# A Comparative Study of Labeling Algorithms within the Branch-and-Price Framework for Vehicle Routing with Time Windows

Thèse présentée en vue de l'obtention du grade de
Docteur en Sciences Économiques et de Gestion par

**Stefano Michelini**
**HEC Liège — Université de Liège**

Promoteur:
**Yasemin Arda**, Université de Liège

Membres du Jury:
**Yves Crama**, Université de Liège
**Dominique Feillet**, École des Mines de Saint-Étienne
**Hande Küçükaydın**, MEF University, Istanbul
**Martine Labbé**, Université Libre de Bruxelles
**Michaël Schyns**, Université de Liège

# Summary

The Vehicle Routing Problem with Time Windows (VRPTW) is a well-known extension of the Vehicle Routing Problem, one of the oldest and most studied problems in combinatorial optimization. The VRPTW consists in finding an optimal set of routes for a fleet of vehicles based in a single depot in order to service a set of customers. Each customer is associated with a time window, which specifies the earliest and the latest possible service start times. Time windows are a natural feature of a number of applications, such as postal deliveries, grocery deliveries, or school bus routing.

In addition to the VRPTW, we consider a variant of this problem where the aim is to minimize the total route duration, instead of just the total travel time. Thus, the waiting times that are incurred by vehicles before they can service a customer are taken into account in the objective function of the problem. This aspect is relevant in applications where waiting times bear an implicit cost, such as when the vehicles are rented by the distribution company, or when they consume energy during idle times, as in the case of refrigerated trucks delivering perishable goods.

Branch-and-Price (BP) is one of the most effective and commonly used exact methodologies for solving routing problems. In recent years, several studies have investigated advanced labeling algorithms to solve the related pricing problem, which is usually a variant of the elementary shortest path problem with resource constraints. Being able to solve this subproblem efficiently is crucial, since it is a major bottleneck for the performance of the BP procedure. Each of these methods uses a certain strategy to relax the elementarity constraints of the pricing problem in order to accelerate its solution. In this study, we investigate the performances of several such methods within a BP framework.

In order to perform rigorous comparisons, we first parametrize several algorithmic components. Then, we search for good parameter configurations for each algorithm with a tool for automated parameter tuning. Finally, we run the best configuration found for each algorithm on benchmark instances and analyze the results with statistical tests. Our results show in particular that a class of hybrid algorithms, where multiple customer sets are used to control the relaxation of the elementarity conditions, rather than a single one, outperforms all the others.

# Acknowledgements

It is fair to say that this work is a collaborative effort. Indeed, colleagues, friends, and family members helped me in so many occasions and in so many different ways that I am afraid that these acknowledgements might not do them justice. But I am going to try.

First of all, I would like to thank my supervisor, Yasemin. This would not have been possible without your guidance, your help, and your patience. Thank you for giving me this opportunity, and for helping me realize its potential from start to finish. You have been a great boss.

I would also like to acknowledge the other members of the thesis committee and jury. Thank you, Hande, for your support and counsel, which you have given me with a contagious cheerfulness from the very start of my doctoral studies, up to the very end. I am grateful to Prof. Crama for his invaluable support as a teacher and as a committee member. I will always treasure your gifts in both knowledge and wisdom. Many thanks to Prof. Labbé for her advice as a committee member, and especially for helping me find this opportunity in Belgium after the end of my studies for my master's degree. Thank you, Prof. Schyns and Prof. Feillet, for your availability as members of the jury, for your precious feedback, and for all your kind words abour my work.

Additionally, I would like to thank Prof. Louveaux for his supervision at the end of my studies for the master's degree, and for introducing me to the Belgian OR community.

Special thanks go to Manuel López-Ibáñez and Leslie Pérez Cáceres, for their crucial support with the `irace` software, and to CÉCI (Consortium des Équipements de Calcul Intensif) for providing computational resources.

I am grateful to the Interuniversity Attraction Poles Programme of the Belgian Science Policy Office (grant P7/36) for funding this thesis.

I am very thankful to all the former and current members of Office 334. Elisabeth, we started this doctoral adventure together and we have ended it (almost) at the same time. You have been a great colleague and a great friend since day one. Thanks to you and Ángel for all the fun times spent together. Virginie, you have been a great source of energy and fun for all of us in the department. I have missed you dearly during the last few years, but I am thankful for the time we shared an office together. Célia, you have offered me invaluable help on so many things and on innumerable occasions, that I am sure I would not have made it without you. Stéphanie, thank you so much for all your advice and heartfelt conversations at the end of the workday. Your support has meant a lot to me. Anne-Sophie, thank you for your kindness and your constant availability to help.

Many thanks go to Alessandro, who has been a great companion for so many good times together, including field trips, holidays, dinners, and drinks after work. I would also like to thank all the former and current members of QuantOM that I have had the pleasure to meet: Thierry, Véronique, Guillaume, Christine, Martine, Julien, Sabine, Anaïs, Maud, Reza, Bart, Quentin, Cédric, Ashwin, Christian, Thu, and Valentin. It has been an honor to be in the same department as you.

A lot of thanks to all my friends, including all the ones back in Italy that welcome me with laughter and good times whenever I come back, and the new ones I have made abroad, including Marta, Itxi, Ramona, Andrew, and Tristan. I am thankful in particular to my friend of a lifetime, Francesco, for all the fun times together, for always being there when I needed to talk, and of course, for all the Lego gifts. You are a brother to me.

Grazie infinite a papà Franco e mamma Raffaella per tutto il vostro aiuto e il vostro affetto. Non c'é giorno in cui non siate disponibili a sostenermi e a darmi consigli, e non ce l'avrei mai fatta senza di voi. Vi voglio un mondo di bene.

Finally, a very special thank you to my dear Charlotte. I am thankful for your unwavering support, for your sweetness and kindness, for somehow tolerating all my questionable attempts at humor, and for helping me up whenever I felt down. You have been the light of my life over the past few years, and give a special meaning to each new day. I love you.

. . . Also, many thanks to Bouh for being the silliest little dog I have ever had the pleasure to befriend.

# Contents

# List of Abbreviations

| | |
|---|---|
| BB | Branch-and-Bound |
| BCP | Branch-and-Cut-and-Price |
| BP | Branch-and-Price |
| CG | Column Generation |
| CVRP | Capacitated Vehicle Routing Problem |
| DSSR | Decremental State Space Relaxation |
| ESPPRC | Elementary Shortest Path Problem with Resource Constraints |
| LP | Linear Program |
| MIP | Mixed-Integer Program |
| MP | Master Problem |
| NGRR | *ng*-route Relaxation |
| REF | Resource Extension Function |
| RMP | Restricted Master Problem |
| VRPTW | Vehicle Routing Problem |
| VRPTW | Vehicle Routing Problem with Time Windows |
| VRPTWWTC | Vehicle Routing Problem with Time Windows and Waiting Time Costs |

# List of Algorithmic Parameters

| | | |
|---|---|---|
| `tree_trav` | {`breadth, depth, best`} | Strategy for BB tree traversal |
| `n_col` | $]0,1]$ | Max. num. of paths inserted in RMP at each CG iteration |
| `n_conc` | $]0,10000]$ | Max. num. of paths generated at end of labeling algorithm iteration |
| `dssr_init_s` | {`none, hca, tca, whca, wtca, mix`} | Strategy for critical vertex set initialization at start of labeling algorithm |
| `dssr_init_n` | $]0,1[$ | Num. of vertices to insert in critical set at labeling algorithm initialization |
| `dssr_path_s` | {`1-p, in-btw, all-p`} | Strategy for quantity of paths to check for elementarity violations at end of labeling algorithm iteration |
| `dssr_path_n` | $]0,1[$ | Num. of paths to check for elementarity violations if `dssr_path_s` is set to `in-btw` |
| `dssr_node_s` | {`1-n, in-btw, all-n`} | Strategy for quantity of nodes visited multiple times in a given path to mark as critical |
| `dssr_node_n` | $]0,1[$ | Num. of nodes visited multiple times in a given path to mark as critical if `dssr_path_s` is set to `in-btw` |
| `ng_m` | {`tt, ccr, mix`} | Type of metric used to build *ng*-neighborhoods |
| `ng_s` | $]0,1[$ | Size of *ng*-neighborhoods |
| `ng_mix` | $]0,1[$ | Affine combination coefficient used in mixed metric if `ng_m` is set to `mix` |

# Chapter 1

# Introduction

The vehicle routing problem (VRP) is one of the most well-known problems in the field of operational research. First formally introduced by Dantzig and Ramser (1959) under the name of "truck dispatching problem", over the years it has been the subject of a large number of studies. Interestingly, the VRP has a relatively straightforward structure: given a set of geographically distributed customers and a fleet of delivery vehicles based at a depot, the aim is to design a set of least cost routes (ordered sequences of customers, each starting and ending at the depot, and each assigned to a single vehicle) under a number of side constraints. In other words, it is a generalization of the traveling salesman problem (TSP), which consists in determining a single Hamiltonian cycle visiting all the given customers.

Even though it is easy to formulate, the VRP is quite difficult to solve: the TSP, despite being an NP-hard problem, can nowadays be quickly solved for thousands or tens of thousands of customers, while the VRP proves to be a challenge for exact algorithms even for just a couple hundred customers (Laporte et al., 2013). Because of this, it has proven to be fertile ground for the development of both exact and heuristic solution methods.

Besides the methodological aspect, the VRP attracts a significant practical interest, given its importance in distribution management. Considering the high number of possible side constraints or other problem features (e.g., whether there are pick-ups instead of, or in addition to deliveries, whether the vehicle fleet is heterogeneous, . . . ), it is faced daily in one form or another by thousands of carriers worldwide.

In this thesis, we focus on a particular variant of the problem, which presents a side constraint that is commonly encountered in many scenarios: the VRP with *time windows* (VRPTW). Here, each customer is associated with a time window, comprised of an earliest and a latest service start time.

In order to service a customer, the assigned vehicle has to reach the corresponding vertex before the latest service start time. However, if it arrives before the earliest service time, it incurs a waiting time. As we are going to see, this problem is an important fixture of the literature in its own right, since it has been the subject of decades of research.

In the classic VRPTW, the cost of a route is defined as in the case of the VRP, i.e., the total distance traveled, which is commonly assumed to be equivalent to the total travel time. However, some authors have pointed out that the presence of time windows gives the chance to introduce more realistic objective functions, such as the minimization of total route duration (Savelsbergh, 1992), which takes into account vehicle waiting times, besides the travel times. Therefore, in addition to the classic VRPTW, in this thesis we consider a variant with such an objective function, which we call VRPTW with waiting time costs (VRPTWWTC), where the departure times of the vehicles at the depot also needs to be determined with the aim of minimizing the total route duration.

There are several possible practical scenarios where waiting times bear a meaningful cost. For instance, the hourly wages of drivers or other on-board personnel can be a significant part of the overall transportation cost, thus making it important to minimize their idle time. Examples include home healthcare routing problems, where vehicles carrying nurses are dispatched in order to provide service to their patients, or scenarios where technicians need transportation in order to perform installations or repair operations for the customers. Furthermore, route duration minimization can be relevant for dial-a-ride problems of the kind faced by ride-sharing services, where in addition to the need to compensate drivers for their idle time, one has to consider that high waiting times can compel a driver to reject a ride request. Another situation where waiting time minimization is relevant is when the continued operation of the vehicle requires some sort of operational cost, such as the energy expenditure for the refrigeration of perishable goods (Hsu et al., 2007), or fuel cost if the vehicle is not allowed to park near a customer while waiting (especially in the case of urban logistics). This can also apply to scenarios where delivery vehicles are not owned, but rented by the distribution company, since fees incurred by the company can be a function of the duration of the rental.

As in the case of the VRP, many solution methods have been studied for the VRPTW, both heuristic and exact. Among the exact methods, one of the most effective and well-known is branch-and-price (BP), which consists in a branch-and-bound algorithm, where each linear relaxation is solved with column generation. Column generation is particularly suited to this problem, given the constraining nature of time windows and the very large number of variables that are present when each route is associated to a variable.

Nevertheless, being able to efficiently solve the associated pricing problem remains an important issue, since it often is the main performance bottleneck. Because of this, several algorithms have been developed in order to accelerate the dynamic programming method that is traditionally used to solve the pricing problem. Each of those methods attempts to speed up the solution procedure of the pricing problem by strategically relaxing the state space in a certain way and gives rise to a different labeling algorithm. To the best of our knowledge, however, there has been no attempt in the literature to systematically compare the performance of these labeling algorithms. Thus, the aim of this thesis is to perform such a comparative study in an attempt to determine whether there are certain common properties underlying the most effective techniques.

While testing these methods on instances of the pricing problem can certainly be useful, this might not offer enough information on their performance when used as a component of a BP type algorithm designed to solve a routing problem. Therefore, each labeling algorithm under our consideration is embedded in a BP framework, and the analysis is carried out for both the VRPTW and the VRPTWWTC.

The analysis of such algorithms presents a methodological challenge. Each procedure we consider can adopt several algorithmic strategies for each aspect of its execution, in addition to a quantity of numerical parameters. In the literature, these strategies and parameters are most commonly set manually after having performed preliminary experiments. However, doing so for each of the algorithms under consideration presents the risk of performing a fallacious analysis. Thus, for the sake of rigor, we aim to carry out a more systematic approach: for each algorithm under our consideration, we perform a *parametrization*, i.e., we describe its possible algorithmic strategies by the use of parameters. Then, we configure all the algorithmic parameters employing a dedicated automatic tuner, with the aim of obtaining the best possible configuration for each procedure. Finally, we test each algorithm thusly tuned on a set of benchmark instances and compare the results with statistical tests, which aim to determine whether some algorithms perform significantly better than others.

