated in order to run CPLEX up to optimality, within a maximum time limit of 20 hours. Five instances have been created with different sizes: 10, 20, 30, 40, and 50 locations.

The instances were all generated from the first instance of the 100 set, namely instance *LC101*, by taking random pairs of locations fom it to produce the smaller instances. The number of vehicles available was taken to be half the total number of locations in the problem. The distance considered is also the *Euclidean distance*, as well as time and costs are the same as the distance.

The basic characteristics are presented in Table 5.2.

Table 5.2: Details of the five *toy* instances

Instance	**Number of Locations**	**Vehicles Available**
LC1_10	10	5
LC1_20	20	10
LC1_30	30	15
LC1_40	40	20
LC1_50	50	25

5.2 Configurations and Parameters

The proposed IVND has been implemented in C++ and compiled with the GNU C++ compiler `g++` with flags `-std=c++11 -O3 -lm`. Tests were carried out in a computer with AMD FX-8150 processor containing 8 cores running at 3.6 GHz, and with 32 GB of RAM memory. Only one core has been used during the tests. Also, the operating system Ubuntu 16.04 LTS 64-bits has been used.

Recalling Chapter 4, the IVND has four parameters: stopping criterion (I), acceptance rate (α), perturbation size (n_r, n_e) and neighborhood order ($\mathcal{N}$). Since they can assume a wide range of values, and have impact on each other, an automatic methodology is needed to best choose their values. This process is usually called *parameter tuning*.

The tuning has been done through the use of the irace package[1]. The scenario defined for irace had a maximum number of experiments of 1000, and the instances used for training are reported in Table 5.3, with a total of 18 instances, 3 for each size. The choice had no special methodology, but we tried to diversify between type 1 and type 2 instances. Parameters' ranges tested, as well as the best values reported by irace can be seen in Table 5.4.

Table 5.3: Training instances for irace

Inst. Size	**Inst. LC**	**Inst. LR**	**Inst. LRC**
100	LC104	LR105	LRC201
200	LC1_2_5	LR2_2_8	LRC1_2_5
400	LC2_4_5	LR1_4_5	LRC1_4_5
600	LC2_6_10	LR1_6_5	LRC2_6_2
800	LC1_8_5	LR1_8_10	LRC1_8_5
1000	LC1_10_5	LR1_10_4	LRC1_10_5

In the tuning process, parameter α is allowed to take values in the set $\{0.0, 0.3, 0.5, 0.9, 150.0\}$. If it assumes 0.0, the algorithm always accepts the newly found local minimum ($s^{*\prime}$). On the other hand, if it takes value 150.0, the algorithm always accepts the incumbent solution (s^*). The other values are allowed so that these acceptances are alternated.

The set of ordered neighborhoods $\mathcal{N}$ is allowed to take all the possible permutations of the four neighborhoods available. This is denoted by {SR, ER, RR, USR} in

[1]The irace package implements the iterated racing procedure, which is an extension of the Iterated F-race procedure, and has a strong statistical basis. Its main purpose is to automatically configure optimization algorithms by finding the most appropriate settings given a set of instances of an optimization problem (LÓPEZ-IBÁNEZ et al., 2011)

Table 5.4. The acronyms stand for the neighborhoods' names: *Shift Request* (SR), *Exchange Request* (ER), *Rearrange Request* (RR) and *Unbalanced Shift Request* (USR).

Table 5.4: Parameter tuning ranges for irace

Parameter	**Values Range**	**Best Value**
I	{50, 100, 150}	100
α	{0.0, 0.3, 0.5, 0.9, 150.0}	0.9
n_e	{0, 2, 4, 6, 10}	4
n_r	{0.00...0.15}	0.02
$\mathcal{N}$	{SR, ER, RR, USR}	USR, RR, ER, SR

5.3 Evaluation

The evaluation of the proposed IVND has been done in two steps. First, the *toy instances* have been used to compare CPLEX running the MILP formulation and the IVND. Then, the standard set of instances has been used to analyse the general performance of IVND compared to the main literature methods.

The comparison considers both *solution quality* and *time*, because these are two important characteristics when designing a real-world application solver. The solution quality is measured according to the objective function considered to the PDPTW, first minimizing the number of vehicles and then the total cost of the routes.

At last, the parameters configuration used is the same returned by the irace run, presented in the previous section.

5.3.1 Toy Instances

This evaluation considers IVND, and CPLEX running the MILP formulation presented in Chapter 2. The *toy instances* have been used in order to better evaluate the CPLEX performance, since it has difficulties solving large instances. CPLEX was allowed to run for at most 20 hours (72000 s) in single thread mode with default configurations for each instance, while 10 runs of IVND were performed and the average results calculated.

Table 5.5 presents the results. Three columns are given for CPLEX and IVND. Column OF gives the value of the objective function of a given solution s, calculated as $OF(s) = \omega * |s| + \sum_{r \in S} d(r)$, where ω is defined based on the number of locations L

in the instance, $\omega = L100$ - this variable cost of the vehicles is needed because the total cost of the solutions grows with the instance size, thus the larger the number of locations, the bigger ω should be. Column gap_f (%)[2] gives the deviation of the obtained solution s, from the best known solution s^* (column BKS), calculated as $\text{gap}_\text{f}(s) = (s - s^*)/s^*$. Finally, column *t(s)* gives the amount of CPU time, in seconds, the method took to reach the reported solution. Solutions presented with an asterisk ($value^*$) have been proven to be the optimal solutions.

Table 5.5: Comparison between CPLEX and IVND for the toy instances

		CPLEX			**IVND**		
Instance	**BKS**	OF	$\text{gap}_\text{f}(\%)$	t(s)	OF	$\text{gap}_\text{f}(\%)$	t(s)
LC_10	1,056.02*	1,056.02*	0.00	1.80	1,056.02*	0.00	0.00
LC_20	4,155.50*	4,155.50*	0.00	3,625.04	4,155.50*	0.00	1.46
LC_30	9,201.94*	9,201.94*	0.00	27,139.61	9,201.94*	0.00	1.79
LC_40	16,303.53*	16,303.53*	0.00	71,941.11	16,303.53*	0.00	1.82
LC_50	25,360.24	-	-	72,000.00	25,360.24	0.00	1.86

Note that CPLEX could not find a feasible solution for the instance with 50 locations within the stipulated time limit. Although, the previous instances were already hard for it to solve, taking one hour to solve an instance with only 20 locations. It is possible to note that the time needed for CPLEX to solve the instances grows exponentially with the instance size, as it would be expected.

The proposed IVND, on the other hand, could handle those small instances very easily, providing the optimal solutions to them in a much smaller time, i.e., less than 2 seconds. This offers a clue on the fact that an heuristic method can be a good choice for solving the addressed problem. More specifically, the IVND can handle small cases well, we still have to analyse its scalability with larger instances.

5.3.2 Li & Lim Instances

This section considers experiments with the IVND and the standard set of instances. Section 5.3.2.1 analyses the general performance of IVND as the instance size grows, while Section 5.3.2.2 compares IVND to the main literature methods.

Because the IVND is an stochastic method, the results presented are the average of a number of runs. For instances of size 100 and 200, 10 runs were allowed, while for

[2]It is denoted gap_f because it is the gap of the *full solution*, that is, the cost of using the vehicles ($\omega|s|$) plus the accumulated cost of the routes ($\sum_{r \in S} d(r)$). It contrasts with the gap_c, which considers only the costs of the routes, and not the one of the vehicles.

sizes 400, 600, 800[3] and 1000[3], 5 runs were allowed. Also, for simplicity, instances are separated by type according to their planning horizon (1 - short, 2 - long), and only the average results for each type and size are presented. For detailed information on each instance, one is referred to Appendix A.

5.3.2.1 General Performance of IVND

Table 5.6 presents the average results for each size and type of instance. Columns $\#V_b$, $\#V_i$ and $\#V_f$ are the average number of vehicles of the best known solution, of the initial solution of IVND, and of the final solution of IVND, respectively. Column gap_c (%) refers to the percentage deviation of the total cost (or distance) of the routes found to the best known solution; It is calculated as $\text{gap}_\text{c}(s) = (s_c - s_c^*)/s_c^*$, where s^* is the BKS, and s_c is the accumulated cost of all routes in solution s. Column gap_v (%) is the percentage deviation of the number of vehicles of the final solution to the BKS, given by $\text{gap}_\text{v}(s) = (s_v - s_v^*)/s_v^*$, where s_v is the number of vehicles used in solution s. The column *t(s)* gives the average CPU time in seconds taken to reach the reported solution.

Table 5.6: Average Initial and Final Results of IVND for standard instances

		BKS		**Initial Solution**			**Final Solution**			
Type	**Inst. Size**	$\#V_b$	Cost	$\#V_i$	gap_c(%)	t(s)	$\#V_f$	gap_v(%)	gap_c(%)	t(s)
1	100	11.10	1,158.50	14.24	28.37	0.01	11.26	1.45	-0.11	4.18
	200	15.43	3,439.06	19.30	39.65	0.06	16.40	7.34	-3.28	60.73
	400	29.37	8,175.50	36.80	42.54	0.31	32.13	11.55	-1.94	222.15
	600	42.47	16,352.54	52.57	47.12	1.31	46.28	11.43	-0.35	763.93
	800	55.13	28,265.12	67.90	48.68	2.83	61.33	14.73	0.90	1,504.54
	1000	68.33	43,590.16	83.27	45.84	14.91	76.29	15.55	2.15	1,790.11
2	100	2.96	906.04	4.30	75.02	0.04	3.05	3.95	1.75	32.54
	200	4.57	2,690.69	6.17	73.73	0.27	5.09	15.17	-0.49	168.23
	400	8.53	6,278.10	11.83	87.31	1.89	10.11	24.27	-5.23	1,801.71
	600	12.03	13,422.25	16.63	82.77	5.59	14.38	26.18	-9.59	4,992.48
	800	15.70	21,595.10	22.03	86.14	15.48	18.88	29.24	-8.16	12,220.91
	1000	19.57	31,431.00	26.50	79.50	25.79	23.21	26.70	-5.07	21,479.84

Note: Detailed results are found in Appendix A.

Regarding the initial solutions obtained by IVND, there is a major difference between solution quality for type 1 and type 2 instances. At first, type 1 instances have 8 more vehicles in average than the BKS, while type 2 instances have only an average of 4 more vehicles. However, the total cost of the routes has an average gap_c of 40% for type 1 instances and 80% for type 2 instances. The main reason for these differences is that the type 1 instances have a shorter planning horizon, that is to say that routes tend to be smaller, thus more routes need to be created at first. On the other hand, it is easier to place

[3]For some of the 800 and 1000 instances of type 2 (long planning horizon) only one run was possible, because of the amount of time required to complete one.

new requests on better positions when routes are small, reducing the the total cost, and hence the gap_c.

As to the final solutions, the improvements obtained relative to the initial solutions are meaningful. The number of additional vehicles drops to only 4 for type 1 instances and only 2 for type 2, a reduction by half when compared to the initial solution. However, the average gap_v of the instances is high, reaching 15% on type 1 and 29% on type 2 instances. This deviation shows clearly that the final solutions obtained by the IVND are not as good as the best known solutions, reaching very large gaps, mainly on type 2 instances.

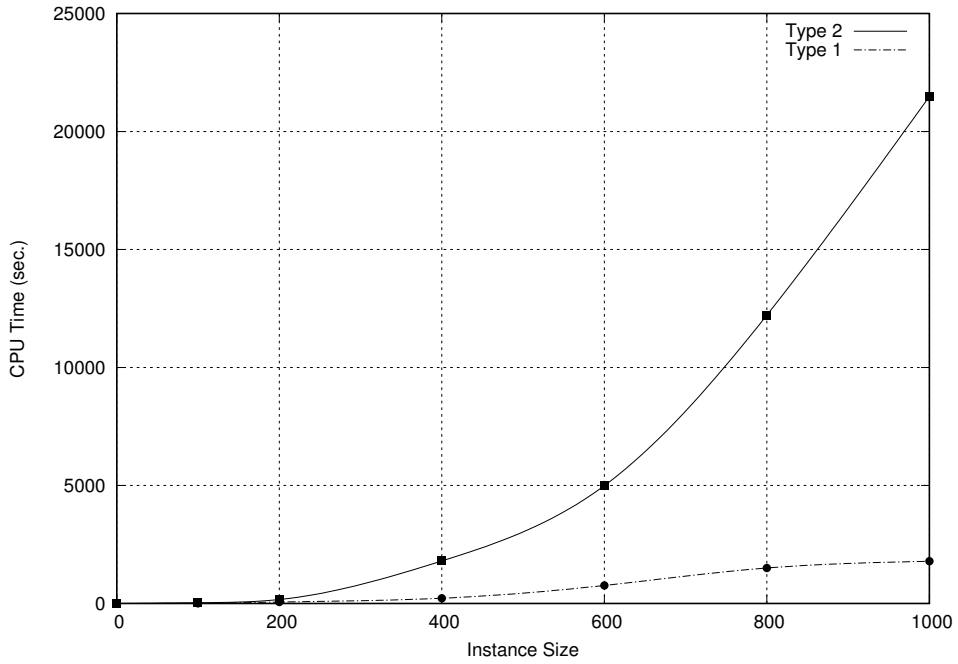
On the other hand, the final gap_c decreases to -0.44% for type 1 instances and to -4.46% for type 2. These negative gaps are possible, even though the solution is not better than the BKS, because when the IVND can no longer reduce the number of vehicles, it reduces the total cost of the routes. Moreover, with more routes, it is easier to construct routes of smaller cost than when using less vehicles.

The analysis of those results makes it clear that the proposed IVND is able to improve the initial solution, as it would be expected. In fact, it is able to reduce both the number of vehicles and the total cost of the routes, sometimes reaching the best known solution (appendix A). Additionally, the method presents better performance on the smaller instances, of size 100, 200 and some of 400. However, as the instance size grows, the average difference on the number of vehicles also grows. This is another expected effect, because with the growth of the instance size there is also the growth of the solution space, which is already enormous for 100 locations instances.

Another important conclusion to take from Table 5.6 is the running time of the algorithm, and if it scales as the instance size grows. For the IVND, there is a major difference in running times when comparing type 1 instances to type 2 instances. Figure 5.1 presents a graphic comparing the average time to reach the final solution for each instance type, according to the growth of the instance size.

Clearly the type 2 instances have a much steeper growth on the average time relative to the instance size, while type 1 instances have an almost linear growth. The main cause for this is the structure of the solutions generated in each type. For type 1 instances, solutions have more vehicles, thus each route has fewer locations. This is an important fact, because the size of a route greatly impacts on the number of combinations of the neighborhoods used in the IVND, since they perform movements between routes in a *best improvement* manner.

Figure 5.1: Graphic of average running times for instances type 1 and 2



Source: From the Author

More specifically, the *movement memory* is the technique that can overcome the problem. On the one hand, it is very effective in reducing the computation time when a solution has many routes. On the other hand, when a solution has many locations (e.g., 1000) and only a small number of vehicles, the memory can only save some calculations, and the modified routes that need to be recomputed are still time consuming, because of their size. This is exactly the case of type 2 instances, and the reason why its growth in time is so steep. The speedup technique of the *movement memory* has little effect on them, but a larger effect on type 1 instances, allowing a scalable growth on the latter.

To make the previous analysis clear, take for example two routes that have been modified in an iteration of the IVND, call them r_i and r_j. Then, in the next iteration, only the movements regarding r_i or r_j will be computed. Also, assume that all routes in the solution have the same size, and take only the exchange neighborhood to calculate the movements, which has worst time complexity $\mathcal{O}(n^2)$. For routes of size a and b, the update time would be $\mathcal{O}(a^2)$ and $\mathcal{O}(b^2)$, respectively. If the relation $a \leq b$ holds, than the update time of the smaller routes (size a, type 1) is clearly smaller than of the larger (size b, type 2).

At last, it is important to note that the proposed IVND always keeps solution feasi-

bility, which is expensive, especially in a problem with many restrictions as the PDPTW. Even though the precedence constraint feasibility, as well as the capacity feasibility, are important in a real-world situation, the time windows could be allowed to become infeasible, for an easier search on the solution space, and still be acceptable from the real point of view.

In fact, two possible improvements for further research would be to drop the *best improvement* strategy, and to allow some infeasibility of solutions.

5.3.2.2 IVND and Literature Methods

This section compares the IVND performace to other methods from the literature. The methods are from the works of Li and Lim (2003), Bent and Hentenryck (2006), and Ropke and Pisinger (2006). These have been chosen because they had the most complete set of informations needed to perform the comparisons, both from the quality and from the time point of view.

Table 5.7 presents the computational environment for each of the methods to be compared. Column *Reference* presents the method, and column *Instances* gives the instances that were tested with the method. Column *Environment* gives the processor used. And column *speed(%)* presents an estimated percentage of speed of each computer relative to the computer used by this work. This means it is considered that the environment of Li and Lim (2003) is eight times slower, that of Bent and Hentenryck (2006) six times slower, and that of Ropke and Pisinger (2006) five times slower. The estimation is highly conservative, and is based on the PassMark Benchmark[4].

Table 5.7: Computational Enviroments

Reference	**Instances**	**Environment**	**speed(%)**
(LI; LIM, 2003)	{100}	Intel 686 Generation	12.50
(BENT; HENTENRYCK, 2006)	{100, 200, 600}	AMD Athlon K7 (1.2 GHz)	16.67
(ROPKE; PISINGER, 2006)	All	Pentium IV (1.5 GHz)	20.00
This Work	All	AMD FX 8150 (3.6 GHz)	100.00

It is important to note that Li and Lim (2003) have only reported detailed information for instances of size 100, even though the authors had run for all other sizes, it is just that their information has been lost from (SINTEF, 2008). However, the same does not apply to Bent and Hentenryck (2006), that had not run at all for instances of sizes 400, 800 and 1000.

[4]http://www.cpubenchmark.net

In Table 5.8 the results of IVND for each size and instance type are compared to the results of the other three methods. For simplicity, Li and Lim (2003) is denoted by LL, Bent and Hentenryck (2006) by BVH, and Ropke and Pisinger (2006) by RP. The BKS column has been omitted, though it is available in Table 5.6. The reported times are all normalized to the time of our tests, following the relation in Table 5.7.

Table 5.8: Comparison of IVND average results with the main literature methods

		LL			**BVH**			**RP**			**IVND**		
Type	**Inst. Size**	$\#V$	$gap_c(\%)$	t(s)	$\#V$	$gap_c(\%)$	t(s)	$\#V$	$gap_c(\%)$	t(s)	$\#V$	$gap_c(\%)$	t(s)
1	100	11.21	-1.17	36.95	11.10	0.00	64.81	11.10	0.01	8.19	11.26	-0.11	4.18
	200	-	-	-	15.77	-2.26	172.17	15.64	-1.63	31.67	16.40	-3.28	60.73
	400	-	-	-	*x*	*x*	*x*	30.15	-1.44	108.52	32.13	-1.94	222.15
	600	-	-	-	43.93	-0.88	602.22	43.72	-1.15	272.31	46.28	-0.35	763.93
	800	-	-	-	*x*	*x*	*x*	50.57	-0.98	493.54	61.33	0.90	1,504.54
	1000	-	-	-	*x*	*x*	*x*	70.15	-1.45	756.14	76.29	2.15	1,790.11
2	100	2.96	1.57	159.72	2.96	0.01	57.68	3.00	0.57	18.48	3.05	1.75	32.54
	200	-	-	-	4.77	-0.23	194.34	4.63	0.66	73.75	5.09	-0.49	168.23
	400	-	-	-	*x*	*x*	*x*	8.77	-0.84	243.82	10.11	-5.23	1,801.71
	600	-	-	-	13.37	-0.89	621.80	12.5	-3.74	615.91	14.38	-9.59	4,992.48
	800	-	-	-	*x*	*x*	*x*	16.59	-6.82	1,073.52	18.88	-8.16	12,220.91
	1000	-	-	-	*x*	*x*	*x*	20.48	-3.01	1,414.42	23.21	-5.07	21,479.84

Note: The sign - means there is not enough information to compute the results; and sign *x* means the method was not tested at all against the set of instances

Those results show that the IVND, in spite of its simplicity, is able to reach good solutions in reasonable time for instances of size 100, 200 and 400 (type 1). For these, it has solution quality and run time comparable to the ones found in the literature. However, the *state-of-the-art* method could not be outperformed by the IVND. For all of the cases, Ropke and Pisinger (2006) still hold very good solutions and running times hard to be beaten up. Perhaps the already mentioned improvements on the proposed algorithm could bring better results, and are up to further research.

6 REAL-WORLD CASE STUDY

In this chapter, a real-world application of the Pickup and Delivery Problem with Time Windows is considered. The study was motivated by a partnership with a software service company in Porto Alegre, uMov.me[1]. All the data has been provided by the company, as well as the description of the scenario, which fits perfectly in the case of the PDPTW. Although, for confidentiality reasons, we are not allowed to give much more information about the case.

The scenario is of a store franchise working in Porto Alegre. It offers a product delivery service to its customers with maximum delivery time, that is, *time windows*. The vehicles are located at several stores existing in the city, and each vehicle serves orders from its particular store, the *depot*. Moreover, vehicles may have to *pickup* a product that has been ordered but has no units in the store it is leaving, thus performing a *pickup* in other stores and a *delivery* to the customers.

The orders are split into two different types, according to the time window restrictions and vehicle capacity: *fast deliveries* and *programmed deliveries*. The first type considers a maximum time window of 1 hour to perform the delivery, and a vehicle may perform at most 6 deliveries. The second type has a wider planning horizon of 4 hours, and vehicles can perform at most 40 deliveries, which is the maximum capacity of a vehicle. Also, in *fast deliveries*, only 1 vehicle is considered, while in *programmed deliveries* up to 4 vehicles are available.

It is important to note that the referred division for the orders is how the franchise works today, and not a particular decision of this work. In fact, we tried to better simulate a real-world situation, in order to compare the results to what they use today. Additionally, most instances may seem small when compared to the literature ones. Although, this is caused by the consideration of small planning horizons with capacity restrictions imposed by the franchise.

6.1 Data Sets

A total of 40 instances have been generated with the data provided by the partner company, being 25 of the *fast delivery* type, and 15 of the *programmed delivery* type. The methodology to generate them is explained next.

[1] www.umov.me

The original data contained orders from several depots, thus we had to separate them by depot in order to apply the studied formulation (*single* depot). Requests have been separated according to the depot and to their time windows, so that all instances are feasible. When a request had no pickup location in the provided data, a virtual pickup location was created in order to simulate the *pickup and delivery*. If a *pickup* existed in the data, it was directly used in the instance.

In the provided data, locations were given by their geographical coordinates (latitude and longitude), and were transformed to 2D Cartesian coordinates during the instance generation. Distances have been computed as the *Euclidean distance*. The time was computed assuming an average speed of a vehicle of 40 km/h, which is a reasonable estimation for big cities, according to our partners.

Because of the previously stated characteristics, *Fast Deliveries* instances have at most 6 requests (12 locations) and one vehicle available. Also, all locations have been given a time window of one hour. For *Programmed Deliveries*, there are at most 160 requests (320 locations), a total of 4 vehicles to attend them (each with capacity for 40 requests), and time windows are all of 4 hours.

The tests have been executed in the same computer configuration presented in Chapter 5. Both CPLEX and IVND were tested against the newly generated instances. CPLEX solver was allowed to run for one hour, and the proposed IVND was run 10 times and average results computed. It received the same parameters returned by the irace, also reported in Chapter 5. One important aspect of the real-world application is the response time, which according to our partners, for this case, should not be larger than 10 seconds. Clearly CPLEX is not a good option to solve the problem, though it is a good source for comparison and evaluation.

However, the solutions obtained by the company's current method were not provided, thus we are not able to compare directly our results to the ones obtained in practice. On the other hand, the experiments allowed us to confirm if the run time of the method would be applicable in a real-world situation.

6.1.1 Experiment analysis

The computational results with the *fast deliveries* and *programmed deliveries* are reported in Table 6.1 and Table 6.2, respectively. The first column presents the instance identifier, and column *Size* gives the number of locations in the problem (N locations

means $N/2$ deliveries). The next five columns are results of the IVND. Column #V_f is the number of vehicles used in solution (both initial and final, because the number was not modified). Column D_0 is the distance (in kilometers) of the initial solution, and column D_f is the average distance of the final solution. Column *t(s)* gives the average CPU time in seconds IVND took to reach the final solution, while $\text{gap}_\text{c}(\%)$ column gives the percentage deviation of the initial solution distance to the final solution distance found by IVND. The last three columns present results for CPLEX solver, with #V_c being the number of vehicles used, D_c the distance of the final solution and *t(s)* the total time to find the given solution, or time out. Solutions with an asterisk ($value^*$) have been proven to be optimal.

Regarding the *fast deliveries*, all the instances could be solved in less than one second, most of them up to optimality. It is safe to say that for these instances the algorithm has met the company's demands.