It is important to keep in mind that in this study we do not aim to develop a competitive solution method for the considered routing problems. The current state-of-the-art is based on branch-and-cut-and-price (BCP), which is the combination of of BP with cut generation. The most recent algorithms (Pecin et al., 2017a,b) employ many advanced methods for the acceleration of BCP present in the literature. In particular, they use valid inequalities that are based on the well-known subset-row inequalities introduced by Jepsen et al. (2008), which can significantly reduce the integrality gap but are also known to complicate the structure of the pricing problem and thus its resolution. While applying the methodology that we use in this study to a more advanced BCP framework is possible, the additional algorithmic strategies used in these BCP methods would increase severely the number of parameters to tune. This would lead, for each labeling algorithm under consideration, either to an increased parameter tuning time or less performant parameter configurations. Thus, in order to maintain a reasonable scope for our study and to focus on the comparison of labeling algorithms using different state space relaxation techniques with regard to the elementarity resources, we use a BP framework that adopts well-known, reasonably effective and relatively simple features.

## 1.1 Thesis Structure

The thesis is structured as follows. In Chapter 2, we give a general introduction to VRPTW and its associated literature. Here, we formally describe the problem, introduce some of its most widely used mathematical formulations, and offer a brief overview of the solution methods that have been developed over the years, both heuristic and exact.

Chapter 3 describes BP methods that have been proposed in the literature for the VRPTW and presents the details of the labeling procedures for its pricing problem, which are at the basis of the analyses carried out in this study, while Chapter 4 introduces the VRPTWWTC, and explains how to adapt these labeling algorithms to its special structure, according to the theoretical results presented by Küçükaydın et al. (2014).

In Chapters 5 and 6, we present our own analyses on BP algorithms for the VRPTW and the VRPTWWTC, and expand on the work by Michelini et al. (2018). In the former, we present each procedure under our consideration by describing how it works and how it is parametrized, and we discuss the details of the shared BP framework. In the latter, we define our experimental methodology, present the results of our computational experiments, and offer some additional analyses.

Finally, Chapter 7 concludes this thesis and discusses possible directions for future research.

# Chapter 2

# An Overview on the Vehicle Routing Problem with Time Windows

In this chapter, we present a brief overview of the body of knowledge concerning the vehicle routing problem with time windows (VRPTW) and its solution methods. This chapter is structured as follows: in Section 2.1, we give a general description of the VRPTW, with some of its formulations, its history, and its applications. In Sections 2.2 and 2.3, respectively, we give an overview of exact and heuristic methods that have been studied in the literature.

## 2.1 Overview

The VRPTW is a well-known generalization of the capacitated vehicle routing problem (CVRP). In the CVRP, we consider a set of customers, each having a certain demand, and a homogeneous fleet of vehicles, each having the same capacity. All the vehicles are based at a single depot. A vehicle can leave the depot and visit a sequence of customers (a *route*) in order to satisfy their demand, and then must return to the depot. The cumulative demand of the customers visited along a route must not exceed the vehicle capacity. The aim of the CVRP is to find a set of routes that visits all the customers, while minimizing the total distance traveled by the vehicles.

In the VRPTW, we additionally establish that the service at each customer has to start within an associated time interval, which we call *time window*. Arrival before the opening of the time window is allowed in general, but the servicing vehicle has to wait until the opening occurs in order to start the service. This requirement is encountered in many practical applications, like postal and bank services, grocery delivery, or school bus routing and scheduling. Several problems, most notably dial-a-ride problems, feature soft time windows, which can be violated barring a penalty cost. In this study, we focus on the more widespread case of hard time windows.

Since the CVRP is a special case of the VRPTW (one in which all the time windows are infinitely large), and the CVRP is an NP-hard problem, it is easy to see that, by restriction, the VRPTW is also NP-hard. In fact, it has been shown that finding a feasible solution for a fixed number of vehicles is in itself an NP-complete problem (Savelsbergh, 1985).

Given the problem difficulty, it is not surprising that early work on the VRPTW consists mainly of case studies treated employing heuristic methods, with some notable examples being the following. Pullen and Webb (1967) considered the problem of van driver scheduling for mail delivery in the London area. Their heuristic solution method manipulates allocations of jobs to vehicles in order to reduce idle time and empty-running time in the schedules. Knight and Hofer (1968) studied the fleet scheduling problem of a contract transport company, and developed a heuristic system to increase the utilization of vehicles. Madsen (1976) tackled a routing problem with tight due dates faced by a newspaper and magazine distribution company, and developed an algorithm based on Monte Carlo simulation.

Later work found fertile ground for further development of heuristic methods. For instance, Solomon (1987) worked on a variety of route construction heuristics, obtaining satisfying results with a sequential time-space insertion approach. Notably, in this work Solomon also introduced a set of randomly generated instances for the VRPTW, which has since become one of the most widely accepted standard benchmark for research on this problem. Relatedly, literature concerning route improvement heuristics started to grow. Some of these studies adapted existing successful methods for the CVRP (see Baker and Schaffer, 1986, Cook and Russell, 1978). The following decade saw the development of more heuristic methods, and also the first metaheuristics, such as tabu search and genetic algorithms, which gradually became the dominant class of heuristic methods for this problem.

Studies on exact methods started to appear at the beginning of the 1980s, with some of the first branch-and-bound methods appearing in the studies by Christofides et al. (1981b), Baker and Rushinek (1982), and Trienekens (1982). Other early exact methods include the studies by Kolen et al. (1987) and Jörnsten et al. (1986), which are respectively an adaptation of the $q$-path relaxation algorithm of Christofides et al. (1981a) for the CVRP, and an example of Lagrangian relaxation.

The following sections will examine in more detail some of the formulations for the VRPTW, as well as heuristic and exact solution methods. For further reading on the subject, the reader is referred to the surveys by Cordeau et al. (2002), Bräysy and Gendreau (2005a,b), Kallehauge (2008), Potvin (2009), Baldacci et al. (2012), and Desaulniers et al. (2014).

### 2.1.1 Formulation

The VRPTW is defined on a directed graph $G = (V, A)$, where $V = \{0, 1, \ldots, n, n+1\}$ is the set of nodes (or vertices) and $A = \{(i, j) : i, j \in V,\ i \neq j\}$ is the set of arcs. The depot is represented by the nodes 0 and $n+1$, denoted as the *source* and *sink*, respectively. Each customer is associated to a single node $i \in N = V \setminus \{0, n+1\}$. For each node $i$ in $V$, we associate a demand $d_i$, a service time $s_i$, and a time window $[a_i, b_i]$, where $a_i$ is the earliest possible and $b_i$ the latest possible start of service time at $i$. To simplify notation, we assign zero demand and service time to the depot, i.e., $d_0 = d_{n+1} = 0$ and $s_0 = s_{n+1} = 0$. Moreover, we also associate a time window $[a_0, b_0] = [a_{n+1}, b_{n+1}]$, where $a_0$ is the earliest possible departure time from the depot and $b_{n+1}$ is the latest possible arrival time at the depot. The depot houses a fleet of vehicles, represented by the set $K$, and each vehicle has a maximum capacity of $Q$. Every feasible route is then defined as a source-sink elementary path in $G$ that respects the capacity and time window constraints. For each arc $(i, j)$ in $A$, we denote its cost with $c_{ij}$ and its travel time with $t_{ij}$. The cost $c_{ij}$ often corresponds to the Euclidean distance between nodes $i$ and $j$. Furthermore, we make the following assumptions regarding the problem parameters:

- The arc travel times satisfy the *triangle inequality*, i.e., $t_{ih} \leq t_{ij} + t_{jh}$ for all $i, j, h$ in $V$.
- We assume $a_0 \leq \min_{i \in V \setminus \{0\}} \{b_i - t_{0i}\}$ and $b_{n+1} \geq \max_{i \in V \setminus \{0\}} \{\max\{a_0 + t_{0i}, a_i\} + s_i + t_{i,n+1}\}$, otherwise no feasible solution can exist.
- In order to simplify the problem, we can remove from $A$ arcs that cannot belong to a feasible solution, either because they violate time window or capacity constraints, i.e., any arc $(i, j)$ such as $a_i + s_i + t_{ij} > b_j$ or $d_i + d_j > Q$.

We present a standard mixed integer programming (MIP) formulation for the VRPTW with two types of variables. Firstly, we associate a binary arc-flow variable $x_{ijk}$ with each arc $(i, j) \in A$ and each vehicle $k \in K$ that assumes value 1 if arc $(i, j)$ is used by vehicle $k$, and 0 otherwise. Secondly, we define a time variable $T_{ik}$ denoting the start of the service time at node $i$ by vehicle $k$. What follows is the multi-commodity network flow model for the VRPTW:

$$\min \sum_{k\in K}\sum_{(i,j)\in A} c_{ij}x_{ijk} \tag{2.1}$$

$$\text{s.t.} \sum_{k\in K}\sum_{j\in\delta^+(i)} x_{ijk} = 1 \qquad \forall i \in N, \tag{2.2}$$

$$\sum_{j\in\delta^+(0)} x_{0jk} = 1 \qquad \forall k \in K, \tag{2.3}$$

$$\sum_{i\in\delta^-(j)} x_{ijk} - \sum_{i\in\delta^+(j)} x_{jik} = 0 \qquad \forall k \in K, j \in N, \tag{2.4}$$

$$\sum_{i\in\delta^-(n+1)} x_{i,n+1,k} = 1 \qquad \forall k \in K, \tag{2.5}$$

$$a_i \le T_{ik} \le b_i \qquad \forall k \in K, i \in V, \tag{2.6}$$

$$T_{ik} + s_i + t_{ij} - T_{jk} \le M_{ij}(1 - x_{ijk}) \qquad \forall k \in K, (i,j) \in A, \tag{2.7}$$

$$\sum_{i\in N} d_i \sum_{j\in\delta^+(i)} x_{ijk} \le Q \qquad \forall k \in K, \tag{2.8}$$

$$x_{ijk} \in \{0,1\} \qquad \forall k \in K, (i,j) \in A, \tag{2.9}$$

where $\delta^-(i)$ and $\delta^+(i)$ are the sets of predecessors and successors of $i$, respectively, and $M_{ij}$ are sufficiently large constants that can be set to $\max\{0, b_i + s_i + t_{ij} - a_j\}$.

In this MIP model, objective function (2.1) aims to minimize the total cost. Constraints (2.2) ensure that each customer is assigned to a single vehicle. Flow constraints (2.3)-(2.5) establish that all routes are elementary paths starting from the source and ending at the sink. Constraints (2.7) and (2.6) ensure scheduling feasibility and that time windows are not violated, respectively. Constraints (2.8) prevent violation of vehicle capacity. Finally, we impose that the arc-flow variables $x_{ijk}$ have to be binary.

Unfortunately, the linear relaxation of this model is known to provide weak lower bounds. It is possible to obtain a different model with stronger lower bounds by applying Dantzig-Wolfe decomposition (Dantzig and Wolfe, 1960) to the one above, once the arc-flow variables of each vehicle are aggregated. This results in the following set partitioning model:

$$\min \sum_{r\in\Omega} c_r y_r \tag{2.10}$$

$$\text{s.t.} \sum_{r\in\Omega} a_{ir} y_r = 1 \qquad \forall i \in N, \tag{2.11}$$

$$\sum_{r\in\Omega} y_r \leq |K| \tag{2.12}$$

$$y_r \in \{0,1\} \qquad \forall r \in \Omega. \tag{2.13}$$

In this model, $\Omega$ is the set of all feasible routes, $c_r$ the cost of route $r \in \Omega$, and $a_{ir}$ the number of times route $r$ visits customer $i \in N$. When route $r$ is elementary, $a_{ir} \in \{0,1\}$ for all $i \in N$. Binary variable $y_r$ assumes value 1 if route $r$ is used in the solution and 0 otherwise. Here, objective function (2.10) aims to minimize the total cost, constraints (2.11) express that each customer is visited exactly once, and constraints (2.13) force the route variables to be binary. As mentioned above, the arc-flow variables are aggregated and do not appear explicitly in this model. Furthermore, constraint (2.12) can be dropped when the number of vehicles at the depot is supposed to be unlimited, i.e., $|K| = +\infty$, which is an oft-used assumption in the VRPTW literature. In the following chapters of this study, we make use of this assumption, and therefore do not use constraint (2.12).

While the linear relaxation of this model does indeed provide stronger lower bounds, its number of variables is extremely large, given that there is one variable for each feasible route. This issue is traditionally addressed with the use of column generation (CG) methods, upon which we will expand later in this study.

## 2.2 Exact Methods

One of the leading methodologies for the VRPTW is branch-and-price, which is treated in more detail in the next chapter. In this section, we briefly overview some of the other existing exact methods for the VRPTW. For additional reading, see the surveys by Kallehauge (2008), Desaulniers et al. (2011), and Baldacci et al. (2012).

### 2.2.1 Valid Inequalities and Branch-and-Cut

Valid inequalities are a versatile tool to solve a MIP, since they can be added when solving a linear relaxation in order to tighten the lower bound or obtain integer solutions, as in the cutting plane method.

For instance, some of the most well-known valid inequalities for routing problems are the $\kappa$-path inequalities, which are used for the linear relaxation of the model (2.1)-(2.9). Let us consider a subset of customers $S \subseteq N$ and the flow $x(S)$ into $S$, i.e., $x(S) = \sum_{k\in K} \sum_{i\in V\setminus S} \sum_{j\in S} x_{ijk}$. Then, the classic subtour elimination constraints can be expressed as

$$x(S) \geq 1, \qquad \forall S \subseteq N,\ \text{s.t.}|S| \geq 2.$$

These constraints can be generalized with the $\kappa$-path inequalities

$$x(S) \geq \kappa(S) \qquad \forall S \subseteq N,\ \text{s.t.}|S| \geq 2,$$

where $\kappa(S)$ is the smallest number of vehicles that can feasibly serve the customers in $S$. In the context of the VRPTW, the presence of time windows makes the efficient calculation of $\kappa(S)$ difficult to achieve for a generic subset $S$. Because of this, Kohl et al. (1999) focus on subsets $S$ that require at least two vehicles but are currently serviced by less than two, i.e., subsets $S$ such that

$$1 < \hat{x}(S) < 2 \wedge \kappa(S) \geq 2,$$

where $\hat{x}(S)$ is the value of $x(S)$ in the current solution. In order to check if $\kappa(S) > 1$ for a given subset $S$, it suffices to solve an instance of a TSP with capacity and time window constraints. While this problem is NP-hard, for sets $S$ of reasonable size it is possible to solve it quite efficiently.

Another recent example consists in the *subset-row* inequalities introduced by Jepsen et al. (2008), which are used for the linear relaxation of the model (2.10)-(2.13). These inequalities are defined over subsets of constraints (2.11) in the set partitioning model, and correspond to a subset of the Chvátal-Gomory rank 1 cuts for the set partitioning polytope. They can be defined as follows:

$$\sum_{r\in\Omega} \left\lfloor \frac{1}{l} \sum_{i\in S} a_{ir} \right\rfloor y_r \leq \left\lfloor \frac{|S|}{l} \right\rfloor \qquad \forall S \subseteq N,\ 0 < l < |S|.$$

In their study, Jepsen et al. (2008) only consider inequalities for $|S| = 3$ and $l = 2$, which can be generated by enumeration. Together with 2-path inequalities, subset-row inequalities have successfully been used in a CG framework, although their handling severely complicates the pricing subproblem.

Valid inequalities are of course an important component in branch-and-cut (BC), a well-known method for solving combinatorial optimization problems (see Padberg and Rinaldi, 1987, Mitchell, 2002). It simply consists in a branch-and-bound algorithm that embeds a cutting plane procedure in order to tighten the linear relaxation lower bounds as much as possible.

The effectiveness of this method strongly depends on the quality of the separation algorithms and of the valid inequalities used in the cutting plane phase. Indeed, many sophisticated families of valid inequalities have been developed since Bard et al. (2002) proposed the first BC algorithm tailored for the VRPTW. To name a few, Lysgaard (2006) develops reachability cuts, which strengthen $\kappa$-path inequalities taking into consideration the fact that, due to time windows, only certain arcs can be traversed on a route serving a given customer. Kallehauge et al. (2007) develop a family of strengthened path inequalities that are facet-defining under certain assumptions. Finally, Letchford and Salazar-González (2006) introduce a two-commodity flow formulation for the VRPTW that allows them to derive a new family of valid inequalities, called projection inequalities, since they are obtained by a projection onto the subspace of the arc-flow variables $x_{ij}$.

### 2.2.2 Lagrangian Relaxation

While the Dantzig-Wolfe approach is one of the most popular decomposition methods for routing problems, another such approach for combinatorial optimization problems that has been used for the VRPTW is Lagrangian relaxation. This method relaxes a subset of constraints and penalizes their violations using values called Lagrange multipliers.

Let us consider constraints (2.2) of the VRPTW model, which require that each customer has to be visited once, and the set of multipliers $\boldsymbol{\alpha} = (\alpha_i, i \in N)$, with $\alpha_i \geq 0$ for all $i \in N$. The Lagrangian subproblem $L(\boldsymbol{\alpha})$ is defined as

$$\min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij} x_{ijk} + \sum_{k \in K} \sum_{i \in N} \alpha_i \left( 1 - \sum_{j \in \delta^+(i)} x_{ijk} \right),$$

subject to (2.3)- (2.9).

For any value of $\boldsymbol{\alpha}$, the optimal value of $L(\boldsymbol{\alpha})$ is a dual lower bound for the original VRPTW. The problem of obtaining the multipliers that yield the best lower bound, called the Lagrangian bound $L = \max_{\boldsymbol{\alpha}} L(\boldsymbol{\alpha})$, is a concave non-differentiable maximization problem. The choice of relaxing only the assignment constraints (2.2) is not arbitrary: if the capacity and time window constraints are relaxed together with (2.2), the associated Lagrangian subproblem is a pure network flow problem for which the integrality property holds, and thus the Lagrangian bound would be no better than the linear programming bound.

Examples of application of this decomposition approach to the VRPTW include the studies by Kohl and Madsen (1997), who attempt to determine optimal multiplier values with a combination of a subgradient algorithm and a bundle algorithm, and Kallehauge et al. (2006), who use this decomposition approach to develop a branch-and-cut-and-price algorithm for the VRPTW in which the Lagrangian dual problem is solved with a cutting plane algorithm.

### 2.2.3 Reduced Set Partitioning

A recent promising technique is the one proposed by Baldacci et al. (2010, 2011), which attempts to reduce the size of the VRPTW set partitioning model by fixing to 0 the value of a large number of its variables. It exploits the following property: given an integer program and an upper bound $U$, a non-negative integer variable takes value 0 in every optimal integer solution if its reduced cost with respect to a feasible dual solution of a linear programming relaxation exceeds $U - L$, where $L$ is the lower bound associated with the dual solution.

In the procedure proposed by the authors, the upper bound $U$ is obtained with a metaheuristic, while the lower bound $L$ and the associated dual solution are obtained with a sophisticated dual ascent method that combines Lagrangian relaxation, subgradient optimization, and CG. Furthermore, the proposed dual ascent method uses sequentially up to four heuristics. Once the bounds are obtained, a dynamic programming algorithm is used to obtain all elementary columns that satisfy the above property with respect to $U$ and $L$, and the set partitioning model is solved with a commercial MIP solver.

## 2.3 Heuristic Methods

Heuristics are a class of solution procedures that aim to find relatively good solutions for an optimization problem while using limited computational resources. Given the inherent difficulty of the VRPTW, heuristic solution methods have always been an extremely popular subject of study for this problem. While initially consisting of route construction and improvement algorithms, the landscape has been dominated by metaheuristics since their introduction in recent years. Heuristics see also widespread adoption among industry practitioners, given their capability to quickly obtain solutions of good quality. For these reasons, and also for the sake of completeness, we present here a short overview of this class of algorithms.

### 2.3.1 Route Construction Heuristics

Route construction heuristics are some of the earliest researched solution methods for the VRPTW. They aim to create a feasible solution by repeatedly selecting nodes or arcs and adding them to existing routes. They can be divided into sequential heuristics, which build a single route at a time, and parallel heuristics, which work on multiple routes simultaneously. We list here some of the most notable ones.

One of the earliest methods is one proposed by Solomon (1986), which is a cluster-first, route second method that uses a giant-tour heuristic. Customers are first scheduled in a single tour that is then divided into multiple routes.

In another seminal paper, Solomon (1987) describes several construction heuristics for the VRPTW. These include the following: an extension of the savings heuristic by Clarke and Wright (1964), which is one of the best-known algorithms for the CVRP; a time-oriented nearest-neighbor heuristic; a time-oriented sweep heuristic; and three variants of a sequential two-phase insertion algorithm, which is reportedly the most successful. This algorithm sequentially initializes each route with a "seed" customer, and then iteratively proceeds with two phases. In the first phase, each unrouted customer is assigned its best feasible insertion position in the current route based on the minimum additional distance and time required. In its second phase, the method selects the customer to insert in its best possible position using a maximum savings approach.

A parallel variant of this insertion procedure was introduced by Potvin and Rousseau (1993). Here, the set of routes is initialized all at once, and the selection of the next customer to be inserted is based on a generalized regret measure over all the routes. The employed measure depends on the the gap between the first best insertion position for a customer and its best insertion position in the other routes.

Bramel and Simchi-Levi (1996) develop an asymptotically optimal heuristic based on an idea for solving the capacitated location problem with time windows (CLPTW). In the CLPTW, the aim is to select a subset of possible sites, assign a vehicle to each site, and to assign a subset of customers to each vehicle. In the context of the VRPTW, selecting a set of sites equates to selecting a seed customer for each route. The authors use a method based on Lagrangian relaxation to solve the associated CLPTW instance, and then associate unrouted customers to the seed customers in a greedy order.

Finally, Ioannou et al. (2001) adapt the greedy look-ahead method of Atkinson (1994) by using the insertion procedure of Solomon (1987) mentioned above. This technique aims to minimize the overall impact of the insertion of an unrouted customer on the chosen route and on all the remaining unrouted customers.

Excluding the approach by Ioannou et al. (2001), which returns solutions of better quality in a longer computing time, this class of heuristics is generally very fast, given their relative simplicity. Compared to the more sophisticated route improvement methods, which we discuss in the next section, they maintain a certain advantage with regards to computational resources, while usually returning solutions of poorer quality.

### 2.3.2 Improvement Heuristics

Improvement heuristics, or *local search* methods, form a class of algorithms that attempt to find better solutions to an optimization problem by iteratively modifying the current solution until no further improvement is possible. In this class of heuristics, *operators* and *neighborhoods* play a central role. An operator changes a solution into another by modifying in a certain way a single attribute, and the set of solutions obtained by applying the operator to all possible attributes of the original solution is its associated neighborhood. Thus, local search equates to iteratively exploring the neighborhood of the current solution in order to find a better one. The procedure stops when the current solution is the best one in its neighborhood, or in other words, it is a *local optimum*.

In the context of the VRPTW, and of routing problems in general, neighborhoods are defined by arc-exchange operators. Many of these operators were first introduced for simpler variants of routing problems, like the traveling salesman problem (TSP) or the CVRP. Efficient implementations of these techniques for the VRPTW are reported in Savelsbergh (1985), Solomon and Desrosiers (1988), Solomon et al. (1988), Savelsbergh (1990), and Savelsbergh (1992).

The size of neighborhoods is an important trade-off for local search methods: larger neighborhoods lead to solutions of better quality, but evaluation of all neighboring solutions becomes more time consuming. In fact, for $k$-exchange neighborhoods (which contain the solutions that can be reached from the current one by swapping a subset of $k$ arcs), verifying local optimality requires $O(n^k)$ time. Neighborhoods can thus be classified in two categories (Desaulniers et al., 2014): *traditional* and *large*. The former contains the neighborhoods whose size grows polynomially in a controlled manner, and thus can be evaluated explicitly. Large neighborhoods grow too fast with $n$ to allow explicit evaluation, and have to be evaluated heuristically. They have typically been used in certain metaheuristics, which will be discussed in the next section.

Traditional neighborhoods have been considered since early studies like Russell (1977), Cook and Russell (1978), and Baker and Schaffer (1986). They can be divided into *intra-route* and *inter-route* approaches. In the former, only one route of the current solution is altered. In the latter, new solutions are obtained by moving customers between two or more routes. Intra-route approaches include the basic 2-exchange operator (or *2-opt*), which exchanges two arcs of a single route, and the well-known Or-opt operator introduced by Or (1976), which relocates a chain of consecutive customers by replacing three arcs in the original route. Inter-route approaches include the 2-opt* introduced by Potvin and Rousseau (1995), which exchanges two arcs between two routes (this has the advantage of maintaining the orientation of the routes, as opposed to the 2-opt operator, clearly an important aspect with the addition of time windows), the cross exchange of Taillard et al. (1997), which selects two subpaths of two routes and exchanges their position, and path relocation, which relocates a subpath from a route to another one. This category also includes many more sophisticated techniques, such as the GENI-Exchange operator of Gendreau et al. (1992), the $\lambda$-interchange of Osman (1993), and the cyclic $k$-transfers of Thompson and Psaraftis (1993).

The quality of the solution returned by local search greatly depends on the quality of the initial solution. This is the basic reason why certain studies propose composite heuristics that blend route construction and improvement methods. Examples of this category include Kontoravdis and Bard (1995), who combine a greedy heuristic and randomization to produce initial routes in parallel, and then improve them with local search; Russell (1995), who develops a procedure that embeds route improvement within the construction process; and Cordone and Calvo (2001), who use Solomon's insertion heuristic with a hierarchical local search approach. For an in-depth review of route construction and improvement heuristics, the reader is referred to the survey by Bräysy and Gendreau (2005a).

### 2.3.3 Metaheuristics

An important drawback of local search is that they terminate when they reach a local optimum, which is clearly not guaranteed to be a global optimum, and indeed might not even be a solution of overall good quality. Metaheuristics address this issue by implementing various techniques to escape local optima and effectively explore a much larger solution space, in the attempt to find a near-optimal solution. This class of general solution procedures is at the core of recent developments on heuristic methods for optimization problems, and routing problems have provided fertile ground for research on these algorithms.

A novel aspect of these methods is that, in order to facilitate the exploration of the solution space, they generally allow deteriorating and even infeasible intermediary solutions during the search process. In the context of the VRPTW, the types of infeasibility that are generally allowed are time window, vehicle capacity, and customer violations (i.e., when some customers are not visited). The first two are typically considered together in a penalized objective function.

For the VRPTW, we can identify two broad categories of metaheuristics, which we briefly overview here: *single trajectory search* and *population-based search*. For a more in-depth overview, the reader is referred to the survey by Bräysy and Gendreau (2005b).

### Single trajectory search

In single trajectory search, each iteration only considers one solution at a time, and thus the search process generates a sequence of solutions that can be seen as a trajectory along the solution space. This class of metaheuristics includes well-known procedures like tabu search, large neighborhood search, and iterated local search.