Table 6.1: Results for the *Fast Deliveries* instances

Data		**IVND**					**CPLEX**		
Inst.	Size	#V_f	D_0	D_f	$\text{gap}_\text{c}(\%)$	*t(s)*	#V_c	D_c	*t(s)*
fd3003	2	1	6.86	6.86*	0.00	0.00	1	6.86*	0.00
fd3021	2	1	6.54	6.54*	0.00	0.00	1	6.54*	0.00
fd4030	2	1	3.48	3.48*	0.00	0.00	1	3.48*	0.00
fd1046	2	1	4.99	4.99*	0.00	0.00	1	4.99*	0.00
fd2040	10	1	9.48	9.48*	0.00	0.01	1	9.48*	326.91
fd0090	2	1	2.83	2.83*	0.00	0.00	1	2.83*	0.00
fd1090	4	1	10.73	10.73*	0.00	0.00	1	10.73*	0.03
fd0102	2	1	3.66	3.66*	0.00	0.00	1	3.66*	0.00
fd1102	12	1	9.79	9.79	0.00	0.01	1	9.79	3600.00
fd1104	6	1	12.82	12.82*	0.00	0.00	1	12.82*	0.83
fd2101	12	1	7.54	7.54	0.00	0.01	1	7.54	3600.00
fd2103	2	1	6.10	6.10*	0.00	0.00	1	6.10*	0.00
fd3100	10	1	11.15	10.62*	-7.81	0.01	1	10.62*	192.65
fd4101	4	1	2.53	2.53*	0.00	0.00	1	2.53*	0.03
fd5100	6	1	22.16	22.16*	0.00	0.00	1	22.16*	0.59
fd0111	12	1	5.59	5.59	0.00	0.00	1	5.59	3600.00
fd1111	12	1	10.63	10.63	0.00	0.00	1	10.63	3600.00
fd1115	4	1	6.50	6.50*	0.00	0.00	1	6.50*	0.03
fd2112	12	1	11.91	11.40	-4.28	0.00	1	11.40	3600.00
fd2114	2	1	1.22	1.22*	0.00	0.00	1	1.22*	0.00
fd3112	4	1	4.01	4.01*	0.00	0.00	1	4.01*	0.03
fd4112	2	1	4.05	4.05*	0.00	0.00	1	4.05*	0.00
fd7110	4	1	2.82	2.82*	0.00	0.00	1	2.82*	0.03
fd1123	12	1	14.56	14.56	0.00	0.01	1	14.56	3600.00
fd5121	2	1	1.98	1.98*	0.00	0.00	1	1.98*	0.00

Table 6.2: Results for the *Programmed Deliveries* instances

Data		**IVND**					**CPLEX**		
Inst.	Size	$\#V_f$	D_0	D_f	$\text{gap}_c(\%)$	*t(s)*	$\#V_c$	D_c	*t(s)*
pd1090	2	1	7.23	7.23*	0.00	0.00	1	7.23*	0.02
pd0130	40	2	71.42	53.97	-24.43	0.22	-	-	3600.00
pd1130	70	2	36.34	24.65	-32.11	3.22	-	-	3600.00
pd6130	6	1	11.33	11.33*	0.00	0.00	1	11.33*	2.99
pd1180	62	1	31.86	29.80	-6.46	0.74	-	-	3600.00
pd2180	70	1	21.10	19.13	-9.33	0.95	-	-	3600.00
pd4180	26	1	20.27	20.20	-0.34	0.05	-	-	3600.00
pd6180	2	1	3.69	3.69*	0.00	0.00	1	3.69*	0.03
pd2190	2	1	0.89	0.89*	0.00	0.00	1	0.89*	0.03
pd0220	4	1	1.25	1.25*	0.00	0.00	1	1.25*	0.08
pd1220	14	1	12.85	12.85	0.00	0.10	1	12.85	3600.00
pd2220	22	1	13.34	12.42	-6.89	0.03	1	12.42	3600.00
pd4220	14	1	11.24	11.24	0.00	0.01	1	11.24	3600.00
pd2130	140	2	38.55	31.26	-18.91	100.23	-	-	3600.00
pd4130	80	2	36.07	27.81	-22.90	10.34	-	-	3600.00

For the *programmed deliveries* instances the results are more diversified. The small instances could be solved to optimality by both CPLEX and IVND. Some solutions, although, could not be found by CPLEX within the time limit. IVND solved 14 instances under the maximum time limit of 10 seconds, but for the larger instance (*pd2130*), with 140 locations, it took 100 seconds, ten times the maximum time limit allowed. Again, this probably happened because of the small number of vehicles used compared to the large number of locations. Indeed, for only 2 vehicles, the *movement memory* has absolutely no effect, thus the larger time for this particular instance.

To conclude this analysis, it is important to note that those newly generated instances try to mimic real-world situations. This is to say that they simulate the spatial distribution of locations in a city, differently from the instances in the Li and Lim (2003) set, that were randomly created without any special distribution besides the clustered or non clustered type. This distribution factor has already been questioned by Ropke (2005), as in how realistic are the literature instances used to test and compare new heuristics, and how well would they behave when truly realistic data were to be considered. For example, according to Fleming, Griffis and Bell (2013), one concern that can have great impact on the algorithm's performance is the consideration of asymmetric distance and/or time between locations.

Those informations can be obtained from several databases, like *Google Maps* by

using Google's *Application Program Interface* (API)[2]. On the one hand, this provides accurate data and even real time estimation of traffic. On the other hand, to obtain such data can be time consuming, because of the overheads involved[3]. For usage in real time applications, such as the one described, a local database would be needed.

As far as the experiments here presented go, we could not test the new instances with those real-world distances and times, because of the hardness on obtaining them (particularly time consuming). Although, the distribution could be consistently simulated, which can be considered a good progress for testing the method.

[2]https://developers.google.com/maps/

[3]Our tests have shown the response time can reach the order of several minutes even for a small number of locations (e.g., 40). There is also a limitation on the number of locations to request the data from, so that the request is not considered forbidden (around 50 locations for the *free* version of Google's API).

7 CONCLUSION

This work has studied the *Pickup and Delivery Problem with Time Windows*, its applications in real-world cases, as well as methods for solving it efficiently. Regarding the applications of the PDPTW, it can be found on many real-world problems, such as product delivery, courier services and dial-a-ride problems. Moreover, it generalizes many other variations of VRP, and thus can also be applied to solve any of them. This is important, because reinforces the concept that research on this problem, and on methods for solving it, can help improve many areas of society, providing improvements on customer service, cost reduction and even environmental impacts.

Specifically to the real-world applications, a study has been conducted on the application of the model and solution of the problem to a real case in Porto Alegre. Informations and data about the case have been provided by a partner company of the software service market, also from Porto Alegre. New instances have been generated from the provided data.

Two methods have been proposed by this work to solve the PDPTW. An exact method, through the usage of a mathematical formulation in Mixed Integer Linear Programming form, applied on CPLEX solver. The second method is a metaheristic based on Iterated Local Search with an embedded Variable Neighborhood Descent method working as local search. Both methods are tested against the literature standard set of instances for the PDPTW, and the newly generated instances from the partner company.

The first hypothesis was that the problem could not be solved by the generic solver on reasonable time for practical uses. The evaluation confirmed the hypothesis for both sets of instances, assuring that a metaheuristic is necessary to solve the problem. The second hypothesis was how would a basic Iterated Local Search algorithm perform on such problem.

The proposed IVND, despite its simplicity, was able to produce good solutions on reasonable time for some instances of the standard set, and for most instances of the new set. When compared to some methods in the literature, it had a comparable quality and run time for the smaller instances. Further, the results on real-world instances showed that the algorithm could be applied to solve the studied case, being able to provide good solutions, most of the time optimal solutions, within the required time by the partner company.

It has been verified, however, that the method presents some drawbacks on its *best improvement* neighborhood strategy, because of the time needed to explore the neighbor-

hood space, especially for large instances. Moreover, the speedup techniques were able to help saving computation time in cases where large number of vehicles were used, but for the opposite case the algorithm had some bottlenecks during execution. Additionally, the fact that feasibility was kept at all moments during the search presented another time consuming restriction, specially for the perturbation procedure in the IVND.

For future research and improvements, it is possible to consider the case of allowing solutions to become infeasible, in order to explore more of the solution space, visiting regions that could not be reached by the current algorithm. The *best improvement* technique was not able to scale up with the instance size, thus experiments with *first improvement* techniques could be considered. At last, for real-world applications, it would be better to consider a PDPTW where time windows restrictions are *soft*, as opposed to the *hard* time windows studied, because in such situations there is no guarantee that it is possible to respect all time windows, especially inside big cities with known transportation issues.

Finally, future projects with the partner company are going to consider the implementation of the method, or a variation of it. This work was able to lead the research for such development, and point out how an ILS algorithm would behave in this problem. It is expected that the knowledge developed has also been able to contribute to the optimization area, specially on the transportation field.

REFERENCES

BATTITI, R.; PROTASI, M. Reactive search, a history-sensitive heuristic for max-sat. **Journal of Experimental Algorithmics (JEA)**, ACM, New York, NY, USA, v. 2, jan. 1997.

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.

BRUGGEN, L. Van der; LENSTRA, J. K.; SCHUUR, P. Variable-depth search for the single-vehicle pickup and delivery problem with time windows. **Transportation Science**, INFORMS, v. 27, n. 3, p. 298–311, 1993.

CORDEAU, J.-F.; LAPORTE, G.; ROPKE, S. Recent models and algorithms for one-to-one pickup and delivery problems. In: ____. **The Vehicle Routing Problem: Latest Advances and New Challenges**. Boston, MA: Springer US, 2008. p. 327–357.

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

DESROSIERS, J.; DUMAS, Y.; SOUMIS, F. A dynamic programming solution of the large-scale single-vehicle dial-a-ride problem with time windows. **American Journal of Mathematical and Management Sciences**, Taylor & Francis, v. 6, n. 3-4, p. 301–325, 1986.

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.

FLEMING, C. L.; GRIFFIS, S. E.; BELL, J. E. The effects of triangle inequality on the vehicle routing problem. **European Journal of Operational Research**, v. 224, n. 1, p. 1 – 7, 2013.

GEHRING, H.; HOMBERGER, J. A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows. In: CITESEER. **Proceedings of EUROGEN99**. [S.l.], 1999. v. 2, p. 57–64.

GRANDINETTI, L. et al. The multi-objective multi-vehicle pickup and delivery problem with time windows. **Procedia - Social and Behavioral Sciences**, v. 111, p. 203 – 212, 2014. ISSN 1877-0428.

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

KING, G. F.; MAST, T. M. **Excess travel: causes, extent, and consequences**. [S.l.]: Transportation Research Board, 1987.

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

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.

LÓPEZ-IBÁNEZ, M. et al. **The irace package, iterated race for automatic algorithm configuration**. [S.l.], 2011.

LOURENCO, H. R. Job-shop scheduling: Computational study of local search and large-step optimization methods. **European Journal of Operational Research**, Elsevier, v. 83, n. 2, p. 347–364, 1995.

LOURENÇO, H. R.; MARTIN, O. C.; STÜTZLE, T. Iterated local search: Framework and applications. In: ____. **Handbook of Metaheuristics**. Boston, MA: Springer US, 2010. p. 363–397.

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.

MLADENOVIĆ, N.; HANSEN, P. Variable neighborhood search. **Computers & Operations Research**, Elsevier, v. 24, n. 11, p. 1097–1100, 1997.

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.

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.

PENNA, P. H. V.; SUBRAMANIAN, A.; OCHI, L. S. An iterated local search heuristic for the heterogeneous fleet vehicle routing problem. **Journal of Heuristics**, Springer, v. 19, n. 2, p. 201–232, 2013.

RIBAS, S. et al. An algorithm based on iterated local search and set partitioning for the vehicle routing problem with time windows. **Strathprints Institutional Repository**, p. 145, 2011.

RODRIGUES, J.-P.; COMTOIS, C.; SLACK, B. **The geography of transport systems**. [S.l.]: Routledge, 2013.

ROPKE, S. **Heuristic and exact algorithms for vehicle routing problems**. Thesis (PhD) — University of Copenhagen, 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.

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.

SAVELSBERGH, M. W.; SOL, M. The general pickup and delivery problem. **Transportation science**, INFORMS, v. 29, n. 1, p. 17–29, 1995.

SIGURD, M.; PISINGER, D.; SIG, M. Scheduling transportation of live animals to avoid the spread of diseases. **Transportation Science**, INFORMS, v. 38, n. 2, p. 197–209, 2004.

SILVESTRIN, P. V. **An efficient heuristic for the Multi-compartment Vehicle Routing Problem**. Dissertation (Master) — Universidade Federal do Rio Grande do Sul, Porto Alegre, 2016. 69 pages. MSc in Computer Science.

SINTEF. **Li & Lim Benchmark Instances**. 2008. Accessed 17-October-2016. Available from Internet: <https://www.sintef.no/projectweb/top/pdptw/li-lim-benchmark/>.

SOL, M. **Column generation techniques for pickup and delivery problems**. Thesis (PhD) — Technische Universiteit Eindhoven, 1994. 121 pages. PhD in Computer Sciente.

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.

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

TOTH, P.; VIGO, D. The vehicle routing problem. In: TOTH, P.; VIGO, D. (Ed.). Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2001. chp. An Overview of Vehicle Routing Problems, p. 1–26.

TREGO, T.; MURRAY, D. An analysis of the operational costs of trucking. In: **Transportation Research Board 2010 Annual Meetings. Washington, DC**. [S.l.: s.n.], 2010. v. 18, p. 20.

VELAZQUEZ, L. et al. Sustainable transportation strategies for decoupling road vehicle transport and carbon dioxide emissions. **Management of Environmental Quality: An International Journal**, Emerald Group Publishing Limited, v. 26, n. 3, p. 373–388, 2015.

APPENDIX A — RESULT TABLES FOR THE STANDARD INSTANCES

All the tables presented in this appendix have been generated by running the proposed IVND to solve the PDPTW. The instances considered are all from the standard set of instances, proposed by Li and Lim (2003), available in SINTEF's website (SINTEF, 2008), as well as the best known solutions, and its references.

For each instance, a number of IVND runs has been executed and the average results are reported, one instance by table line. For instances of size 100 and 200, 10 runs were allowed. Instances of size 400, 600, 800 and 1000, were allowed 5 runs (except for some of the 800 and 1000 that could only run one time - these are marked in their respective tables).

Column *Instance* gives the instance identifier. The three columns of *BKS* refer to the best known solution for the given instance. Column #V_b gives the number of vehicles of the BKS, column *Cost* gives the total cost (or distance) of the BKS, and column *Ref.* gives the reference that first discovered the result. The complete references are available in (SINTEF, 2008).

The next four columns refer to the initial solution generated by IVND. Column #V_i gives the average number of vehicles in the initial solution, column $\text{gap}_\text{c}(\%)$ gives the deviation of the initial solution s cost to the BKS, calculated as $\text{gap}_\text{c}(s) = (s_c - s^*_c)/s^*_c$, where s^* is the BKS, and s_c is the accumulated cost of all routes in solution s. Column $\text{gap}_\text{f}(\%)$ reports the deviation of the full solution value to the BKS. The full solution value of s is computed by $OF(s) = \omega * |s| + \sum_{r \in S} d(r)$, where ω is defined based on the number of locations L in the instance, $\omega = L100$. The deviation is given by $\text{gap}_\text{f}(s) = (s_f - s^*_f)/s^*_f$, where s^* is the BKS, and s_f is the full solution value of s. At last, column *t(s)* gives the average CPU time in seconds taken to reach the solution.

The last six columns refer to the final solution obtained by IVND. Columns #V_f, *Cost*, $\text{gap}_\text{c}(\%)$, $\text{gap}_\text{f}(\%)$ and *t(s)*, are all similar to the columns that have already been described. Column $\text{gap}_\text{v}(\%)$ gives the deviation of the number of vehicles from the BKS. Column $\sigma(\%)$ gives the standard deviation in percentage from the solutions found during all the runs performed for the given instance. That is to say that the full solution value (considering the vehicles) has varied $\pm\ x$ percent.

The last line presents the general average results for all instances in the set. Note that the results were not separated here between type 1 and type 2, however all the information needed to compute it is available in the tables.

Table A.1: Average results for 100 locations instances

	BKS			Initial Solution				Final Solution						
Instance	$\#V_b$	Cost	Ref.	$\#V_i$	$gap_c(\%)$	$gap_f(\%)$	t(s)	$\#V_f$	Cost	$gap_v(\%)$	$gap_c(\%)$	$gap_f(\%)$	$\sigma(\%)$	t(s)
LC101	10	828.94	LL	10.00	0.00	0.00	0.01	10.00	828.94	0.00	0.00	0.00	± 0.00	2.91
LC102	10	828.94	LL	12.00	54.61	20.28	0.01	10.00	828.94	0.00	0.00	0.00	± 0.00	4.58
LC103	9	1035.35	BVH	10.00	16.91	11.18	0.01	9.30	996.57	3.33	-3.75	3.25	± 1.59	5.34
LC104	9	860.01	SAM	10.00	43.08	11.41	0.01	9.10	878.21	1.11	2.12	1.12	± 1.08	8.44
LC105	10	828.94	LL	10.00	4.55	0.04	0.01	10.00	828.94	0.00	0.00	0.00	± 0.00	3.35
LC106	10	828.94	LL	11.00	5.83	9.97	0.01	10.00	828.94	0.00	0.00	0.00	± 0.00	3.41
LC107	10	828.94	LL	10.00	2.90	0.02	0.01	10.00	832.22	0.00	0.40	0.00	± 0.00	4.53
LC108	10	826.44	LL	10.00	3.55	0.03	0.01	10.00	827.61	0.00	0.14	0.00	± 0.00	4.86
LC109	9	1000.60	BVH	10.00	3.87	11.03	0.02	10.00	853.78	11.11	-14.67	10.83	± 0.00	6.26
LC201	3	591.56	LL	3.00	-0.00	0.00	0.01	3.00	591.56	0.00	-0.00	0.00	± 0.00	41.06
LC202	3	591.56	LL	4.00	70.27	34.05	0.02	3.00	591.56	0.00	-0.00	0.00	± 0.00	56.73
LC203	3	591.17	SAM	4.00	43.98	33.54	0.03	3.00	591.17	0.00	0.00	0.00	± 0.00	39.61
LC204	3	590.60	SAM	4.00	39.82	33.46	0.07	3.00	590.83	0.00	0.04	0.00	± 0.00	75.98
LC205	3	588.88	LL	4.00	171.50	35.99	0.01	3.00	588.88	0.00	0.00	0.00	± 0.00	57.01
LC206	3	588.49	LL	4.00	39.85	33.46	0.03	3.00	588.49	0.00	-0.00	0.00	± 0.00	85.72
LC207	3	588.29	LL	4.00	90.60	34.43	0.02	3.00	588.29	0.00	0.00	0.00	± 0.00	32.55
LC208	3	588.32	LL	4.00	88.35	34.39	0.02	3.00	588.32	0.00	-0.00	0.00	± 0.00	72.13
LR101	19	1650.80	LL	23.00	19.58	21.04	0.00	19.00	1650.80	0.00	-0.00	0.00	± 0.00	1.75
LR102	17	1487.57	LL	20.00	29.31	17.75	0.01	17.00	1491.58	0.00	0.27	0.00	± 0.00	5.31
LR103	13	1292.68	LL	18.00	41.27	38.49	0.01	13.00	1292.77	0.00	0.01	0.00	± 0.00	3.60
LR104	9	1013.39	LL	13.00	38.53	44.38	0.01	9.10	1018.81	1.11	0.54	1.10	± 1.09	4.93
LR105	14	1377.11	LL	18.00	23.84	28.53	0.00	14.00	1377.30	0.00	0.01	0.00	± 0.00	3.26
LR106	12	1252.62	LL	15.00	25.25	25.00	0.01	12.00	1258.70	0.00	0.49	0.01	± 0.00	2.69
LR107	10	1111.31	LL	16.00	49.51	59.88	0.01	10.00	1120.79	0.00	0.85	0.01	± 0.01	3.74
LR108	9	968.97	LL	12.00	38.72	33.39	0.01	9.00	968.97	0.00	0.00	0.00	± 0.00	4.07
LR109	11	1208.96	SAM	16.00	33.39	45.32	0.01	11.40	1234.83	3.64	2.14	3.62	± 1.43	3.28
LR110	10	1159.35	LL	13.00	20.87	29.90	0.01	10.60	1170.64	6.00	0.97	5.94	± 2.07	5.84
LR111	10	1108.90	LL	16.00	51.11	59.90	0.01	10.10	1117.28	1.00	0.76	1.00	± 0.98	4.45
LR112	9	1003.77	LL	13.00	41.77	44.41	0.01	9.00	1004.07	0.00	0.03	0.00	± 0.00	5.69
LR201	4	1253.23	SAM	6.00	52.10	50.06	0.02	4.00	1261.95	0.00	0.70	0.02	± 0.02	9.52
LR202	3	1197.67	LL	6.00	77.35	99.13	0.03	3.00	1201.82	0.00	0.35	0.01	± 0.01	4.75
LR203	3	949.40	LL	4.00	61.58	34.20	0.04	3.00	951.95	0.00	0.27	0.01	± 0.01	12.73
LR204	2	849.05	LL	4.00	102.14	100.09	0.06	2.00	852.47	0.00	0.40	0.02	± 0.02	8.38
LR205	3	1054.02	LL	5.00	69.67	66.77	0.02	3.00	1054.04	0.00	0.00	0.00	± 0.00	4.48
LR206	3	931.63	LL	4.00	64.18	34.26	0.04	3.00	931.63	0.00	-0.00	0.00	± 0.00	8.94
LR207	2	903.06	LL	3.00	48.80	49.95	0.08	2.70	994.67	35.00	10.14	33.93	± 5.50	145.37
LR208	2	734.85	LL	3.00	65.06	50.53	0.08	2.00	739.35	0.00	0.61	0.02	± 0.01	15.47
LR209	3	930.59	SAM	4.00	52.67	33.92	0.04	3.00	971.34	0.00	4.38	0.13	± 0.05	15.13
LR210	3	964.22	LL	5.00	102.55	67.78	0.03	3.20	1032.65	6.67	7.10	6.68	± 4.12	14.10
LR211	2	911.52	SAM	3.00	65.64	50.68	0.05	2.90	907.79	45.00	-0.41	43.02	± 3.34	49.69
LRC101	14	1708.80	LL	19.00	21.46	35.54	0.00	14.20	1718.86	1.43	0.59	1.42	± 0.93	2.25
LRC102	12	1558.07	SAM	19.00	42.46	58.13	0.01	12.20	1564.27	1.67	0.40	1.65	± 1.08	3.49
LRC103	11	1258.74	LL	14.00	31.70	27.32	0.01	11.00	1258.74	0.00	0.00	0.00	± 0.00	3.61
LRC104	10	1128.40	LL	13.00	39.24	30.10	0.01	10.00	1132.92	0.00	0.40	0.00	± 0.00	4.74
LRC105	13	1637.62	LL	18.00	30.86	38.37	0.01	14.50	1710.63	11.54	4.46	11.45	± 1.53	4.04
LRC106	11	1424.73	SAM	16.00	32.70	45.29	0.01	11.00	1428.20	0.00	0.24	0.00	± 0.00	4.00
LRC107	11	1230.14	SAM	15.00	46.57	36.48	0.01	11.00	1230.14	0.00	-0.00	0.00	± 0.00	3.33
LRC108	10	1147.43	SAM	13.00	29.22	29.99	0.01	10.00	1151.87	0.00	0.39	0.00	± 0.00	3.56
LRC201	4	1406.94	SAM	6.00	85.25	51.20	0.01	4.00	1466.86	0.00	4.26	0.14	± 0.01	5.96
LRC202	3	1374.27	LL	4.00	40.36	33.64	0.02	3.20	1378.49	6.67	0.31	6.39	± 4.00	44.09
LRC203	3	1089.07	LL	5.00	100.45	67.85	0.03	3.20	1142.28	6.67	4.89	6.60	± 4.04	13.99
LRC204	3	818.66	SAM	4.00	91.69	34.88	0.08	3.00	826.11	0.00	0.91	0.02	± 0.02	29.69
LRC205	4	1302.20	LL	5.00	94.97	27.21	0.02	4.00	1360.39	0.00	4.47	0.14	± 0.03	5.52
LRC206	3	1159.03	SAM	5.00	90.44	67.55	0.02	3.10	1189.51	3.33	2.63	3.31	± 3.11	8.69
LRC207	3	1062.05	SAM	5.00	95.10	67.64	0.03	3.10	1090.57	3.33	2.69	3.31	± 3.15	8.25
LRC208	3	852.76	LL	4.00	121.29	35.76	0.04	3.00	882.67	0.00	3.51	0.10	± 0.08	13.05
Average	7.18	1036.78	-	9.45	50.86	37.06	0.02	7.30	1045.75	2.65	0.79	2.59	± 0.70	17.86