Tabu search, first introduced by Glover (1986), explores the solution space by iteratively moving from the current solution $s$ to the best solution in a certain subset of its neighborhood $N(s)$. Thus, the new solution might be of poorer quality than $s$, in which case it is accepted only to avoid trajectories already investigated, and ultimately to escape local optima. Furthermore, in order to avoid cycling, solutions possessing certain features of recently explored solutions are temporarily declared forbidden, or tabu, for a certain number of iterations. Tabu search was first adapted for the VRPTW by Garcia et al. (1994), who presented a parallel implementation utilizing Solomon's insertion framework for initial solutions and 2-opt* and Or-opt operators for local search.

More recent examples include the studies by Cordeau et al. (2001, 2004). The authors use an initial solution obtained with a sweep heuristic and use customer relocation as the search operator. When removing a customer from a route, its reassignment to the same route is declared tabu. Additionally, this algorithm allows intermediary solutions that are infeasible with respect to both the time windows and vehicle capacity, and terminates by performing a post-optimization procedure applying to each route a heuristic designed for the traveling salesman problem with time windows.

Variable neighborhood search (VNS), originally proposed by Mladenović and Hansen (1997), essentially consists in a local search method that oscillates between several neighborhood structures. More precisely, whenever a local optimum is reached when performing local search according to a certain operator, the algorithm switches over to a different one in order to search for improving solutions. Indeed, a solution that is locally optimal with respect to a certain neighborhood structure might not be so for other structures in general, since this is guaranteed to be true only for a global optimum.

Examples of applications of this paradigm to the VRPTW include the works of Rousseau et al. (2002), who use a VNS scheme with several new operators (including the GENI procedure of Gendreau et al. (1992), ejection chains, and the SMART operator, which removes arcs from the solution instead of customers) within a constraint programming framework, and Bräysy (2003), who proposes a multi-phase algorithm in which VNS is used with modified cross-exchange and cheapest insertion heuristic. Also, Repoussis et al. (2006) propose a multi-start metaheuristic that combines greedy randomized adaptive search procedures (GRASP) with a hybrid variable neighborhood tabu search, where tabu search is used as an internal component of VNS. In this method, VNS iteratively selects a random solution of the current neighborhood and performs local search. If the best solution does not change after the current iteration, a different neighborhood structure is chosen and a new search is performed, until a termination condition is met. The study of de Armas and Melián-Batista (2015) proposes a similar approach for a dynamic rich VRPTW. The sequence of operators used by their method includes GENI, Or-opt, cross-exchange, 2-opt, customer relocation, and swapping.

Large neighborhood search (LNS) is a heuristic originally developed for routing problems by Shaw (1998). As mentioned above, while large neighborhoods would provide solutions of better quality in a steepest descent algorithm, their size grows with $n$ too quickly to explicitly evaluate all solutions in them. Thus, they need to be explored heuristically. The main attributes of LNS, which allow it to work on large neighborhoods, are the *destroy* and *repair* operators. At each iteration, a destroy operator partially disintegrates a solution, for instance by removing certain customers from their routes, and a repair operator rebuilds it, for instance by reinserting all unrouted customers, thus obtaining a temporary solution. Then, a criterion is employed in order to determine if the temporary solution is retained for the next iteration, for instance if it is better than the current solution. An example of the application of LNS to the VRPTW is the work of Pisinger and Ropke (2007), who transform each instance of the problem to the pickup-and-delivery problem with time windows, which is then solved by the LNS proposed by Ropke and Pisinger (2006). This method adopts several operators, including simple greedy remove and insert. The algorithm dynamically adapts itself to prefer the most successful operators and is therefore called *adaptive* LNS (ALNS).

Iterated local search, introduced by Lourenço et al. (2010), is a relatively simple metaheuristic framework, which is based on a series of local searches. At each iteration, the algorithm performs a random perturbation of the current solution to obtain a temporary one, and then uses local search to improve the temporary solution until a stopping criterion is met. Instead of local search, one might employ a more sophisticated metaheuristic, like tabu search or simulated annealing. An example of this kind of iterated local search for the VRPTW is the study by Cordeau and Maischberger (2012), which instead of local search uses a version of the tabu search algorithm of Cordeau et al. (2001, 2004). In this example, the perturbation step is based on large neighborhood search, and consists in removing a cluster of customers, which are then reinsterted in a random order using a cheapest-insertion policy.

### Population-based search

Algorithms of this category do not consider only a single solution per iteration, but instead maintain a pool of solutions throughout execution. The set of population-based methods for the VRPTW is mostly composed of evolutionary algorithms and path-relinking algorithms.

Evolutionary algorithms include the well-known class of genetic algorithms, introduced by Holland (1992), and memetic algorithms (Moscato and Cotta, 2010), which extend genetic algorithms by hybridizing them with other heuristics, such as local search. At each iteration of the classical genetic algorithm, a set of new solutions $S$ is formed by combining ones from the existing population $P$, in a step called *crossover*. A subset of $S$ is then randomly perturbed in the *mutation* phase, obtaining $S'$. The pool $P$ is then updated with solutions from $S'$ according to a certain criterion. In a memetic algorithm, the solutions in $S'$ are first improved with local search or another heuristic before updating the pool $P$. A crucial component in the effectiveness of these algorithms is the crossover step. Some of the most successful crossover operators include: the EAX crossover, used originally for the TSP by Nagata (1997), the OX crossover (Falkenauer and Bouffouix, 1991), and multiparent recombination (Repoussis et al., 2009).

Path-relinking is a novel idea by Hashimoto and Yagiura (2008) that treats the pool of solutions differently from evolutionary algorithms. At each iteration, two solutions $s_a$ and $s_b$ from the pool are selected randomly, and the algorithm then transforms $s_a$ into $s_b$ with a series of 2-opt* and Or-opt operations. Some of the intermediary solutions are then improved with local search and then reinserted in the pool.

### 2.3.4 Matheuristics

Over the last few decades, the progress on exact solution methods and the increasing availability of powerful computer hardware has made it possible to solve large MIPs in reasonable amounts of time. Because of this, several recent studies have introduced the idea of combining heuristics with exact methodologies, or more specifically, with mathematical programming techniques, thus coining the term *matheuristics*.

Archetti and Speranza (2014) survey the recent matheuristics developed for routing problems, and classify these methods into three groups: decomposition approaches, improvement heuristics, and BP/CG-based approaches. In the first group, a heuristic decomposes the problem in smaller subproblems, which are then solved with mathematical programming, such as the cluster-first-route-second approach for the VRP. In improvement approaches, the quality of a solution found by a heuristic is improved by solving a MIP model. This can either be done as the final phase of the solution method, or as a tool to explore the solution space and improve the solution within a local search procedure. Finally, BP/CG-based approaches exploit the structure of a BP scheme to produce heuristic solutions.

To the best of our knowledge, studies that apply a matheuristic approach to the VRPTW, or one of its variants, usually fall under the latter category, with the exception of the study by Prescott-Gagnon et al. (2009), which consists in an improvement approach. Here, the authors propose an ALNS that uses four heuristics in the destruction phase and a single reconstruction procedure based on the heuristic BP algorithm proposed by Desaulniers et al. (2008). Here, BP is used to determine the optimal set of routes servicing the customers that have been removed during the destruction phase.

Examples of BP-based matheuristics for the VRPTW include the following. Danna and Le Pape (2005) propose a collaboration scheme between BP and metaheuristics with the objective to obtain good quality solutions early in the execution of the algorithm. The authors use an LNS procedure and a guided tabu search for two purposes. First, they generate an initial solution for the BP algorithm. Then, at certain points during the execution of BP, they call the heuristics to improve the incumbent solution.Desaulniers et al.

(2008) use tabu search to solve the pricing problem within a BCP method. Their algorithm uses customer insertion and deletion as search operators, and performs a multiple start procedure to diversify the search. Finally, Pillac et al. (2013) study an extension of the VRPTW, called the technician routing and scheduling problem, where personnel is routed to perform work on a set of tasks, taking into account constraints such as time windows, technician skills, tools, and available spare parts. Here, an initial solution is generated with a constructive heuristic, which is then improved with an ALNS procedure. The columns generated by the ALNS are then used in a set covering model that is solved to optimality.

For more information on matheuristics for routing problems, see the surveys of Doerner and Schmid (2010), Ball (2011) and Archetti and Speranza (2014).

## 2.4 Conclusions

In this chapter, we have offered a brief overview of the vast body of work on the VRPTW, by describing some of its formulations and presenting some highlights in the rich landscape of solution methods, both heuristic and exact.

# Chapter 3

# Branch-and-Price Algorithms for the VRPTW

As we have mentioned earlier, branch-and-price (BP) is a leading methodology for solving exactly rich vehicle routing problems. Simply put, BP consists in a branch-and-bound (BB) algorithm in which the linear relaxation associated with each node of the BB tree is solved with column generation (CG). In this chapter, we describe its main characteristics, as well as several refinements that are an integral part of the algorithms under consideration in this study. In Section 3.1, we give a brief overview of the CG method for the VRPTW, and the associated pricing problem. In Section 3.2, we describe the dynamic programming algorithm for the pricing problem used in all the methods considered in this study. Sections 3.3 and 3.4 describe two important extensions of the dynamic programming algorithm, while Section 3.5 introduces possible hybridizations of these extensions. Finally, Section 3.6 gives a quick overview of BB, with some specific considerations regarding the VRPTW. For additional details on the implementation of BP for routing problems, see the tutorial of Feillet (2010).

## 3.1 Column Generation

The history of CG goes back to the 1960s, with the introduction of the decomposition principle by Dantzig and Wolfe (1960). Here, the original linear program (LP) is reformulated as a combination of a master program and a number of independent subprograms. The new variables represent the extreme points and rays of the polyhedra associated to the subprograms, and therefore they are usually exponential in number. In their seminal papers,

**Algorithm 3.1** Basic CG.

```
1: procedure CG
2:     Initialize RMP
3:     Optimal ← false
4:     while Optimal == false do
5:         Solve RMP
6:         Solve pricing problem
7:         if No negative reduced cost column found then
8:             Optimal ← true
9:         else
10:            Add column(s) to RMP
11:    return RMP solution
```

Gilmore and Gomory (1961, 1963) introduced a formulation for the cutting stock program that treated implicitly a huge number of variables. Since then, this paradigm has been applied in many scenarios, especially ones that are prone to include a huge number of variables, like routing problems.

Let us consider the set partitioning formulation for the VRPTW (2.10)-(2.13) described in Chapter 2. As we have mentioned, this formulation has the advantage of providing a relatively strong lower bound, which is obviously useful in a BB procedure. Unfortunately, since there is one variable per feasible route, trying to solve the linear relaxation of (2.10)-(2.13) is clearly intractable. CG aims to address this issue.

It is possible to replace the set partitioning formulation with a set covering one, in which we simply replace constraints (2.11) with $\sum_{r\in\Omega} a_{ir} y_r \geq 1$, i.e., simply impose that each customer has to be visited at least once. This does not impact the optimal solution, since it is easy to see that under optimality no customer would be visited more than once, due to the triangle inequality. This modification is usually desirable since it forces the associated dual variables to be non-negative, which is a favorable condition for the convergence of CG algorithms. Let us then consider the linear relaxation of the set covering variant of the formulation for the VRPTW (2.10)-(2.13), which we call in this context the Master Problem (MP), and let us introduce an initial subset of variables $\overline{\Omega} \subseteq \Omega$.

Then the Restricted Master Problem (RMP) is simply the restriction of the MP to the subset $\overline{\Omega}$:

$$\min \sum_{r\in\overline{\Omega}} c_r y_r \tag{3.1}$$

$$\text{s.t.} \sum_{r\in\overline{\Omega}} a_{ir} y_r \geq 1, \qquad \forall i \in N, \tag{3.2}$$

$$y_r \geq 0, \qquad \forall r \in \overline{\Omega}. \tag{3.3}$$

The associated dual program is the following, where $\eta_i$ is the dual variable associated with customer $i$:

$$\max \sum_{i\in N} \eta_i \tag{3.4}$$

$$\text{s.t.} \sum_{i\in N} a_{ir} \eta_i \leq c_r, \qquad \forall r \in \overline{\Omega}, \tag{3.5}$$

$$\eta_i \geq 0, \qquad \forall i \in N. \tag{3.6}$$

If there is no column with negative reduced cost in $\Omega \setminus \overline{\Omega}$, i.e., columns such that

$$c_r - \sum_{i\in N} a_{ir}\eta_i < 0, \tag{3.7}$$

then the optimal solution for the RMP is also optimal for the MP. Hence, the main loop of CG, illustrated in Algorithm 3.1, is the following. First, we initialize the restricted set of feasible columns $\overline{\Omega}$. Since we have no constraint on the fleet size, we simply initialize it with the set of trivial columns, i.e., the routes that visit only a single customer. Whenever we solve the RMP to optimality, we look for columns in $\Omega\setminus\overline{\Omega}$ with negative reduced cost to include in $\overline{\Omega}$, and then we reoptimize the RMP. Incidentally, finding such a column amounts to finding a constraint of type (3.5) in $\Omega\setminus\overline{\Omega}$ that the current optimal solution for the dual program (3.4)-(3.6) violates. The pricing problem we need to solve in order to find such columns is then an elementary shortest path problem with resource constraints (ESPPRC), and can be formulated in the following way:

$$\min\ c_r - \sum_{i\in N} a_{ir}\eta_i \tag{3.8}$$

$$\text{s.t. } r \in \Omega. \tag{3.9}$$

In the next section, we describe the most well-known exact algorithms for the ESPPRC, which forms the basis for the ones under consideration in this study.

## 3.2 Labeling Algorithm for the Pricing Problem

As we have mentioned above, we need to solve an ESPPRC, which consists in finding a route with the least reduced cost. This route has to be feasible with respect to the VRPTW, i.e., it has to be a path starting and ending at the depot, visiting a subset of customers in $N$, and respecting the capacity, time window, and elementarity constraints. As we have mentioned in Chapter 2, the depot is split in two identical copies, the source, denoted as 0, and the sink, denoted as $n+1$. The former only has exiting arcs in the associated graph $G = (V, A)$, where $V = N \cup \{0, n+1\}$, and the latter only entering arcs. Each feasible path is then a source-sink path that respects the aforementioned constraints. Here, we define an elementary path as a path that does not visit the same customer more than once.