Table A.2: Average results for 200 locations instances

	BKS			Initial Solution				Final Solution						
Instance	$\#V_b$	Cost	Ref.	$\#V_i$	gap_c(%)	gap_f(%)	t(s)	$\#V_f$	Cost	gap_v(%)	gap_c(%)	gap_f(%)	σ(%)	t(s)
LC1_2_1	20	2704.57	LL	20.00	6.98	0.05	0.03	20.00	2705.94	0.00	0.05	0.00	± 0.00	5.74
LC1_2_2	19	2764.56	LL	20.00	48.58	5.58	0.04	19.00	2814.40	0.00	1.80	0.01	± 0.00	16.01
LC1_2_3	17	3127.78	H	20.00	35.00	17.81	0.05	18.00	2773.30	5.88	-11.33	5.73	± 0.00	38.96
LC1_2_4	17	2693.41	BVH	19.00	55.00	12.10	0.09	17.00	2709.44	0.00	0.60	0.00	± 0.00	90.02
LC1_2_5	20	2702.05	LL	20.00	15.69	0.11	0.03	20.00	2765.41	0.00	2.34	0.02	± 0.00	7.64
LC1_2_6	20	2701.04	LL	21.00	18.62	5.09	0.04	20.00	2701.04	0.00	0.00	0.00	± 0.00	7.15
LC1_2_7	20	2701.04	LL	20.00	21.21	0.14	0.04	20.00	2702.13	0.00	0.04	0.00	± 0.00	14.48
LC1_2_8	19	3379.97	EOE	22.00	34.73	15.96	0.04	20.00	2707.47	5.26	-19.90	5.04	± 0.00	37.21
LC1_2_9	18	2724.24	LL	21.00	82.18	17.16	0.05	18.00	2724.48	0.00	0.01	0.00	± 0.00	26.96
LC1_2_10	17	2942.13	EOE	20.00	55.00	17.97	0.07	18.00	2769.30	5.88	-5.87	5.78	± 0.00	55.15
LC2_2_1	6	1931.44	LL	7.00	12.45	16.60	0.07	6.00	1931.44	0.00	0.00	0.00	± 0.00	55.95
LC2_2_2	6	1881.40	LL	8.00	53.21	33.64	0.12	6.00	1881.40	0.00	0.00	0.00	± 0.00	42.88
LC2_2_3	6	1844.33	SAM	8.00	96.07	34.28	0.16	6.00	1847.60	0.00	0.18	0.00	± 0.00	52.71
LC2_2_4	6	1767.12	LL	8.00	85.16	34.09	0.33	6.10	1807.47	1.67	2.28	1.68	± 1.63	176.02
LC2_2_5	6	1891.21	LL	7.00	40.11	17.03	0.09	6.00	1891.21	0.00	-0.00	0.00	± 0.00	58.55
LC2_2_6	6	1857.78	SAM	7.00	72.73	17.52	0.14	6.00	1859.02	0.00	0.07	0.00	± 0.00	61.59
LC2_2_7	6	1850.13	SAM	7.00	90.28	17.78	0.13	6.00	1857.30	0.00	0.39	0.01	± 0.00	72.19
LC2_2_8	6	1824.34	LL	7.00	69.13	17.45	0.20	6.00	1824.51	0.00	0.01	0.00	± 0.00	104.16
LC2_2_9	6	1854.21	SAM	8.00	79.93	34.04	0.17	6.00	1887.85	0.00	1.81	0.03	± 0.02	66.03
LC2_2_10	6	1817.45	LL	7.00	63.56	17.37	0.26	6.00	1817.93	0.00	0.03	0.00	± 0.00	83.37
LR1_2_1	20	4819.12	LL	26.00	33.49	30.04	0.03	20.50	4937.42	2.50	2.45	2.50	± 0.81	11.80
LR1_2_2	17	4621.21	RP	23.00	35.52	35.30	0.04	18.30	4226.63	7.65	-8.54	7.43	± 0.82	52.44
LR1_2_3	14	4402.38	CLS	19.00	25.00	35.55	0.06	15.80	3711.44	12.86	-15.69	12.42	± 0.84	75.17
LR1_2_4	10	3030.03	H	15.00	55.72	50.09	0.14	11.60	2971.32	16.00	-1.94	15.73	± 1.38	137.74
LR1_2_5	16	4760.18	BVH	23.00	24.95	43.47	0.03	17.10	4421.78	6.88	-7.11	6.67	± 0.58	33.48
LR1_2_6	13	4800.94	CLS	19.00	18.23	45.65	0.06	14.80	4184.82	13.85	-12.83	13.36	± 1.65	81.43
LR1_2_7	12	3550.61	RP	17.00	45.97	41.73	0.06	13.90	3341.93	15.83	-5.88	15.52	± 1.65	103.23
LR1_2_8	9	2766.42	EOE	12.00	38.94	33.42	0.16	10.10	2741.74	12.22	-0.89	12.02	± 0.98	90.82
LR1_2_9	13	5050.75	CLS	17.00	10.22	30.38	0.04	15.20	4266.51	16.92	-15.53	16.30	± 0.86	58.77
LR1_2_10	11	3664.08	H	15.00	31.70	36.29	0.06	12.30	3635.88	11.82	-0.77	11.61	± 1.22	62.95
LR2_2_1	5	4073.10	SAM	6.00	56.41	21.42	0.11	5.00	4266.22	0.00	4.74	0.19	± 0.01	31.54
LR2_2_2	4	3796.00	SAM	6.00	68.78	50.85	0.21	5.40	4117.19	35.00	8.46	33.80	± 3.96	166.24
LR2_2_3	4	3098.36	RP	6.00	116.17	52.47	0.35	4.60	3433.70	15.00	10.82	14.84	± 3.41	292.59
LR2_2_4	3	2486.00	H	4.00	96.80	35.86	0.68	4.00	2393.46	33.33	-3.72	31.86	± 0.04	499.03
LR2_2_5	4	3438.39	SAM	6.00	99.75	52.05	0.18	4.50	3615.46	12.50	5.15	12.20	± 3.58	103.06
LR2_2_6	3	4518.93	H	5.00	46.82	65.28	0.22	4.10	3289.70	36.67	-27.20	32.19	± 2.37	195.92
LR2_2_7	3	3098.35	H	4.00	72.33	35.25	0.41	4.00	2838.20	33.33	-8.40	31.28	± 0.05	360.05
LR2_2_8	2	2455.87	H	4.00	53.14	97.29	1.17	3.00	2568.10	50.00	4.57	47.37	± 0.04	448.99
LR2_2_9	3	3927.13	CLS	5.00	34.40	64.68	0.21	4.00	3413.90	33.33	-13.07	30.48	± 0.09	101.73
LR2_2_10	3	3274.96	CLS	4.00	74.61	35.47	0.25	3.60	3429.74	20.00	4.73	19.21	± 4.24	132.19
LRC1_2_1	19	3606.06	SAM	23.00	54.42	21.37	0.04	19.00	3722.21	0.00	3.22	0.03	± 0.01	41.24
LRC1_2_2	15	3671.02	EOE	21.00	34.95	39.94	0.04	17.00	3466.20	13.33	-5.58	13.10	± 0.01	90.27
LRC1_2_3	13	3154.92	CLS	16.00	40.12	23.28	0.06	14.10	3333.52	8.46	5.66	8.43	± 0.70	96.63
LRC1_2_4	10	2631.82	RP	13.00	48.93	30.25	0.20	11.20	2629.96	12.00	-0.07	11.84	± 1.18	95.62
LRC1_2_5	16	3715.81	BVH	21.00	51.56	31.48	0.04	16.40	3944.94	2.50	6.17	2.54	± 0.99	48.04
LRC1_2_6	16	3572.16	EOE	23.00	59.07	43.92	0.04	17.00	3459.91	6.25	-3.14	6.15	± 0.01	70.58
LRC1_2_7	14	3666.34	K	19.00	44.98	35.83	0.05	15.90	3503.27	13.57	-4.45	13.34	± 1.12	95.80
LRC1_2_8	13	3161.06	H	19.00	61.93	46.34	0.05	14.00	3122.17	7.69	-1.23	7.59	± 0.00	84.56
LRC1_2_9	13	3157.34	EOE	18.00	54.25	38.65	0.05	14.90	3161.21	14.62	0.12	14.44	± 0.66	88.36
LRC1_2_10	12	2928.90	EOE	17.00	46.62	41.73	0.07	13.00	2929.88	8.33	0.03	8.23	± 0.01	103.62
LRC2_2_1	6	3595.18	K	8.00	43.34	33.62	0.10	7.00	3217.20	16.67	-10.51	15.88	± 0.05	39.82
LRC2_2_2	5	3158.25	H	7.00	85.51	41.39	0.17	6.10	3103.26	22.00	-1.74	21.27	± 1.59	167.74
LRC2_2_3	4	2881.99	H	6.00	90.08	51.39	0.24	5.70	2666.42	42.50	-7.48	40.76	± 2.63	240.43
LRC2_2_4	3	2849.43	H	5.00	39.71	65.44	0.54	4.00	2635.52	33.33	-7.51	31.48	± 0.03	620.53
LRC2_2_5	5	2776.93	BVH	8.00	109.49	61.34	0.16	5.20	2917.10	4.00	5.05	4.03	± 2.52	112.90
LRC2_2_6	5	2707.96	SAM	6.00	92.04	21.90	0.15	5.00	2748.69	0.00	1.50	0.04	± 0.00	62.51
LRC2_2_7	4	3018.05	EOE	6.00	75.44	50.92	0.22	5.10	2690.28	27.50	-10.86	26.11	± 1.92	197.27
LRC2_2_8	4	2399.89	EOE	5.00	93.51	27.00	0.32	4.00	2774.15	0.00	15.59	0.45	± 0.05	82.47
LRC2_2_9	4	2208.49	RP	5.00	110.95	27.31	0.24	4.20	2597.32	5.00	17.61	5.34	± 3.12	204.17
LRC2_2_10	3	2437.88	H	5.00	89.93	67.57	0.37	4.00	2261.90	33.33	-7.22	31.75	± 0.06	214.35
Average	10.00	3064.87	-	12.73	56.69	33.75	0.16	10.75	2977.81	11.26	-1.88	10.80	± 0.78	114.48