Since the underlying graph may have negative cost cycles, given the definition of reduced cost (3.7), and since we require that the solution be an elementary path, it is impossible to use classic algorithms for the shortest path problem such as Dijkstra's or Bellman-Ford. Furthermore, it is possible to prove that the ESPPRC is in fact NP-hard: Dror (1994) proved it by showing that the NP-hard problem of sequencing within intervals (SWI) can be polynomially transformed into the ESPPRC.

In the first implementation of BP for the VRPTW, Desrochers et al. (1992) did not in fact consider the ESPPRC as the pricing subproblem, given its difficulty. Indeed, they observed that by relaxing the elementarity constraints, the resulting shortest path problem with resource constraints (SPPRC) admits a pseudo-polynomial procedure, which is an extension of the Bellman-Ford dynamic programming algorithm. To accommodate this change, they modify model (3.1)-(3.3) by letting parameter $a_{ir}$ in constraint (3.2) indicate the number of times customer $i$ is visited in route $r$. Their procedure is detailed in the next subsection.

### 3.2.1 Basic Labeling Algorithm

In this algorithm, a set of *states* is associated with each vertex. Each state represents a feasible partial path $P$ starting at source $v_0 = 0$, visiting $i$ customers and ending at vertex $v_i$, which we describe with the notation $P = (0 = v_0, v_1, \ldots, v_i)$. With each state we associate a cost $C_{v_i}$ and a resource vector $\mathbf{R}_{v_i}$, in which each component represents the consumption of a different resource along the partial path. Cost $C_{v_i}$ of partial path $P$ is defined as $\sum_{(v_k, v_{k+1}) \in P} c_{v_k, v_{k+1}} - \sum_{v_k \in P} \eta_{v_k}$, since the cost of a path ending at the sink, i.e., a complete path, has to correspond to its reduced cost. While vector $\mathbf{R}_{v_i}$ can accommodate different resources depending on the additional constraints of the routing problem under consideration (for instance, whether or not there are pick-ups in addition to deliveries), in the case of the VRPTW we consider two resources: the path duration $\tau_{v_i}$ and accumulated vehicle load $q_{v_i}$. More specifically, $\tau_{v_i}$ denotes the timespan between the departure from the source and the start of service at customer $v_i$. Each resource is used to monitor feasibility of the partial path with regard to the vehicle capacity constraints and the time window constraints, respectively. Each state can therefore be represented by a *label* $l_{v_i}$, which is a tuple of the form $l_{v_i} = (C_{v_i}, \mathbf{R}_{v_i})$. In the following, we expand this notation to $l_{v_i} = (C_{v_i}, \tau_{v_i}, q_{v_i})$. For simplicity, we also denote index $v_i$ as $i$ in the following, e.g., we denote $l_{v_i}$ as $l_i$.

The dynamic programming algorithm iteratively extends each label $l_i$ along all feasible arcs $(i, j) \in A$ to obtain new states, according to the resource extension functions (REFs) explained below. If a newly computed state is not feasible with respect to the problem constraints, it is discarded. The algorithm is initialized with a single label at the source $l_0 = (0, 0, 0)$, representing the empty path, and stops when all possible label extensions have been performed. If there are labels associated with the sink $n + 1$, it then returns label $l^*_{n+1} = (C^*_{n+1}, \tau^*_{n+1}, q^*_{n+1})$ with the least cost $C^*_{n+1}$, which corresponds to a feasible complete path with the least reduced cost.

### Label Extension Rules

In order to extend a label $l_i = (C_i, \tau_i, q_i)$ along an arc $(i, j) \in A$ to obtain label $l_j = (C_j, \tau_j, q_j)$, the following REFs are adopted:

$$C_j = C_i + c_{ij} - \eta_j, \tag{3.10}$$

$$\tau_j = \max\{\tau_i + s_i + t_{ij}, a_j\}, \tag{3.11}$$

$$q_j = q_i + d_j;. \tag{3.12}$$

Thus, label $l_j$ is feasible if and only if $\tau_j \leq b_j$ and $q_j \leq Q$.

### Label Domination

A crucial aspect in these label extension algorithms is that their computational complexity heavily depends on the proliferation of labels throughout their execution. This is easy to see, since at every iteration, the algorithm attempts all possible extensions for each existing label. Therefore it is crucial to implement techniques that limit as much as possible label proliferation, such as dominance rules. Let $l^1_i = (C^1_i, \tau^1_i, q^1_i)$ and $l^2_i = (C^2_i, \tau^2_i, q^2_i)$ be two labels associated with two partial paths ending at customer $i$. If the following conditions hold true:

$$C_i^1 \leq C_i^2, \tag{3.13}$$

$$\tau_i^1 \leq \tau_i^2, \tag{3.14}$$

$$q_i^1 \leq q_i^2, \tag{3.15}$$

with at least one inequality being strict, then extending label $l_i^2$ cannot lead to the optimal solution, since any possible extension from $l_i^2$ would be guaranteed to be worse than the same extension from $l_i^1$. We say that label $l_i^2$ is then *Pareto-dominated*, or just *dominated*, by $l_i^1$, and thus we can freely discard it.

### Enforcing Path Elementarity

In the implementation of Desrochers et al. (1992), while not explicitly enforcing elementary routes, the authors exclude paths with 2-cycles, i.e., paths with cycles composed of two arcs. This technique necessitates only a minor change in the dynamic programming algorithm, since it suffices to record in each label the vertex previously visited. Moreover, it provides a substantial advantage in terms of the number of possible paths removed from the formulation and bound quality, and became a standard implementation technique in subsequent studies.

Given the efficiency of the algorithm by Desrochers et al. (1992), Feillet et al. (2004) consider the tractability of its application to the original ESPPRC. In order to adapt the labeling algorithm, the authors add a new set of *elementarity* resources $(E_i^k)_{k \in N}$ (henceforth denoted as $(E_i^k)_k$) to redefine labels, which thus adopt the form $l_i = (C_i, \tau_i, q_i, (E_i^k)_k)$. For each customer $k \in N$, resource $E_i^k$ assumes value 1 if customer $k$ is visited by the associated partial path, and 0 otherwise. We then introduce, in addition to functions (3.10)—(3.11), the following REF to be employed when extending label $l_i = (C_i, \tau_i, q_i, (E_i^k)_k)$ along arc $(i, j)$ to obtain label $l_j = (C_j, \tau_j, q_j, (E_j^k)_k)$:

$$E_j^k = \begin{cases} E_i^k + 1, & \text{if } k = j, \\ E_i^k, & \text{if } k \neq j. \end{cases} \tag{3.16}$$

Label $l_j$ corresponds to an elementary path if and only if $E_i^k \leq 1 \ \forall k \in N$, otherwise, we eliminate it. Analogously, we add to the established dominance rules (3.13)-(3.14) the following ones:

$$E_i^{k,1} \leq E_i^{k,2}, \forall k \in N. \tag{3.17}$$

**Algorithm 3.2** Monodirectional Dynamic Programming.

```
1: procedure MDP
2:     // Initialization //
3:     $\Gamma_0 \leftarrow \{(0, 0, 0, \mathbf{0})\}$
4:     for all $i \in V \setminus \{0\}$ do
5:         $\Gamma_i \leftarrow \emptyset$
6:     $E \leftarrow \{0\}$
7:     // Search //
8:     repeat
9:         Select $i \in E$
10:        // Extension //
11:        for all $l_i = (C_i, \tau_i, q_i, (E_i^k)_k) \in \overline{\Gamma}_i$ do
12:            for all $j \in \delta_i^+$ such that $E_i^j = 0$ do
13:                $l_j \leftarrow \textsc{Extend}(l_i, j)$
14:                if $l_j$ is feasible then
15:                    $\Gamma_j \leftarrow \text{EFF}(\Gamma_j, l_j)$
16:                if $\overline{\Gamma}_j \neq \emptyset$ and $j \neq n + 1$ then
17:                    $E \leftarrow E \cup \{j\}$
18:        $E \leftarrow E \setminus \{i\}$
19:    until $E = \emptyset$
20:    return best path in $\Gamma_{n+1}$
```

Furthermore, Feillet et al. (2004) developed an improved use of the elementarity resources. When a customer cannot be visited in any feasible extension of a given label because of the resource constraints, we denote such customers as *unreachable*. The authors observed that by setting the consumption of the elementarity resources associated with such customers to 1, as if they had already been visited, allows the algorithm to fathom a larger number of states without losing the guarantee of optimality.

Algorithm 3.2 illustrates the labeling algorithm as described so far. Here, $\Gamma_i$ denotes the set of labels associated with customer $i \in N$, $\overline{\Gamma}_i \subseteq \Gamma_i$ is the set of labels that have not yet been extended, $E$ is the set of customers to evaluate, and $\delta_i^+$ is the set of successors of $i$, i.e., $\delta_i^+ = \{j \in V | (i, j) \in A\}$. Procedure $\textsc{Extend}(l_i, j)$ extends label $l_i$ to vertex $j$ according to the given rules, checks feasibility with respect to the resource constraints, and adjusts the elementarity resource consumption of unreachable customers. Finally, Procedure $\text{EFF}(\Gamma_i, l_i)$ compares the newly generated label $l_i$ with all the ones existing in $\Gamma_i$ using the dominance rules: if label $l_i$ is dominated by any of the labels in $\Gamma_i$, it is discarded; otherwise it is inserted in $\overline{\Gamma}_i$ and if any other label in $\Gamma_i$ is dominated by $l_i$, those are discarded instead.

### 3.2.2 Bounded Bidirectional Search

The complexity of the labeling algorithm depends on the number of labels that are generated throughout its execution. It is easy to observe that in the worst case, the number of labels generated at each execution is exponential with the number of arcs in the path, since each time a label associated to customer $i$ is extended, the algorithm can generate as many labels as the number of successors of $i$. The size of the problem and the tightness of its constraints both play a role in this, since they clearly affect the maximum possible length of a feasible route.

Because of this property, Righini and Salani (2006) propose to use *bidirectional* label extension and *bounding* in order to limit the length of the generated partial paths without losing guarantee of optimality. Intuitively, this technique consists in performing label extensions simultaneously from the source (as we have described above) and backwards from the sink, while using certain bounds to limit path length to half of what they would normally be. In the following, we describe these phases in detail.

#### Backward Label Extensions

Let us denote with $\Gamma_i^f$ and $\Gamma_i^b$ the sets of forward and backward labels associated with vertex $i \in V$. In order to perform label extension backwards from the sink, we can keep the same label structure as in the forward extension case. Thus, label $l_i^b = (C_i^b, \tau_i^b, q_i^b, (E_i^k)_k^b) \in \Gamma_i^b$ represents a partial path starting from vertex $i$ and ending at sink $n+1$, with all elements in $l_i$ denoting the same quantities as in the forward case. In particular, $\tau_i^b$ denotes the elapsed time between the end of service time at $i$ and the arrival at the sink. The algorithm is initialized with label $l_{n+1}^b = (0, 0, 0, \mathbf{0}) \in \Gamma_{n+1}^b$, representing the empty path at the sink, and each label in $\Gamma_i^b$ is extended along the predecessors of $i$, i.e., the arcs in the set $\delta_i^- = \{j \in V | (j, i) \in A\}$.

To be able to perform backward search, we now simply need to appropriately modify the REFs. In fact, it suffices to change the REFs related to the path duration, since this is the only resource that is generally not symmetrical with respect to the time horizon. For this purpose, Righini and Salani (2006) introduce the *backward time window* $[a_i^b, b_i^b]$ for each vertex $i \in V$, with $a_i^b := a_i + s_i$ and $b_i^b := b_i + s_i$, representing the range of feasible departure times from $i$. Let us also denote with $T$ the latest feasible arrival time at the sink, i.e., $T = \max_{i \in V}\{b_i + s_i + t_{i,n+1}\}$.

Thus, when extending label $l_i^b$ to label $l_j^b = (C_j^b, \tau_j^b, q_j^b, (E_j^k)_k^b)$ along arc $(j, i) \in A$ we adopt the following REF regarding resource $\tau_j^b$:

$$\tau_j^b = \max\{\tau_i^b + s_i + t_{ji}, T - b_j^b\}, \tag{3.18}$$

with $l_j^b$ being feasible with respect to its duration if and only if $\tau_j^b \leq T - a_j^b$, since $\tau_j^b$ represents the time span between the departure from $j$ and the arrival at $t$. The remaining REFs are the same as in the forward case:

$$C_j^b = C_i^b + c_{ji} - \eta_j; \tag{3.19}$$

$$q_j^b = q_i^b + d_j; \tag{3.20}$$

$$E_j^{k,b} = \begin{cases} E_i^{k,b} + 1, & \text{if } k = j, \\ E_i^{k,b}, & \text{if } k \neq j, \end{cases} \tag{3.21}$$

where label $l_j^b$ is feasible with respect to capacity and elementarity if and only if $q_j^b \leq Q$ and $E_j^{k,b} \leq 1$, for all $k \in N$. The dominance rules are also the same as in the forward case: given two backward labels $l_i^{b,1} = (C_i^{b,1}, \tau_i^{b,1}, q_i^{b,1}, (E_j^k)_k^{b,1})$ and $l_i^{b,2} = (C_i^{b,2}, \tau_i^{b,2}, q_i^{b,2}, (E_j^k)_k^{b,2})$ associated with two partial paths ending at customer $v_i$, the former dominates the latter if

$$C_i^{b,1} \leq C_i^{b,2}, \tag{3.22}$$

$$\tau_i^{b,1} \leq \tau_i^{b,2}, \tag{3.23}$$

$$q_i^{b,1} \leq q_i^{b,2}, \tag{3.24}$$

$$E_i^{k,b,1} \leq E_i^{k,b,2}, \ \forall k \in N, \tag{3.25}$$

with at least one inequality being strict.