Table A.3: Average results for 400 locations instances

	BKS			Initial Solution				Final Solution						
Instance	$\#V_b$	Cost	Ref.	$\#V_i$	$\text{gap}_\text{c}(\%)$	$\text{gap}_\text{f}(\%)$	t(s)	$\#V_f$	Cost	$\text{gap}_\text{v}(\%)$	$\text{gap}_\text{c}(\%)$	$\text{gap}_\text{f}(\%)$	$\sigma(\%)$	t(s)
LC1_4_1	40	7152.06	SAM	40.00	13.91	0.06	0.17	40.00	7152.06	0.00	0.00	0.00	± 0.00	14.20
LC1_4_2	38	8007.79	Q	45.00	43.37	18.55	0.22	40.00	7238.68	5.26	-9.60	5.19	± 0.00	94.64
LC1_4_3	32	9252.95	CLS	40.00	26.53	25.01	0.29	37.20	7428.31	16.25	-19.72	15.99	± 0.25	195.99
LC1_4_4	30	6451.68	LL	36.00	48.68	20.15	0.45	31.40	6915.64	4.67	7.19	4.68	± 0.36	329.94
LC1_4_5	40	7150.00	SAM	40.00	18.98	0.08	0.20	40.00	7151.31	0.00	0.02	0.00	± 0.00	23.28
LC1_4_6	40	7154.02	LL	42.00	35.22	5.13	0.18	40.00	7197.02	0.00	0.60	0.00	± 0.00	22.67
LC1_4_7	40	7149.43	SAM	41.00	27.26	2.61	0.20	40.00	7559.94	0.00	5.74	0.03	± 0.00	19.41
LC1_4_8	39	7111.16	LL	42.00	41.32	7.84	0.20	39.00	7313.15	0.00	2.84	0.01	± 0.00	42.19
LC1_4_9	36	7451.20	Q	42.00	62.27	16.90	0.28	37.00	7478.73	2.78	0.37	2.77	± 0.00	82.08
LC1_4_10	35	7325.01	CLS	42.00	66.01	20.24	0.28	36.00	7229.33	2.86	-1.31	2.84	± 0.00	104.94
LC2_4_1	12	4116.33	LL	14.00	58.97	17.03	0.40	12.00	4116.33	0.00	0.00	0.00	± 0.00	84.57
LC2_4_2	12	4144.29	SAM	15.00	91.41	25.57	0.64	12.60	4231.53	5.00	2.11	4.98	± 0.91	202.60
LC2_4_3	12	4418.88	CLS	15.00	117.97	25.85	0.94	13.00	4102.07	8.33	-7.17	8.19	± 0.00	447.87
LC2_4_4	12	3743.95	LL	14.00	106.34	17.36	1.91	13.00	4436.96	8.33	18.51	8.41	± 0.01	1631.23
LC2_4_5	12	4030.63	SAM	16.00	111.06	33.98	0.55	12.00	4035.93	0.00	0.13	0.00	± 0.00	158.39
LC2_4_6	12	3900.29	SAM	14.00	82.50	17.20	0.66	12.00	3912.41	0.00	0.31	0.00	± 0.00	194.77
LC2_4_7	12	3962.51	BVH	15.00	113.30	25.72	0.77	12.00	3977.15	0.00	0.37	0.00	± 0.00	273.77
LC2_4_8	12	3844.45	LL	14.00	93.96	17.28	0.78	12.00	3851.50	0.00	0.18	0.00	± 0.00	346.39
LC2_4_9	12	4188.93	RP	15.00	95.44	25.61	0.88	13.00	4432.44	8.33	5.81	8.31	± 0.00	436.21
LC2_4_10	12	3828.44	BVH	14.00	105.03	17.37	0.94	12.00	3841.93	0.00	0.35	0.00	± 0.00	409.29
LR1_4_1	40	10639.75	SAM	50.00	48.29	25.15	0.17	40.00	10988.31	0.00	3.28	0.02	± 0.00	74.42
LR1_4_2	31	9968.19	BN	41.00	44.78	32.36	0.27	34.40	9887.53	10.97	-0.81	10.87	± 0.33	215.73
LR1_4_3	22	9291.25	CLS	33.00	30.08	49.79	0.37	27.20	8317.27	23.64	-10.48	23.28	± 0.34	459.79
LR1_4_4	16	6710.99	CLS	24.00	53.90	50.04	0.68	19.40	6675.91	21.25	-0.52	21.02	± 0.59	578.37
LR1_4_5	28	11374.06	CLS	39.00	23.51	39.13	0.20	33.20	10060.52	18.57	-11.55	18.27	± 0.53	85.79
LR1_4_6	24	9891.02	CLS	34.00	35.99	41.61	0.30	28.80	9221.67	20.00	-6.77	19.73	± 0.61	218.01
LR1_4_7	18	8999.97	CLS	28.00	38.39	55.34	0.40	24.40	7715.49	35.56	-14.27	34.94	± 0.98	417.47
LR1_4_8	14	5944.55	H	20.00	53.34	42.97	0.78	17.60	6393.23	25.71	7.55	25.52	± 0.65	455.75
LR1_4_9	24	9862.65	CLS	33.00	35.50	37.48	0.27	28.40	9529.36	18.33	-3.38	18.11	± 0.40	237.94
LR1_4_10	20	8364.66	CLS	27.00	41.62	35.07	0.34	23.40	8063.67	17.00	-3.60	16.79	± 0.49	331.12
LR2_4_1	8	9726.88	BVH	12.00	71.55	50.64	0.69	9.20	9761.06	15.00	0.35	14.57	± 1.01	452.44
LR2_4_2	7	9440.93	EOE	10.00	56.49	43.30	1.16	9.00	8242.79	28.57	-12.69	27.23	± 1.62	864.93
LR2_4_3	5	10658.64	CLS	9.00	35.02	77.72	2.16	8.80	7288.25	76.00	-31.62	70.55	± 1.08	2442.19
LR2_4_4	4	6404.87	H	7.00	62.13	74.50	8.21	6.00	5905.46	50.00	-7.80	47.78	± 0.01	6432.21
LR2_4_5	6	10084.44	Q	9.00	41.90	49.67	1.12	8.20	8674.45	36.67	-13.98	34.62	± 1.13	923.15
LR2_4_6	5	9044.03	EOE	8.00	51.17	59.62	1.64	7.60	7826.57	52.00	-13.46	49.17	± 1.45	2250.79
LR2_4_7	5	6729.67	EOE	7.00	92.50	41.71	3.04	6.00	6710.38	20.00	-0.29	19.34	± 0.03	5600.59
LR2_4_8	4	5330.41	H	6.00	76.39	50.85	10.54	5.00	5723.83	25.00	7.38	24.43	± 0.02	7714.39
LR2_4_9	6	7930.55	Q	10.00	80.75	67.12	1.19	8.00	7162.81	33.33	-9.68	31.96	± 0.03	1187.85
LR2_4_10	5	7846.99	EOE	9.00	86.46	80.24	1.61	7.20	7401.74	44.00	-5.67	42.12	± 1.27	1923.58
LRC1_4_1	36	9124.52	CLS	45.00	44.72	25.12	0.18	37.00	9200.78	2.78	0.84	2.77	± 0.00	140.46
LRC1_4_2	31	8346.06	RP	41.00	58.62	32.43	0.26	34.40	8106.64	10.97	-2.87	10.88	± 0.33	295.90
LRC1_4_3	24	7856.72	CLS	31.00	40.12	29.26	0.41	28.40	7541.69	18.33	-4.01	18.15	± 0.40	378.90
LRC1_4_4	19	5806.20	DK	26.00	63.98	37.05	0.62	21.20	6349.16	11.58	9.35	11.56	± 0.44	490.42
LRC1_4_5	32	8867.38	DK	43.00	46.44	34.46	0.20	35.40	8905.91	10.62	0.43	10.55	± 0.33	116.14
LRC1_4_6	30	8396.08	DK	39.00	48.19	30.13	0.22	33.60	8230.63	12.00	-1.97	11.90	± 0.34	171.78
LRC1_4_7	28	8037.87	DK	38.00	45.61	35.78	0.25	31.60	8090.20	12.86	0.65	12.77	± 0.36	201.12
LRC1_4_8	26	8173.63	CLS	35.00	43.73	34.69	0.26	30.20	7778.33	16.15	-4.84	15.99	± 0.31	237.12
LRC1_4_9	25	8181.32	CLS	35.00	44.62	40.04	0.25	28.60	8059.21	14.40	-1.49	14.27	± 0.40	252.92
LRC1_4_10	23	7222.97	CLS	32.00	51.13	39.22	0.30	26.20	7237.41	13.91	0.20	13.81	± 0.67	375.86
LRC2_4_1	12	7454.14	Q	16.00	95.62	34.29	0.54	14.20	6874.34	18.33	-7.78	17.93	± 0.66	310.67
LRC2_4_2	10	7424.72	Q	14.00	84.56	40.81	0.94	13.40	6737.12	34.00	-9.26	33.21	± 1.39	946.27
LRC2_4_3	8	6576.48	CLS	13.00	102.92	63.31	2.13	11.40	5955.03	42.50	-9.45	41.45	± 1.00	2495.38
LRC2_4_4	5	5322.43	RP	9.00	82.10	80.05	6.80	8.00	4404.63	60.00	-17.24	58.00	± 0.00	9289.78
LRC2_4_5	10	7462.66	CLS	15.00	99.80	50.91	0.66	11.60	6448.38	16.00	-13.59	15.46	± 0.98	609.74
LRC2_4_6	9	6337.08	Q	13.00	112.31	45.62	0.77	11.60	6179.41	28.89	-2.49	28.35	± 0.98	796.49
LRC2_4_7	8	6322.35	H	12.00	103.69	51.04	0.99	9.20	6123.99	15.00	-3.14	14.65	± 1.01	915.52
LRC2_4_8	7	5814.93	Q	11.00	121.86	58.46	1.21	8.60	5624.92	22.86	-3.27	22.33	± 1.33	990.13
LRC2_4_9	6	6666.94	H	10.00	77.50	66.96	1.26	9.20	5553.57	53.33	-16.70	51.44	± 1.02	1709.97
LRC2_4_10	6	5585.18	Q	9.00	108.65	51.33	1.53	7.60	5192.02	26.67	-7.04	25.90	± 1.48	2010.20
Average	18.95	7226.80	-	24.32	64.92	37.08	1.10	21.12	6895.77	17.91	-3.58	17.38	± 0.46	1011.93

Table A.4: Average results for 600 locations instances

	BKS			Initial Solution				Final Solution						
Instance	$\#V_b$	Cost	Ref.	$\#V_i$	gap_c(%)	gap_f(%)	t(s)	$\#V_f$	Cost	gap_v(%)	gap_c(%)	gap_f(%)	σ(%)	t(s)
LC1_6_1	60	14095.60	LL	64.00	23.32	6.73	0.55	60.00	14098.06	0.00	0.02	0.00	± 0.00	48.30
LC1_6_2	57	15048.16	CLS	75.00	59.58	31.70	0.68	59.00	14237.48	3.51	-5.39	3.47	± 0.00	154.32
LC1_6_3	50	14683.43	RP	64.00	43.89	28.08	0.84	53.80	14366.28	7.60	-2.16	7.55	± 0.17	435.69
LC1_6_4	47	13648.03	RP	56.00	45.07	19.27	1.35	49.20	13535.37	4.68	-0.83	4.65	± 0.36	555.02
LC1_6_5	60	14086.30	LL	60.00	18.33	0.07	0.57	60.00	14197.37	0.00	0.79	0.00	± 0.00	63.21
LC1_6_6	60	14090.79	RP	62.00	25.90	3.42	0.58	60.00	14235.14	0.00	1.02	0.00	± 0.00	59.67
LC1_6_7	60	14083.76	SAM	60.00	23.31	0.09	0.58	60.00	14439.80	0.00	2.53	0.01	± 0.00	92.71
LC1_6_8	58	14880.70	CLS	68.00	53.02	17.39	0.66	59.00	15035.42	1.72	1.04	1.72	± 0.00	96.63
LC1_6_9	54	14661.73	CLS	65.00	56.15	20.53	0.80	55.80	14767.32	3.33	0.72	3.32	± 0.17	189.26
LC1_6_10	52	15204.30	CLS	62.00	45.54	19.36	0.92	55.20	14811.99	6.15	-2.58	6.11	± 0.17	306.95
LC2_6_1	19	7977.98	SAM	21.00	73.45	10.96	1.27	19.00	7977.98	0.00	-0.00	0.00	± 0.00	179.07
LC2_6_2	18	9914.10	Q	24.00	92.14	33.87	1.59	19.00	8254.01	5.56	-16.74	5.35	± 0.00	451.38
LC2_6_3	17	8718.22	Q	21.00	53.02	23.78	2.46	18.00	7461.78	5.88	-14.41	5.71	± 0.00	702.78
LC2_6_4	17	7902.66	Q	20.00	94.26	18.24	5.12	18.00	8248.77	5.88	4.38	5.87	± 0.00	2241.85
LC2_6_5	19	8047.37	BVH	23.00	84.29	21.50	1.51	19.20	8102.45	1.05	0.68	1.05	± 0.49	485.96
LC2_6_6	18	8859.78	CLS	22.00	92.35	22.79	1.89	19.00	8217.35	5.56	-7.25	5.45	± 0.00	543.59
LC2_6_7	19	7997.96	Q	23.00	123.48	21.77	1.79	19.00	8074.56	0.00	0.96	0.01	± 0.00	720.52
LC2_6_8	18	7579.93	RP	21.00	109.02	17.31	2.16	18.00	7742.78	0.00	2.15	0.01	± 0.00	915.54
LC2_6_9	18	8864.29	Q	22.00	91.57	22.79	2.25	19.00	8207.71	5.56	-7.41	5.45	± 0.00	905.19
LC2_6_10	17	7965.41	Q	20.00	67.46	18.03	2.94	18.00	7502.65	5.88	-5.81	5.79	± 0.00	961.17
LR1_6_1	59	22824.32	CLS	67.00	47.38	13.78	0.66	60.20	23800.82	2.03	4.28	2.05	± 0.45	309.67
LR1_6_2	45	20246.18	RP	56.00	53.34	24.66	1.00	48.20	20880.43	7.11	3.13	7.08	± 0.36	557.91
LR1_6_3	37	17987.49	CLS	44.00	55.57	19.21	1.63	38.40	18910.09	3.78	5.13	3.79	± 0.30	1266.60
LR1_6_4	28	13191.79	CLS	34.00	66.85	21.78	4.10	29.00	14058.73	3.57	6.57	3.59	± 0.00	2259.25
LR1_6_5	38	22489.30	CLS	52.00	31.12	36.79	0.80	46.60	21766.33	22.63	-3.21	22.38	± 0.25	312.86
LR1_6_6	31	22188.80	CLS	46.00	32.95	48.21	1.08	40.40	19819.52	30.32	-10.68	29.84	± 0.46	939.20
LR1_6_7	24	18531.68	CLS	35.00	28.34	45.61	1.60	33.20	16059.14	38.33	-13.34	37.68	± 0.28	1375.38
LR1_6_8	18	12255.29	CLS	29.00	51.15	61.00	4.23	23.60	12256.80	31.11	0.01	30.76	± 0.48	2394.36
LR1_6_9	32	21117.75	CLS	45.00	32.60	40.54	0.86	39.40	20073.20	23.12	-4.95	22.82	± 0.61	450.73
LR1_6_10	26	19028.25	CLS	37.00	33.03	42.20	1.13	32.80	18011.07	26.15	-5.35	25.77	± 0.53	1124.68
LR2_6_1	11	21945.30	RP	16.00	52.08	45.67	2.35	12.80	19175.86	16.36	-12.62	15.43	± 1.35	1143.47
LR2_6_2	9	23903.03	CLS	15.00	40.00	65.54	3.62	13.00	15706.10	44.44	-34.29	41.11	± 0.00	2483.07
LR2_6_3	7	19183.41	CLS	11.00	45.83	56.65	6.60	11.00	15220.83	57.14	-20.66	53.74	± 0.01	8628.04
LR2_6_4	6	10819.45	RP	9.00	93.25	51.26	26.52	8.00	12093.77	33.33	11.78	32.70	± 0.04	18880.96
LR2_6_5	9	19411.73	CLS	13.00	58.88	44.95	3.34	12.00	17484.45	33.33	-9.93	31.83	± 0.02	2304.02
LR2_6_6	7	22570.45	CLS	12.00	39.59	69.80	4.78	11.00	15312.07	57.14	-32.16	52.59	± 0.00	5721.55
LR2_6_7	6	15526.81	CLS	11.00	90.26	83.62	10.17	10.00	14912.88	66.67	-3.95	63.75	± 0.03	14729.37
LR2_6_8	5	10950.90	RP	7.00	95.83	41.97	16.70	6.00	11879.64	20.00	8.48	19.59	± 0.00	23389.19
LR2_6_9	8	18799.36	RP	13.00	61.73	62.47	3.76	11.00	14981.90	37.50	-20.31	35.32	± 0.00	3093.35
LR2_6_10	7	17034.63	RP	10.00	55.38	43.35	5.29	9.60	15960.17	37.14	-6.31	35.45	± 1.16	3115.23
LRC1_6_1	52	18312.60	CLS	65.00	49.39	25.14	0.67	55.20	18162.99	6.15	-0.82	6.11	± 0.17	495.32
LRC1_6_2	43	17063.21	CLS	57.00	53.92	32.70	1.02	48.60	16651.81	13.02	-2.41	12.92	± 0.49	929.90
LRC1_6_3	36	14060.31	RP	43.00	71.45	19.78	1.93	38.00	15173.70	5.56	7.92	5.57	± 0.39	1376.26
LRC1_6_4	25	10950.52	RP	30.00	64.69	20.32	4.98	26.80	11986.21	7.20	9.46	7.22	± 0.35	2817.18
LRC1_6_5	46	17067.17	CLS	60.00	57.64	30.60	1.43	49.80	17452.85	8.26	2.26	8.22	± 0.35	439.06
LRC1_6_6	42	17405.48	CLS	57.00	56.72	35.86	0.82	47.80	17132.22	13.81	-1.57	13.70	± 0.37	502.40
LRC1_6_7	38	15609.86	CLS	51.00	60.37	34.39	0.87	43.60	15922.78	14.74	2.00	14.65	± 0.55	866.65
LRC1_6_8	33	15919.78	CLS	46.00	57.58	39.54	0.93	39.60	15635.47	20.00	-1.79	19.83	± 0.29	746.23
LRC1_6_9	34	15236.23	CLS	45.00	61.66	32.57	1.15	40.00	15179.70	17.65	-0.37	17.51	± 0.83	971.98
LRC1_6_10	29	14607.38	CLS	42.00	53.71	44.90	1.01	35.20	14305.47	21.38	-2.07	21.18	± 0.50	780.45
LRC2_6_1	16	14782.39	EOE	22.00	105.24	38.53	1.82	19.80	13787.87	23.75	-6.73	23.29	± 1.16	1214.39
LRC2_6_2	13	13958.91	EOE	19.00	111.98	47.31	3.01	16.20	12100.90	24.62	-13.31	23.95	± 1.08	2097.50
LRC2_6_3	10	12741.64	EOE	14.00	109.33	41.44	10.36	13.00	10644.20	30.00	-16.46	29.03	± 0.00	7057.32
LRC2_6_4	7	10536.79	EOE	11.00	106.43	58.35	26.42	10.00	11004.04	42.86	4.43	41.92	± 0.02	29709.58
LRC2_6_5	13	14886.35	EOE	21.00	93.35	62.13	2.78	17.60	12663.15	35.38	-14.93	34.44	± 0.65	2154.25
LRC2_6_6	12	15315.05	EOE	20.00	103.00	67.42	2.42	14.80	13807.69	23.33	-9.84	22.64	± 1.18	1528.62
LRC2_6_7	10	15032.16	EOE	17.00	84.91	70.36	3.38	15.00	12393.18	50.00	-17.56	48.35	± 0.00	3124.05
LRC2_6_8	9	13836.07	EOE	15.00	88.45	67.21	3.67	12.40	11302.06	37.78	-18.31	36.38	± 0.92	3362.97
LRC2_6_9	9	13464.16	EOE	14.00	104.78	56.75	3.05	11.00	11600.16	22.22	-13.84	21.34	± 0.01	3645.81
LRC2_6_10	7	14141.16	EOE	12.00	61.67	71.11	4.76	11.00	11647.15	57.14	-17.64	54.71	± 0.00	4284.72
Average	27.25	14887.39	-	34.60	64.94	36.22	3.45	30.33	14007.19	18.81	-4.97	18.20	± 0.28	2878.21

Table A.5: Average results for 800 locations instances

	BKS			Initial Solution				Final Solution						
Instance	$\#V_b$	Cost	Ref.	$\#V_i$	$\text{gap}_\text{C}(\%)$	$\text{gap}_\text{F}(\%)$	t(s)	$\#V_f$	Cost	$\text{gap}_\text{V}(\%)$	$\text{gap}_\text{C}(\%)$	$\text{gap}_\text{F}(\%)$	$\sigma(\%)$	t(s)
LC1_8_1	80	25184.38	SAM	80.00	17.84	0.07	1.29	80.00	25401.59	0.00	0.86	0.00	$\pm$ 0.00	113.30
LC1_8_2	77	26864.13	CLS	96.00	62.31	24.84	1.56	80.00	25936.49	3.90	-3.45	3.86	$\pm$ 0.00	407.06
LC1_8_3	63	27459.81	CLS	78.00	25.03	23.82	2.02	72.60	25892.34	15.24	-5.71	15.12	$\pm$ 0.16	810.16
LC1_8_4	60	22943.54	CLS	70.00	39.68	16.78	3.06	63.20	23827.18	5.33	3.85	5.33	$\pm$ 0.28	744.81
LC1_8_5	80	25211.22	SAM	80.00	17.70	0.07	1.34	80.00	25331.30	0.00	0.48	0.00	$\pm$ 0.00	110.97
LC1_8_6	80	25164.25	SAM	83.00	29.74	3.85	1.34	80.00	26092.09	0.00	3.69	0.01	$\pm$ 0.00	154.31
LC1_8_7	80	25158.38	SAM	81.00	30.97	1.37	1.38	80.20	25468.26	0.25	1.23	0.25	$\pm$ 0.12	131.29
LC1_8_8	78	25348.45	RP	91.00	64.46	16.86	1.38	79.60	26056.24	2.05	2.79	2.05	$\pm$ 0.15	237.36
LC1_8_9	72	26360.69	CLS	84.00	42.02	16.78	1.70	75.80	26948.77	5.28	2.23	5.26	$\pm$ 0.12	297.98
LC1_8_10	70	26811.45	CLS	84.00	47.35	20.13	1.88	74.00	26608.03	5.71	-0.76	5.68	$\pm$ 0.20	371.70
LC2_8_1	24	11687.06	SAM	27.00	52.97	12.74	3.14	24.00	11793.77	0.00	0.91	0.01	$\pm$ 0.00	226.69
LC2_8_2	24	13990.57	EOE	31.00	68.85	29.45	4.21	25.00	12620.78	4.17	-9.79	4.07	$\pm$ 0.00	1146.81
LC2_8_3	24	13195.83	EOE	31.00	106.16	29.69	5.47	25.00	12535.35	4.17	-5.01	4.10	$\pm$ 0.00	3777.52
LC2_8_4	23	13376.82	RP	29.00	83.20	26.50	10.42	24.00	12537.57	4.35	-6.27	4.27	$\pm$ 0.00	3257.31
LC2_8_5	25	12298.33	EOE	30.00	93.60	20.45	3.33	25.00	12361.32	0.00	0.51	0.00	$\pm$ 0.00	520.66
LC2_8_6	24	12645.72	EOE	30.00	123.55	25.64	3.76	25.20	12417.26	5.00	-1.81	4.96	$\pm$ 0.37	1042.15
LC2_8_7	25	11854.44	CLS	31.00	105.35	24.48	4.52	25.00	12143.40	0.00	2.44	0.01	$\pm$ 0.00	777.70
LC2_8_8	24	11454.33	CLS	30.00	120.51	25.57	4.45	24.20	11916.20	0.83	4.03	0.85	$\pm$ 0.39	1009.01
LC2_8_9	24	11629.41	CLS	30.00	124.52	25.60	4.43	24.00	11782.16	0.00	1.31	0.01	$\pm$ 0.00	989.86
LC2_8_10	23	12459.43	EOE	29.00	92.98	26.54	5.67	24.20	11912.38	5.22	-4.39	5.15	$\pm$ 0.39	1713.21
LR1_8_1	80	39292.13	CLS	88.00	61.79	10.32	1.61	80.20	41508.91	0.25	5.64	0.28	$\pm$ 0.12	353.92
LR1_8_2	59	34325.92	CLS	72.00	61.94	22.32	2.36	63.00	35553.01	6.78	3.57	6.76	$\pm$ 0.47	948.88
LR1_8_3	44	29676.42	CLS	57.00	64.99	29.84	4.40	50.20	29917.95	14.09	0.81	13.98	$\pm$ 0.19	4010.71
LR1_8_4	25	21189.75	CLS	37.00	59.92	48.12	8.94	32.60	21878.47	30.40	3.25	30.12	$\pm$ 0.73	7349.16
LR1_8_5	49	39624.94	CLS	69.00	28.58	40.69	1.83	62.80	37040.42	28.16	-6.52	27.82	$\pm$ 0.37	417.17
LR1_8_6	40	35042.41	CLS	57.00	49.27	42.57	2.60	52.80	33200.06	32.00	-5.26	31.60	$\pm$ 0.33	2223.11
LR1_8_7	30	28252.49	CLS	44.00	51.10	46.72	3.99	41.80	27099.19	39.33	-4.08	38.83	$\pm$ 0.55	2025.28
LR1_8_8	20	20037.07	CLS	30.00	58.49	50.11	7.92	28.20	20366.18	41.00	1.64	40.51	$\pm$ 0.33	6928.47
LR1_8_9	40	40077.86	CLS	55.00	20.28	37.29	2.17	51.80	35615.08	29.50	-11.14	29.00	$\pm$ 0.34	597.73
LR1_8_10	31	32241.06	CLS	43.00	27.32	38.56	2.66	40.40	29750.05	30.32	-7.73	29.83	$\pm$ 0.28	1013.22
LR2_8_1	14	46452.00	CLS	22.00	34.22	56.23	5.55	18.40	30957.02	31.43	-33.36	28.85	$\pm$ 0.63	2481.96
LR2_8_2	12	32575.97	RP	18.00	63.72	50.45	9.86	17.00	26834.31	41.67	-17.63	39.72	$\pm$ 0.01	5998.20
LR2_8_3$^\bullet$	9	30151.50	EOE	14.00	50.61	55.36	23.95	14.00	23469.11	55.56	-22.16	52.43	$\pm$ 0.00	17036.43
LR2_8_4$^\bullet$	6	24285.15	CLS	10.00	45.42	65.64	96.01	10.00	21125.05	66.67	-13.01	62.83	$\pm$ 0.00	63054.35
LR2_8_5	11	37332.53	CLS	16.00	42.24	45.32	7.21	15.00	30563.84	36.36	-18.13	34.15	$\pm$ 0.02	3705.67
LR2_8_6	9	29832.94	EOE	15.00	71.28	66.85	14.02	14.00	25519.90	55.56	-14.46	52.77	$\pm$ 0.01	8814.73
LR2_8_7$^\bullet$	7	27499.87	EOE	12.00	55.17	70.67	43.25	11.00	24260.42	57.14	-11.78	53.92	$\pm$ 0.00	25443.85
LR2_8_8$^\bullet$	5	19256.79	RP	9.00	66.42	79.38	70.76	8.00	18504.01	60.00	-3.91	57.06	$\pm$ 0.00	68343.16
LR2_8_9	10	30791.77	RP	16.00	67.89	60.29	9.87	15.00	27117.35	50.00	-11.93	47.70	$\pm$ 0.01	4746.50
LR2_8_10	9	28265.24	RP	13.00	64.16	45.19	10.95	13.00	29344.71	44.44	3.82	42.91	$\pm$ 0.01	6626.80
LRC1_8_1	66	32302.57	CLS	86.00	59.65	30.48	1.56	72.20	33569.91	9.39	3.92	9.36	$\pm$ 0.38	676.19
LRC1_8_2	56	28042.91	CLS	77.00	63.71	37.66	2.51	61.80	29586.46	10.36	5.50	10.33	$\pm$ 0.50	1279.79
LRC1_8_3	48	24693.73	CLS	62.00	70.11	29.43	4.09	55.20	26355.51	15.00	6.73	14.95	$\pm$ 0.32	3022.48
LRC1_8_4	34	18712.08	EOE	40.00	66.98	17.98	8.37	37.00	20153.09	8.82	7.70	8.82	$\pm$ 0.40	6342.52
LRC1_8_5	58	31457.69	CLS	78.00	51.53	34.60	1.90	68.00	33606.19	17.24	6.83	17.17	$\pm$ 0.65	465.86
LRC1_8_6	54	29836.27	CLS	73.00	53.90	35.31	1.98	63.40	30051.63	17.41	0.72	17.29	$\pm$ 0.44	609.09
LRC1_8_7	51	28705.17	CLS	67.00	57.13	31.55	2.00	58.00	29544.51	13.73	2.92	13.65	$\pm$ 0.44	640.88
LRC1_8_8	45	27374.52	CLS	61.00	59.39	35.74	2.34	55.40	27856.12	23.11	1.76	22.95	$\pm$ 0.21	870.22
LRC1_8_9	44	25980.07	CLS	58.00	56.41	32.00	2.29	52.20	26367.41	18.64	1.49	18.51	$\pm$ 0.60	972.62
LRC1_8_10	40	24582.28	CLS	56.00	60.96	40.16	2.47	47.40	25560.29	18.50	3.98	18.39	$\pm$ 0.51	1009.94
LRC2_8_1	20	23157.34	CLS	30.00	120.76	51.01	4.83	24.40	22733.81	22.00	-1.83	21.66	$\pm$ 1.30	1869.54
LRC2_8_2	17	22686.62	CLS	26.00	118.38	54.01	8.23	22.00	21146.21	29.41	-6.79	28.82	$\pm$ 0.00	4397.41
LRC2_8_3$^\bullet$	14	21651.20	CLS	20.00	93.56	43.82	14.14	19.00	19117.81	35.71	-11.70	34.82	$\pm$ 0.00	13464.57
LRC2_8_4$^\bullet$	11	16149.39	EOE	15.00	93.73	37.40	48.21	15.00	16582.77	36.36	2.68	35.76	$\pm$ 0.00	93746.74
LRC2_8_5	16	24404.69	CLS	27.00	114.50	69.61	6.21	21.00	22075.18	31.25	-9.55	30.49	$\pm$ 0.00	2747.64
LRC2_8_6	15	22992.93	EOE	23.00	101.71	54.24	6.57	19.00	20783.70	26.67	-9.61	25.98	$\pm$ 0.00	2963.34
LRC2_8_7	14	23391.20	EOE	23.00	116.14	65.35	7.22	18.00	19153.42	28.57	-18.12	27.62	$\pm$ 0.01	5118.94
LRC2_8_8	12	20934.68	EOE	21.00	103.94	75.62	8.56	17.00	18700.34	41.67	-10.67	40.55	$\pm$ 0.00	7467.82
LRC2_8_9	11	20444.00	EOE	18.00	103.70	64.55	9.09	15.00	18996.02	36.36	-7.08	35.38	$\pm$ 0.01	5384.56
LRC2_8_10	9	21005.25	EOE	15.00	85.04	67.19	10.39	15.00	18563.98	66.67	-11.62	64.45	$\pm$ 0.01	8754.33
Average	35.42	24930.11	-	44.97	67.41	37.35	9.15	40.10	23828.53	21.98	-3.63	21.32	$\pm$ 0.21	6862.73