### Label Concatenation

In order to obtain a feasible complete path by concatenating a forward label $l_i^f = (C_i^f, q_i^f, \tau_i^f, (E_i^k)_k^f)$ and a backward label $l_j^b = (C_j^b, q_j^b, \tau_j^b, (E_j^k)_k^b)$ along arc $(i, j) \in A$, we need to compute the total reduced cost $\overline{C}$, the total vehicle load $\overline{q}$, and the total duration $\overline{\tau}$ in the following way:

$$\overline{C} = C_i^f + C_j^b + c_{ij}, \tag{3.26}$$

$$\overline{\tau} = \tau_i^f + s_i + t_{ij} + s_j + \tau_j^b, \tag{3.27}$$

$$\overline{q} = q_i^f + q_j^b. \tag{3.28}$$

The label concatenation is feasible if and only if $\overline{q} \leq Q$, $\overline{\tau} \leq T$. Furthermore, we need to impose that the same customer cannot be visited by both partial paths, i.e.,

$$E_i^{k,f} + E_j^{k,b} \leq 1, \ \forall k \in N. \tag{3.29}$$

### Bounding

When using bidirectional search, we have to impose an additional fathoming condition on label extensions, otherwise we would simply generate twice as many states, compared to the original algorithm. Righini and Salani (2006) describe two different techniques to achieve this: *arc bounding* and *resource bounding*.

Arc bounding consists in computing for each label an upper bound on the number of arcs that can be added to the associated path without violating the resource constraints. This can be achieved by formulating and solving an instance of a multi-knapsack problem. We do not consider this technique in our study, since it has been shown to be outperformed by the second one.

Resource bounding consists in selecting a resource, denoted as the *critical* resource, whose consumption is monotone along the paths. Then, we do not extend the labels in which at least half of the available amount of this resource has been already consumed. In the case of the VRPTW, Righini and Salani (2006) propose to designate as the critical resource the total path duration $\tau$, for which the total available amount corresponds to the latest feasible arrival time at the sink $T$. Thus, this corresponds to the following stopping condition: extend only states such that $\tau < T/2$.

### Duplicate Elimination

After having performed label extensions, concatenation is performed iterating over the feasible arcs $(i, j) \in A$. In this phase, there is the risk of generating the same path more than once. For instance, if a feasible path contains the subsequence of vertices $(i, j, k)$, it can be obtained by concatenating a pair of labels along arc $(i, j)$ or arc $(j, k)$. To avoid producing duplicate paths in this phase, we apply a method also described by Righini and Salani (2006), which takes into account the critical resource used for bounding the extension procedure. Among all possible ways to produce the same $s-t$ path, we choose the one produced in the point where the forward and backward consumptions of the critical resource are as close as possible to the half of the overall consumption of that path. Let $\tau_i^f$ and $\tau_j^b$ indicate the consumption of the critical resource on the forward partial path ending at $i$ and the backward

**Algorithm 3.3** Duplicate elimination.

```
1: procedure HALFWAY($l_i^f, l_j^b$)
2:     Take $\tau_i^f$ from $l_i^f$
3:     Take $\tau_j^b$ from $l_j^b$
4:     $\Phi_{(i,j)} = |\tau_i^f - \tau_j^b|$
5:     if $\tau_i^f < \tau_j^b$ then
6:         $\tau_j^f = \tau^f(l_i^f, j)$
7:         Take $\tau_{j+1}^b$ from $l_{j+1}^b$
8:         $\Phi_{(j,j+1)} \leftarrow |\tau_j^f - \tau_{j+1}^b|$
9:         if $\Phi_{(i,j)} < \Phi_{(j,j+1)}$ then
10:            return true
11:        else
12:            return false
13:    else
14:        $\tau_i^b = \tau^b(l_j^b, i)$
15:        Take $\tau_{i-1}^f$ from $l_{i-1}^f$
16:        $\Phi_{(i-1,i)} \leftarrow |\tau_{i-1}^f - \tau_i^b|$
17:        if $\Phi_{(i,j)} \leq \Phi_{(i-1,i)}$ then
18:            return true
19:        else
20:            return false
```

partial path ending at $i$, respectively. Then we accept the path only if it is produced on an arc $(i, j)$ where $\Phi_{(i,j)} = |\tau_i^f - \tau_j^b|$ is minimum.

The procedure HALFWAY($l_i^f, l_j^b$) is described in Algorithm 3.3. In this procedure, we indicate with $i - 1$ the predecessor of $i$ in the partial path associated with label $l_i^f$, and with $j + 1$ the successor of $j$ in the one associated with $l_j^b$. Function $\tau^f(l_i^f, j)$ determines the forward critical resource consumption at vertex $j$ by using the associated REF on label $l_i^f$, and function $\tau^b(l_j^b, i)$ the backward one at $i$ from label $l_j^b$. Since these functions take constant time to return their values, function HALFWAY($l_i^f, l_j^b$) takes constant time as well.

The bounded bidirectional search procedure with all the components described so is given in Algorithm 3.4, as presented by Righini and Salani (2006). Here, the procedure FEASIBLE($l_i^f, l_j^b$) checks if the two labels can generate a feasible path, while the procedure CONCATENATE($l_i^f, l_j^b$) performs the concatenation and returns the resulting path, which is added to the set

$\Pi$ of feasible paths with negative reduced cost.

## 3.3 Decremental State Space Relaxation

Feillet et al. (2004) observed that the addition of elementarity resources did indeed improve the quality of the bounds, but at a severe cost in terms of computational complexity. Because of this, and given the success of the 2-cycle elimination technique used by Desrochers et al. (1992), other authors started experimenting with alternative techniques to handle the elementarity constraints. For instance, Irnich and Villeneuve (2006) consider the removal of $k$-cycles, performing experiments for $k = 3$ and $k = 4$. In this section, we consider one of the first of such techniques that behave dynamically, developed by Righini and Salani (2008): decremental state space relaxation (DSSR). The underlying concept of DSSR was simultaneously developed by Boland et al. (2006), who called it *state space augmenting* algorithm. Here, we are going to predominantly use the notation and the algorithmic structure described by the former authors.

State space relaxation for dynamic programming algorithms was originally described by Christofides et al. (1981b), and it consists in projecting the state space $\mathcal{S}$ explored by the algorithm onto a lower dimensional space $\mathcal{T}$ in such a way that every state in $\mathcal{T}$ keeps the best cost of the states in $\mathcal{S}$ that are mapped to it. The reason behind this technique is that a lower dimensional space is faster to explore; however the search in a relaxed state space does not guarantee feasibility, since infeasible solutions in $\mathcal{S}$ may be projected onto feasible ones in $\mathcal{T}$. In the context of the VRPTW, state space relaxation can be applied by relaxing the elementarity constraint on the vertices, thus allowing cycles in the solutions. Therefore, it is equivalent to solving the SPPRC, as in Desrochers et al. (1992) (without 2-cycle elimination).

In DSSR, the aim is to generalize the aforementioned technique by dynamically transitioning from a complete state space relaxation to an exact algorithm for the ESPPRC. Intuitively, we progressively "activate" as many of the binary resources as necessary in order to ensure the elementarity of the optimal path. The algorithm works as follows.

We maintain a set $\Theta$ of *critical* vertices, that is initialized as empty (or with a certain subset of vertices, according to the strategies described in Section 3.3.2), and at each iteration of the label extension algorithm we only use the elementarity resources $E^k$ such that $k \in \Theta$. Thus, we explicitly forbid multiple visits only to the vertices in $\Theta$, potentially allowing them on the others. If at the end of an iteration there are no paths with negative reduced cost, the algorithm stops. Otherwise, if there are non-elementary solutions

**Algorithm 3.4** Bounded bidirectional dynamic programming.

```
1: procedure BBDP
2:     // Initialization //
3:     $\Gamma_0^f \leftarrow \{(0,0,0,\mathbf{0})\}$, $\Gamma_{n+1}^b \leftarrow \{(0,0,0,\mathbf{0})\}$, $\Pi \leftarrow \emptyset$
4:     for all $i \in V \setminus \{0\}$ do
5:         $\Gamma_i^f \leftarrow \emptyset$
6:     for all $i \in V \setminus \{n+1\}$ do
7:         $\Gamma_i^b \leftarrow \emptyset$
8:     $E \leftarrow \{0, n+1\}$
9:     // Search //
10:    repeat
11:        Select $i \in E$
12:        for all $l_i^f = (C_i, \tau_i, q_i, (E_i^k)_k) \in \overline{\Gamma}_i^f$ do // Forward extension //
13:            for all $j \in \delta_i^+$ such that $E_i^j = 0$ do
14:                $l_j \leftarrow \textsc{Extend}^f(l_i, j)$
15:                if $l_j$ is feasible then
16:                    $\Gamma_j^f \leftarrow \mathrm{EFF}(\Gamma_j^f, l_j)$
17:                if $\overline{\Gamma}_j^f \neq \emptyset$ then
18:                    $E \leftarrow E \cup \{j\}$
19:        for all $l_i^b = (C_i, \tau_i, q_i, (E_i^k)_k^b) \in \overline{\Gamma}_i^b$ do // Backward extension //
20:            for all $k \in \delta_i^-$ such that $E_i^{k,b} = 0$ do
21:                $l_k^b \leftarrow \textsc{Extend}^b(l_i^b, k)$
22:                if $l_k^b$ is feasible then
23:                    $\Gamma_k^b \leftarrow \mathrm{EFF}(\Gamma_k^b, l_k^b)$
24:                if $\overline{\Gamma}_k^b \neq \emptyset$ then
25:                    $E \leftarrow E \cup \{k\}$
26:        $E \leftarrow E \setminus \{i\}$
27:    until $E = \emptyset$
28:    // Label concatenation //
29:    for all $(i, j) \in A$ do
30:        for all $l_i^{\mathrm{f}} \in \Gamma_i^{\mathrm{f}}$ do
31:            for all $l_j^{\mathrm{b}} \in \Gamma_j^{\mathrm{f}}$ do
32:                if $\textsc{Feasible}(l_i^{\mathrm{f}}, l_j^{\mathrm{b}})$ and $\textsc{Halfway}(l_i^{\mathrm{f}}, l_j^{\mathrm{b}})$ then
33:                    $P = \textsc{Concatenate}(l_i^{\mathrm{f}}, l_j^{\mathrm{b}})$
34:                    $\Pi \leftarrow \Pi \cup \{P\}$
35:    return best path in $\Pi$
```

with negative reduced cost, we identify vertices that are visited multiple times (according to the different criteria described in Section 3.3.1) and insert them in $\Theta$, and we perform another iteration of the labeling algorithm. When $\Theta$ is empty, the algorithm is equivalent to the normal state space relaxation algorithm. When $\Theta = N$, it is equivalent to the conventional bidirectional dynamic programming algorithm.

It is important to observe that the main trade-off emerging from this procedure is the cost of a single iteration versus the number of iterations overall. In fact, when $\Theta$ has few vertices, the cost of an iteration is low, but the algorithm is expected to require a higher number of iterations to obtain an elementary solution. Conversely, when $\Theta$ is large, the cost of an iteration is higher, but it is more likely to find a feasible solution within a lower number of iterations. The initialization and vertex insertion strategies described below are focused on balancing this trade-off.

### 3.3.1 Insertion strategies

At the end of each iteration of DSSR, if the path with the least reduced cost is not feasible with respect to the elementarity constraints we need to mark one or more of the vertices visited multiple times as critical and insert them in $\Theta$. In a study about an exact optimization algorithm for the orienteering problem with time windows, Righini and Salani (2009) compare the performance of various different approaches to perform this step, defined originally by Boland et al. (2006):

- HMO (highest multiplicity on the optimal path): insert one vertex at a time, selecting the vertex that is visited the most in the optimal path. In case of *ex aequo*, choose one vertex at random.
- HMO-All: insert all vertices in the path that are visited the maximum number of times.
- MO-All (multiplicity greater than one on the optimal path): insert all vertices visited more than once in the optimal path. This strategy is equivalent to the one used originally by Righini and Salani (2008).

According to the computational experiments carried out by the authors, the HMO strategy is the most conservative approach, yielding the smallest cardinality of $\Theta$ at the end of the algorithm, and requiring the highest number of iterations. The MO-All strategy behaves in a complementary manner, requiring the least number of iterations but eagerly increasing the size of $\Theta$. While neither of these two strategies dominate one another, they both seem to outperform the HMO-All strategy.

**Algorithm 3.5** Decremental State Space Relaxation

```
1: procedure DSSR
2:     // Initialization //
3:     $\Theta \leftarrow$ InitCriticalSet()
4:     $\Psi \leftarrow \emptyset$
5:     repeat
6:         $\Theta = \Theta \cup \Psi$
7:         $P \leftarrow$ BBDP($\Theta$)
8:         // Search for vertices visited more than once //
9:         $\Psi \leftarrow$ MultipleVisits($P$)
10:    until $\Psi = \emptyset$
11:    return $P$
```

In our own analysis, described in Chapter 5, we attempt to generalize those strategies, even extending the search of vertices visited multiple times outside of the optimal path.