Note: Names of instances denoted with *name*$^\bullet$ have been run only one time, because of their potentially long computation time.

Table A.6: Average results for 1000 locations instances

	BKS			Initial Solution				Final Solution						
Instance	$\#V_b$	Cost	Ref.	$\#V_i$	$gap_c(\%)$	$gap_f(\%)$	t(s)	$\#V_f$	Cost	$gap_v(\%)$	$gap_c(\%)$	$gap_f(\%)$	$\sigma(\%)$	t(s)
LC1_10_1	100	42488.66	SAM	100.00	14.14	0.07	2.31	100.00	43373.32	0.00	2.08	0.01	± 0.00	143.50
LC1_10_2	94	45238.29	CLS	117.00	44.18	24.59	2.60	97.60	43895.90	3.83	-2.97	3.79	± 0.12	420.89
LC1_10_3	80	45175.07	CLS	100.00	31.33	25.04	3.49	89.80	42560.93	12.25	-5.79	12.12	± 0.20	1045.48
LC1_10_4	74	38376.00	CLS	87.00	28.10	17.64	4.77	79.20	39328.99	7.03	2.48	7.00	± 0.12	1251.60
LC1_10_5	100	42477.40	SAM	100.00	15.72	0.08	2.50	100.00	43864.44	0.00	3.27	0.02	± 0.00	234.42
LC1_10_6	101	42838.39	SAM	107.00	28.89	6.06	2.42	101.00	43499.55	0.00	1.54	0.01	± 0.00	185.04
LC1_10_7	100	42854.99	SAM	103.00	23.80	3.11	2.53	100.60	43845.07	0.60	2.31	0.61	± 0.12	155.68
LC1_10_8	98	42951.56	RP	111.00	44.05	13.43	2.42	100.20	44154.11	2.24	2.80	2.25	± 0.09	322.58
LC1_10_9	91	43426.76	CLS	109.00	51.04	19.97	2.89	94.80	44067.76	4.18	1.48	4.16	± 0.19	435.34
LC1_10_10	88	42873.50	CLS	103.00	46.34	17.22	3.04	92.60	43367.14	5.23	1.15	5.20	± 0.13	846.90
LC2_10_1	30	16879.24	SAM	34.00	43.07	13.54	5.70	30.00	16879.24	0.00	0.00	0.00	± 0.00	290.12
LC2_10_2	31	18980.98	RP	39.00	70.76	26.15	7.49	32.40	17999.86	4.52	-5.17	4.44	± 0.36	1913.80
LC2_10_3	30	17772.49	RP	37.00	93.47	23.85	9.71	31.60	17798.29	5.33	0.15	5.30	± 0.37	3382.27
LC2_10_4	29	18089.98	RP	35.00	76.38	21.12	17.11	32.00	19705.29	10.34	8.93	10.33	± 0.00	5965.67
LC2_10_5	31	17137.53	RP	38.00	76.16	22.95	5.98	31.00	17437.32	0.00	1.75	0.01	± 0.00	915.95
LC2_10_6	31	17198.01	RP	38.00	97.45	23.10	7.16	31.00	17509.16	0.00	1.81	0.01	± 0.00	1055.11
LC2_10_7	31	19117.67	RP	38.00	81.47	23.03	6.57	32.00	18128.56	3.23	-5.17	3.16	± 0.00	1998.26
LC2_10_8	30	17015.41	CLS	37.00	111.23	23.95	7.63	30.00	17557.63	0.00	3.19	0.02	± 0.00	1369.80
LC2_10_9	30	20057.42	CLS	39.00	97.32	30.56	9.09	31.40	18013.26	4.67	-10.19	4.54	± 0.37	1513.33
LC2_10_10	29	17425.55	RP	36.00	80.69	24.56	10.02	30.40	17932.10	4.83	2.91	4.81	± 0.38	2657.55
LR1_10_1	100	56875.21	CLS	106.00	57.88	6.37	2.99	100.00	61016.83	0.00	7.28	0.05	± 0.00	610.37
LR1_10_2	80	49627.16	CLS	83.00	56.71	4.16	4.08	80.00	51888.55	0.00	4.56	0.04	± 0.00	1169.33
LR1_10_3	54	42124.44	RP	67.00	62.76	24.45	8.14	61.40	44267.68	13.70	5.09	13.62	± 0.19	3381.62
LR1_10_4	28	31617.58	CLS	43.00	54.95	53.59	16.82	39.00	31769.20	39.29	0.48	38.75	± 0.00	5712.96
LR1_10_5	59	60616.90	CLS	85.00	26.39	43.84	3.68	77.80	56083.72	31.86	-7.48	31.37	± 0.44	1438.18
LR1_10_6	48	51842.12	CLS	69.00	32.08	43.59	4.94	64.00	46847.02	33.33	-9.64	32.76	± 0.00	3389.25
LR1_10_7	36	40127.52	CLS	53.00	52.21	47.29	7.97	49.00	37896.55	36.11	-5.56	35.54	± 0.00	3257.60
LR1_10_8	26	29099.22	CLS	40.00	66.42	54.02	16.02	36.00	30799.76	38.46	5.84	38.01	± 0.00	6677.52
LR1_10_9	49	53353.22	CLS	70.00	34.30	42.74	4.16	63.80	51495.58	30.20	-3.48	29.75	± 0.15	724.66
LR1_10_10	39	47290.00	CLS	53.00	28.79	35.79	4.82	49.00	45086.82	25.64	-4.66	25.19	± 0.52	1186.52
LR2_10_1	18	56089.88	CLS	27.00	47.98	49.92	10.48	22.00	47725.75	22.22	-14.91	20.83	± 0.01	4291.17
LR2_10_2	14	58654.95	CLS	20.00	28.97	42.17	19.14	20.00	45915.94	42.86	-21.72	39.64	± 0.02	7926.49
LR2_10_3$^\bullet$	11	39894.32	RP	17.00	63.84	54.95	42.28	17.00	34414.86	54.55	-13.73	51.58	± 0.00	31577.05
LR2_10_4$^\bullet$	8	28314.95	RP	12.00	79.76	51.26	65.30	11.00	29594.53	37.50	4.52	36.10	± 0.00	122430.03
LR2_10_5	14	53209.98	RP	19.00	47.89	36.27	14.33	18.00	43888.82	28.57	-17.52	26.48	± 0.01	5405.58
LR2_10_6$^\bullet$	11	53051.42	CLS	18.00	42.42	62.43	23.70	17.00	41776.22	54.55	-21.25	50.24	± 0.00	14960.15
LR2_10_7$^\bullet$	9	36728.20	RP	15.00	80.34	67.33	44.78	14.00	33244.64	55.56	-9.48	52.40	± 0.00	62304.89
LR2_10_8$^\bullet$	7	26278.09	RP	11.00	94.73	58.83	90.43	10.00	27454.60	42.86	4.48	41.14	± 0.00	146045.85
LR2_10_9	13	48447.49	RP	20.00	74.60	54.77	15.02	20.00	39473.93	53.85	-18.52	50.62	± 0.00	8469.63
LR2_10_10$^\bullet$	11	43889.20	CLS	17.00	67.02	55.14	19.18	16.00	40832.30	45.45	-6.97	42.96	± 0.00	11830.38
LRC1_10_1	82	49285.19	CLS	100.00	52.55	22.18	3.84	91.00	51963.26	10.98	5.43	10.93	± 0.28	672.97
LRC1_10_2	72	45289.03	CLS	93.00	66.27	29.46	4.31	81.60	48729.25	13.33	7.60	13.29	± 0.23	1743.08
LRC1_10_3	53	36499.96	CLS	68.00	61.68	28.59	8.40	62.00	38503.35	16.98	5.49	16.88	± 0.00	3959.52
LRC1_10_4	40	27680.12	K	55.00	69.84	37.78	20.49	46.00	31862.66	15.00	15.11	15.00	± 0.00	7012.44
LRC1_10_5	72	51733.03	CLS	94.00	46.45	30.70	3.30	85.60	55046.03	18.89	6.40	18.78	± 0.13	744.23
LRC1_10_6	68	44444.34	CLS	85.00	51.53	25.22	3.33	79.00	47445.35	16.18	6.75	16.10	± 0.27	833.61
LRC1_10_7	61	41917.62	CLS	81.00	57.14	32.99	3.68	73.40	43835.85	20.33	4.58	20.19	± 0.33	1071.41
LRC1_10_8	56	42640.89	CLS	76.00	56.31	35.91	3.95	69.20	43921.15	23.57	3.00	23.38	± 0.45	1174.09
LRC1_10_9	53	40848.27	CLS	73.00	50.77	37.86	4.02	66.20	42251.39	24.91	3.43	24.70	± 0.47	1243.88
LRC1_10_10	48	36092.22	CLS	67.00	62.65	39.80	4.54	58.80	38177.57	22.50	5.78	22.34	± 0.30	2658.57
LRC2_10_1	22	34960.69	CLS	31.00	96.65	41.99	8.72	27.00	35411.72	22.73	1.29	22.31	± 0.02	4220.67
LRC2_10_2	20	33576.15	CLS	27.00	102.58	36.39	16.43	26.00	34624.09	30.00	3.12	29.45	± 0.02	6313.46
LRC2_10_3$^\bullet$	16	28403.51	RP	23.00	106.13	45.10	41.41	21.00	29124.13	31.25	2.54	30.63	± 0.00	22934.50
LRC2_10_4$^\bullet$	11	26239.60	CLS	18.00	82.94	64.20	172.12	17.00	25610.10	54.55	-2.40	52.90	± 0.00	99520.10
LRC2_10_5	17	37312.43	CLS	27.00	91.54	59.70	10.99	22.00	30055.26	29.41	-19.45	28.11	± 0.00	5935.13
LRC2_10_6	17	31470.58	EOE	27.00	109.08	59.96	11.88	24.00	33505.18	41.18	6.47	40.39	± 0.03	5696.61
LRC2_10_7	16	32537.87	CLS	24.00	95.46	51.13	12.45	21.00	31373.14	31.25	-3.58	30.39	± 0.02	6221.15
LRC2_10_10$^\bullet$	11	31334.49	CLS	18.00	86.18	64.41	16.98	15.00	27230.27	36.36	-13.10	34.66	± 0.00	14290.88
Average	44.79	37720.22	-	55.86	62.09	34.35	15.25	50.66	36914.84	20.93	-1.34	20.26	± 0.11	11295.50

Note: Names of instances denoted with *name*$^\bullet$ have been run only one time, because of their potentially long computation time.