### 3.3.2 Initialization strategies

In addition to the techniques described above, Righini and Salani (2009) explore different strategies concerning the initialization of the set of critical vertices $\Theta$. The intuition behind this is that by inserting vertices that are highly likely to be visited multiple times from the start of the algorithm, it is possible to accelerate the whole procedure. In order to achieve this, the authors define a measure $f_{ij}$ of "cycling attractiveness" of a vertex $i$ with respect to a vertex $j$ as the ratio of the dual price $\eta_i$ over the duration of the 2-cycle $i - j - i$:

$$f_{ij} = \eta_i/(s_i + t_{ij} + s_j + t_{ji}).$$

Then they introduce four possible orderings of the vertices in $N$ according to this measure (all these orderings are non-increasing):

1. Highest cycling attractiveness (HCA): order by $\max_{j \in N \setminus \{i\}} f_{ij}$.
2. Total cycling attractiveness (TCA): order by $\sum_{j \in N \setminus \{i\}} f_{ij}$.
3. Weighted HCA (WHCA): order by $(b_i - a_i) \max_{j \in N \setminus \{i\}} f_{ij}$.
4. Weighted TCA (WTCA): order by $(b_i - a_i) \sum_{j \in N \setminus \{i\}} f_{ij}$.

Additionally, the authors define a *mixed* strategy, which works in the following way: let us define the sets $HCA_m$, $TCA_m$, $WHCA_m$, and $WTCA_m$, $1 \leq m \leq n$, as the sets composed by the first $m$ elements of $N$ sorted according to their respective ordering. Then, the mixed strategy consists in initializing the set of critical vertices with their intersection, i.e., $MIX_m = HCA_m \cap TCA_m \cap WHCA_m \cap WTCA_m$.

According to the results obtained by the authors, while none of the original four strategies dominates another, initialization proves to be advantageous, since in general it seems to reduce the number of iterations and overall computing time. For the MO-All insertion strategy in particular, initializing $\Theta$ seems to mitigate the insertion of vertices that are not necessary to compute an optimal path. The adoption of the mixed strategy seems also to reduce considerably the overall computing time for the HMO and MO-All insertion strategies. The authors perform computational experiments on a mixed data set composed of the Solomon's and Cordeau's instances for the VRPTW, with a cardinality of at least 100 customers. For the parameter $m$, they consider values of $m = 5$, $m = 10$ and $m = 20$, concluding that the best results are obtained when $m$ is about half the expected optimal path length.

Algorithm 3.5 describes DSSR. Here, the procedure InitCriticalSet() uses one of the strategies described above to determine how to initialize the set $\Theta$. Procedure MultipleVisits($P$) extracts from path $P$ the vertices that are to be marked as critical according to the chosen insertion strategy and stores them in the auxiliary set $\Psi$, which is used to update $\Theta$ at the start of each iteration of the dynamic programming procedure.

## 3.4 *ng*-route Relaxation

Another state space relaxation strategy for solving ESPPRC efficiently is *ng-route relaxation* (NGRR), originally proposed by Baldacci et al. (2010, 2011) for their reduced set partitioning method described in Section 2.2.3. This technique partially relaxes the elementarity conditions while loosening the lower bound in order to improve the pricing efficiency. To this end, it first introduces an *ng*-neighborhood $M_i \subset N$ for each vertex $i \in N$ consisting of its nearest vertices and vertex $i$ itself. A path that is feasible with respect to capacity, duration, and time window constraints but contains cycles in the form $(i - \cdots - j - \cdots - i)$ such that $i \notin M_j$ is called an *ng-path.* Since such cycles tend to be expensive, they are therefore unlikely to appear in the best solution found by the algorithm. In this study, we refer to them as cycles that are valid according to the *ng*-neighborhoods, or *valid ng-cycles.*

Each time a label $l_i$ is extended to create a new label $l_j$, the elementarity resource $E_j^k$ is set to zero for $k \notin M_j$. In other words, for each label $l_i$ it is possible to define a set of vertices $R(l_i)$ containing the vertices that cannot be visited from $l_j$, i.e., $R(l_i) \coloneqq \{k \in N | E_i^k = 1\}$. Then, when creating a new label $l_j$, it is possible to determine $R(l_j)$ with

$$R(l_j) = (R(l_i) \cup \{j\} \cup U_j) \cap M_j,$$

where $U_j$ is the set of unreachable vertices for $l_j$ as defined in Section 3.2. This accelerates the label dominance since at most $|M_j|$ elementarity resources of two partial paths arriving at vertex $j$ are compared.

Let us illustrate how NGRR works within the label extension algorithm with the following example, illustrated in Figure 3.4.

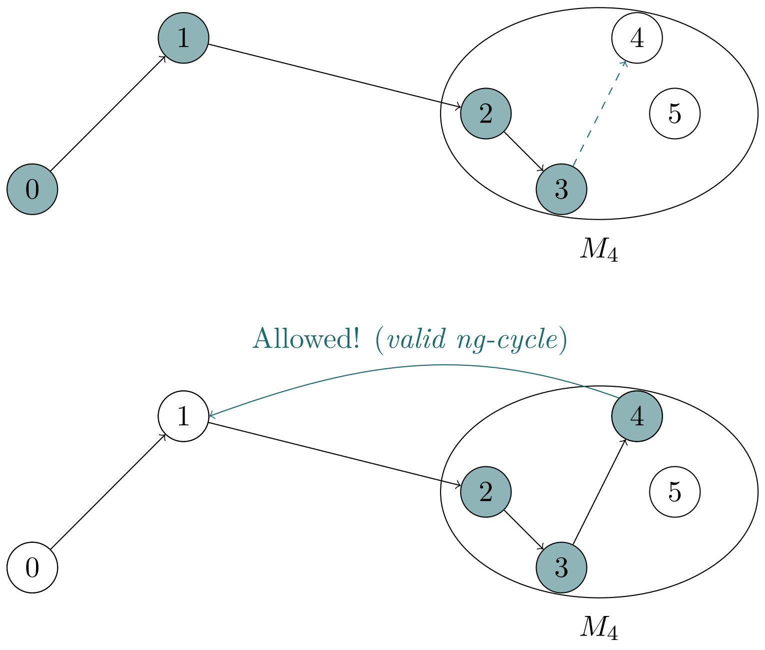
**Example 3.4.1.** We want to extend the forward label $l_3$ representing partial path $P = (0, 1, 2, 3)$ to node 4. Here, the set of prohibited vertices is simply comprised by the ones visited by P, i.e., $R(l_3) = \{0, 1, 2, 3\}$. Thus, the consumption of the elementarity resource is set to one only for the respective customers, i.e., $E_3^j = 1$ for $j = 1, 2, 3$ (recall that there is no need for an elementarity resource for the source 0). Let us define the *ng*-neighborhood $M_4 = \{2, 3, 4, 5\}$. Since 0 and vertex 1 are not in $M_4$, we remove them from the set of prohibited vertices for the new label $l_4$. Therefore, $R(l_4) = \{2, 3, 4\}$ and the elementarity resource consumption $E_4^1$ is set to 0. Note that extending $l_4$ to vertex 1 is now possible, although it is not likely to lead to the optimal solution since $t_{4,1}$ is implied to be large.

## 3.5 Hybrid Algorithms

Both DSSR and NGRR are attempts to treat efficiently the binary resources of a label in order to ensure elementarity either for each optimal path found at the subproblem level, or at least for the paths that are part of the final VRPTW solution returned by the BP algorithm. Thus, it is natural to attempt to combine these two techniques, and indeed a few authors have developed some promising techniques in the recent years.

In particular, Martinelli et al. (2014) and Dayarian et al. (2015b) simultaneously observed that it is possible to augment the *ng*-neighborhoods of NGRR dynamically, in an analogous way to the set of critical vertices $\Theta$ in DSSR. Using the same notation of Dayarian et al. (2015b), we then introduce the *applied neighborhoods* $\widehat{M_i} \subseteq M_i$, for all $i \in N$. These sets are then used throughout the label extension procedure, according to the rules described in the previous section, instead of the original sets $M_i$. The applied neighborhoods are initialized as empty, and updated dynamically according to the following rule.

Figure 3.1: Label extension with *ng*-route relaxation. Prohibited vertices for the partial paths are colored.



Let $C = (i, j_1, \ldots, j_m, i)$ be a cycle that is not valid with respect to the original neighborhoods $M_j$, $j \in N$. Therefore, according to the rules described in Section 3.4, it holds that $i \in M_{j_k}$, for all $k \in \{1, \ldots, m\}$. Indeed, if there were one neighborhood that did not contain $i$, say $M_{j_1}$, then the elementarity resource $E^i_{j_1}$ would be reset to 0 and another extension to $i$ would be allowed. Since this cycle occurred after running the labeling algorithm, it follows that there exists a $k \in \{1, \ldots, m\}$ such that $i \notin \widehat{M}_{j_k}$. Thus, in order to prevent the generation of this cycle in the future, it suffices to insert vertex $i$ in all applied neighborhoods $\widehat{M}_{j_k}$.

In their study, Dayarian et al. (2015b) compare this method with a different hybridization between NGRR and DSSR. Here, at the end of each iteration, vertex $i$ in the invalid cycle $C = (i, j_1, \ldots, j_k, i)$ is marked as critical and added to the applied neighborhoods of all other vertices that consider $i$ as neighbor, i.e., we add $i$ to each $\widehat{M}_j$ such that $i \in M_j$. According to the computational experiments of the authors, this strategy is less efficient when compared to the previous one.

Finally, let us describe two additional variations of the original hybrid algorithm, considered by the same authors mentioned above. Martinelli et al. (2014) also consider a modified version in which the sets $\widehat{M_i}$, while behaving in the same way as explained above, are not necessarily a subset of the original neighborhoods. Here, the update procedure is applied whenever any cycle is detected. Indeed, this version does not use *ng*-routes at all, producing only elementary routes.

In a different study, Dayarian et al. (2015a) prevent obtaining a non-elementary optimal path while using *ng*-routes in a different manner. Here, if the algorithm returns a non-elementary *ng*-route, it is not left to terminate normally, but an additional layer of DSSR is run instead. Thus, the repeating vertex $i$ in any valid *ng*-cycle is then marked as critical and added to all the applied neighborhoods $\widehat{M_j}$, $j \in N$, such that $i \in M_j$.

All the algorithms described in this section are going to be considered in more detail in Chapters 5 and 6.

## 3.6 Branch-and-Bound

As mentioned earlier, in order to obtain a BP procedure, we need to embed CG within BB. BB is a classic methodology to solve combinatorial optimization problems, and in particular integer programs, to optimality. It basically consists in a selective exploration of the solution space that uses available bounds on the optimal value in order to avoid unnecessary exploration of the areas of the solution space in which the optimal solution cannot be found. This exploration is performed within a tree structure, and at each node of the tree we solve a linear relaxation of the original problem. In BP, this relaxation is solved with CG.

We consider here as an illustrative example a generic 0–1 minimization problem:

$$\min\ cx \tag{3.30}$$

$$\text{s.t. } Ax \geq b, \tag{3.31}$$

$$x \in \{0,1\}^n. \tag{3.32}$$

Let $\overline{x}$ be the solution to its linear relaxation. In general, $\overline{x}$ is not an integer solution, so here we can assume it to be fractional (if it was integer, the algorithm would simply stop). Let then $\overline{x_0}$ be a fractional component of $\overline{x}$. We can then perform the *branching* operation by introducing two subproblems, which consist in the same linear relaxation of the LP above, but with one additional constraint each: in the first one, we impose $\overline{x_0} = 0$, while in the second one we impose $\overline{x_0} = 1$. We then create two *child* nodes for the *root* node represented by the original LP, for which the fractional solution $\overline{x}$ is clearly infeasible.

It is important to observe that performing this operation recursively to obtain a complete binary tree corresponds to an exhaustive search of the solution space, which would clearly be too inefficient in general, and especially in our case. This is where *bounding* plays an important role. Assume that we know a valid upper bound $z'$ to the optimal value of the original integer program (for instance, obtained through the use of an heuristic), and let $z^*$ be the optimal solution of the linear relaxation at the current node of the BB tree. It is easy to see that if $z' < z^*$, then the part of the tree that is rooted at the current node cannot lead to the optimal solution of the original LP, since $z^*$ is a lower bound for the optimal integer solution of the current node. Therefore, we can discard (or *prune*) the current node and explore the remaining nodes. Furthermore, we can avoid branching on a node if the associated LP is infeasible or if its solution is integer. In the latter case, if the value of the solution is lower than the best available upper bound, then the solution becomes the new best available one (also known as the *incumbent*). Algorithm 3.6 describes branch-and-bound for a 0–1 minimization problem.

Due to the versatility of BB, there are several other algorithmic choices that one can make when using it within CG for routing problems, such as the strategy used to navigate the tree, or where and how to use certain methods to further tighten the bounds. In Chapter 6, we discuss all the implementation details that are relevant for our study. We conclude this chapter with an observation about how to perform branching when solving routing problems with CG.

### 3.6.1 Branching rules for the VRPTW

While branching on the 0–1 route variables $y_r$ in the RMP (3.1)-(3.3) is certainly possible, it is ill-advised for two reasons. The first one is the difficulty of implementation: while imposing $y_r = 1$ is easy, forbidding a route by imposing $y_r = 0$ would present some complications at the subproblem level (see Feillet, 2010). The second and main reason is that the resulting BB tree would be heavily imbalanced, and therefore inefficient: while imposing a route is a very strong branching rule, doing the opposite is an exceptionally weak one. In fact, removing a single route from the search space would affect the difficulty of the problem at a child node only negligibly, given the very large amount of route variables.

Because of this, alternative branching rules are used in the context of

**Algorithm 3.6** Generic branch-and-bound.

```
1: procedure BB
2:     UB ← +∞
3:     Incumbent ← null
4:     $Q \leftarrow \{\text{Root Node}\}$
5:     while Q not empty do
6:         Take node $q$ from $Q$
7:         Solve LP of $N$
8:         if LP is feasible then
9:             $S \leftarrow$ LP solution
10:            $C \leftarrow$ cost of $S$
11:            if $C < UB$ then
12:                if $S$ is integral then
13:                    Incumbent ← $S$
14:                    UB ← $C$
15:                else
16:                    Find fractional variable $x$ of $S$
17:                    Generate child node $q_0$ where $x \equiv 0$
18:                    Generate child node $q_1$ where $x \equiv 1$
19:                    $Q \leftarrow Q \cup \{q_0, q_1\}$
20:    return Incumbent
```

routing problems. The most popular of such rules (see the survey of Desaulniers et al., 2014), and the one that we adopt in this study, consists in branching on fractional arc flow, i.e., branching on the variables of the compact formulation (2.1)-(2.9). Given a fractional solution to the RMP, let $b_{ijr} = 1$ if route $r$ traverses arc $(i, j) \in A$ and 0 otherwise. We identify an arc $(i, j) \in A$ such that $0 < f_{ij} < 1$, where the arc flow $f_{ij}$ is computed as $f_{ij} = \sum_{r \in \overline{\Omega}} b_{ijr} x_r$. If this is not possible, i.e., we have a solution such that $f_{ij} \in \{0, 1\}$ for every arc $(i, j)$, then one can prove that such a solution represents a feasible set of routes (Feillet, 2010). While any fractional arc can be used, in our implementation we choose an arc whose flow is closest to $\frac{1}{2}$, in other words we choose the most fractional arc. We then create two branches: one in which the arc is forbidden ($x_{ij} = 0$) and one in which it is enforced ($x_{ij} = 1$). Implementation-wise, in order to forbid an arc in the CG scheme at a child node, one simply has to first remove any route $r$ such that $b_{ijr} = 1$, and then remove arc $(i, j)$ from the graph. Analogously, in order to enforce an arc $(i, j)$, one simply has to forbid any arc of the form $(k, j)$ with $k \neq i$ or $(i, h)$ with $h \neq j$.

## 3.7 Conclusions

In this chapter, we have presented the general structure of BP for the VRPTW, discussing in detail the labeling algorithms that are used to solve the associated pricing problem, the ESPPRC. Since solving the ESPPRC efficiently is crucial for the effectiveness of the whole BP procedure, we have presented some of the most well-known techniques that have been developed to improve these labeling algorithms.

However, due to previously mentioned considerations related to the scope of this study, we have omitted certain advanced topics. In particular, we have not considered dual variable stabilization, which is an important set of techniques for the effectiveness of CG. In cases in which the RMP is degenerate, the marginal cost of customers can be badly estimated and dual variables can oscillate between very different values, causing the insertion of undesirable columns in the RMP and the increase of the number of necessary CG iterations. To address this, several techniques have been proposed to counter dual variable oscillation by limiting the distance traveled in the dual space from one iteration to the next. These methods include for instance the definition of boxes around the current dual values and preventing the selection of values outside them (Marsten et al., 1975), or the linear penalization of the distance traveled in the dual space (Kim et al., 1995). For a more detailed review on stabilization techniques, see for instance the surveys of Lübbecke and Desrosiers (2005) or Briant et al. (2008).

In the next chapter, we are going to discuss how to adapt the labeling procedures discussed so far to a variant of the VRPTW, before presenting our own analysis of these techniques later in this study.

# Chapter 4

# The Vehicle Routing Problem with Time Windows and Waiting Time Costs

In most of the existing studies for the VRPTW, the objective of the problem is to minimize the total distance traveled, which is usually supposed to be equivalent to the total traveling time. In this chapter, we consider instead a variant of the VRPTW, whose aim is to minimize the total route duration, i.e., the sum of the traveling, service, and waiting times. Thus, the vehicle departure times from the depot directly influence the objective value. We refer to this problem as the vehicle routing problem with time windows and waiting time costs (VRPTWWTC).

Route duration is defined as the length of the interval between the departure and arrival times of the vehicle at the depot. As such, it is supposed that operational costs are not incurred before the departure and after the arrival at the depot while waiting after the departure is as costly as traveling. This objective function can be suitable for modeling situations where waiting times bear a certain operational cost, such as fuel consumption during vehicle idle time (Suzuki, 2011). Another example is the delivery of perishable goods, where the need for on-board refrigeration entails a certain expenditure of energy per time unit (Hsu et al., 2007).

Route duration minimization is also relevant when the vehicles are not owned but hired by the distribution company to be used for servicing customers. Each vehicle is requested to be at the depot at the departure time determined by the distribution company and the rental fee is charged per time unit of vehicle usage, i.e., the rental fee is a function of the total route duration. In some situations, the on-board personnel is qualified for servicing operations and, additionally, they also execute on-ground operations during a working day. The general aim is then to maximize the utilization rate of the on-board personnel for the servicing operations, which can be defined as the percentage of the total working time where personnel is servicing customers. Such an objective can be faced for example when the on-board personnel comprises nurses that carry out home healthcare operations or qualified technicians executing material installation or repair operations.

Working time minimization can also be applicable in dial-a-ride problems where drivers are paid per working time unit. In dial-a-ride problems (Cordeau and Laporte, 2007), passengers request rides where each ride is specified for a pick-up and a drop-off location together with a pick-up time and/or a drop-off time. While dial-a-ride problems are commonly faced by non-profit organizations providing transportation services for elderly or disabled people, they also are increasingly attractive for ride-sharing businesses offering on-demand door-to door transportation services in urban areas. Platform-based ride-sharing companies like Uber and Lyft connect drivers with riders through a digital platform. The studied objective function can be relevant in similar dial-a-ride systems since waiting for riders reduces the willingness of the drivers to accept requests, and since drivers are usually compensated for the time spent waiting at a pick-up location besides the travel related expenses.

Total route duration has been considered in several studies. Savelsbergh (1992) investigates edge-exchange methods for the VRPTW with the aim of minimizing the total route duration. Dabia et al. (2013) adopt this objective function for a VRPTW in which arc travel times are time-dependent. They develop a BP algorithm treating the vehicle departure times from the depot as decision variables. Bettinelli et al. (2011) develop a branch-and-cut-and-price algorithm for a VRPTW variant that generates routes of minimal duration, while ultimately aiming to minimize the total traveling distance and the vehicle fixed costs. François et al. (2017) show that, in the context of a multi-trip VRPTW, adopting the total duration as the objective function, instead of total travel time, results in huge savings in terms of waiting times with a relatively small travel time increase.

Various studies consider certain generalizations of this attribute. For instance, Ioachim et al. (1998) develop a dynamic programming algorithm for the shortest path problem with time windows and linear node costs, which includes as a special case the shortest path problem with linear waiting costs studied by Desaulniers and Villeneuve (2000). Additionally, a related class of problems employ soft time windows, in which a linear penalty is paid when a time window violation occurs (see for instance Liberatore et al., 2011).

This chapter is structured as follows. Section 4.1 formally describes the problem, while the following sections present the main theoretical results of the work by Küçükaydın et al. (2014). Section 4.2 discusses how some the aspects of the studied problem are described as functions of the route start time. Then, we introduce the main properties that allow obtaining a new label structure, as well as the associated resource extension functions (REFs) and dominance rules for a monodirectional algorithm (Section 4.3). Finally, we discuss the remaining results that are necessary to transform the monodirectional algorithm into a bounded bidirectional one (Section 4.4).

## 4.1 Problem Definition

Consider a directed graph $G = (V, A)$ with vertex set $V$ and arc set $A$. The vertex set is defined as $V = N \cup \{0, n + 1\}$, where $N$ is the set of $n$ customers, and vertex 0 and vertex $n + 1$ represent two copies of the depot known as the source and the sink, respectively. Each customer $i \in N$ is associated with a delivery demand $d_i \geq 0$, a service time $s_i \geq 0$, and a time window $[a_i, b_i]$, where $a_i$ is the earliest service start time and $b_i$ the latest. Every customer $i \in N$ has to receive its demand $d_i$ within its associated time window $[a_i, b_i]$ but can be visited at most once. Arriving at a customer $i$ before the start $a_i$ of its time window is allowed, but the vehicle must then wait until $a_i$ to start serving the customer. In addition, it takes $s_0$ units of time to load a vehicle at the depot. Without loss of generality, we assume that $d_0 = d_{n+1} = s_{n+1} = 0$.

The arc set is defined as $A = \{(i, j) \in V \times V : i \neq j, a_i + s_i + t_{ij} \leq b_j\}$, where $t_{ij}$ denotes the non-negative travel time from vertex $i$ to vertex $j$. There is an infinite fleet of homogeneous vehicles, each with capacity $Q$, available at the depot. A vehicle can serve a subset of customers by traveling along a route starting at the source and ending at the sink. In order to represent the real-life constraint related to driver shifts, we impose a maximum duration of $S$ time units for each route. The transportation cost of a route depends on the total amount of time that the vehicle spends to serve its assigned customers, including traveling, waiting, and service times. The departure time from the depot is a decision variable for each vehicle and thus we do not impose explicit bounds on the time windows of the source and the sink, i.e., $[a_0, b_0] = [a_{n+1}, b_{n+1}] = ]-\infty, \infty[$.

## 4.2 Service Start Time and Path Duration Functions

In addition to the sequence of customers to be visited, the start time $T_0$ of a vehicle has also to be decided with the aim of minimizing the total duration of the path followed by the vehicle. As a consequence, the service start time at any customer and the resulting transportation cost, which is simply the path duration, become functions of $T_0$, where $T_0 \in \,]-\infty, \infty[$. Let us consider a feasible partial path $P = (0 = v_0, v_1, \ldots, v_i)$. Since such a path either arrives at vertex $v_i$ before $a_{v_i}$ or within $[a_{v_i}, b_{v_i}]$, it is clear that the service start time at $v_i$, denoted by $T_{v_i}(T_0)$, is represented by the recursive function

$$T_{v_i}(T_0) = \max\left\{a_{v_i}, T_{v_{i-1}}(T_0) + \bar{t}_{v_{i-1},v_i}\right\}, \tag{4.1}$$

where $\bar{t}_{v_{i-1},v_i} \coloneqq s_{v_{i-1}} + t_{v_{i-1},v_i}$. For simplicity, we denote the index $v_i$ as $i$ in the following, e.g., we denote the service start time $T_{v_i}(T_0)$ at $v_i$ as $T_i(T_0)$.

The service start time function $T_i(T_0)$ can be explicitly written by means of the *latest feasible start time* $\nu_i$ from the source and the *earliest feasible service start time* $\mu_i$ at $v_i$. The latest feasible start time $\nu_i$ can be expressed as

$$\nu_i = \min_{1 \le k \le i} \left\{b_k - \theta_k\right\}, \tag{4.2}$$

where $\theta_k$ is the cumulative travel time and updated as $\theta_k = \theta_{k-1} + \bar{t}_{k-1,k}$. Thus, $T_0$ is restricted to lie in the range $]-\infty, \nu_i]$.

Similarly, the earliest feasible service start time $\mu_i$ is determined recursively by

$$\mu_i = \max\left\{a_i, \mu_{i-1} + \bar{t}_{i-1,i}\right\}, \tag{4.3}$$

where $\mu_0 \coloneqq -\infty$. Hence, when $\mu_i < \nu_i + \theta_i$ (which we refer to as Condition I), we can compute the service start time $T_i(T_0)$ at vertex $v_i$ visited by path $P$ as follows:

$$T_i(T_0) = \begin{cases} \mu_i & \text{if } T_0 \le \mu_i - \theta_i, \\ T_0 + \theta_i & \text{if } \mu_i - \theta_i \le T_0 \le \nu_i. \end{cases} \tag{4.4}$$

In this case, the service start time is a piecewise linear function composed of two pieces, one constant and one linear. Whenever $\mu_i$ becomes greater than or equal to $\nu_i + \theta_i$ (which we refer to as Condition II), $T_i(T_0)$ becomes a constant function and is equal to $\mu_i$ for $T_0 \le \nu_i$. This case can only arise when waiting at a vertex $k$, $1 \le k \le i$, cannot be avoided.

The time spent by a vehicle following path $P$, in other words the duration of path $P$, is calculated as $D_i(T_0) := T_i(T_0) - T_0$. Evidently, it is also a function of $T_0$, and when Condition I holds, it can be rewritten as follows:

$$D_i(T_0) = \begin{cases} \mu_i - T_0 & \text{if } T_0 \leq \mu_i - \theta_i, \\ \theta_i & \text{if } \mu_i - \theta_i \leq T_0 \leq \nu_i. \end{cases} \tag{4.5}$$

Analogously to the service start time function $T_i(T_0)$, the path duration function $D_i(T_0)$ is a piecewise linear function consisting of one linear and one constant piece when $\mu_i < \nu_i + \theta_i$. In the case where waiting becomes inevitable for path $P$, $D_i(T_0)$ becomes a linear function expressed by $\mu_i - T_0$ with $T_0 \leq \nu_i$, as opposed to $T_i(T_0)$.

If functions $T_i(T_0)$ and $D_i(T_0)$ are composed of two pieces, the minimum duration of path $P$ is equal to $\theta_i$, which is the cumulative travel time, obtained for any value of the start time $T_0$ within the range $[\mu_i - \theta_i, \nu_i]$. On the other hand, if waiting becomes unavoidable, both functions are minimized for $T_0 = \nu_i$ and the minimum duration of $P$ is calculated as $\mu_i - \nu_i$. We can easily conclude that for any path $P$, the optimal value for $T_0$ is provided by the latest feasible start time $\nu_i$ from the source regardless of whether or not waiting takes place.

Although we are able to find the optimal value of $T_0$ for any predefined path, establishing paths taking into account their feasible start times from the source is a challenging task within a BP framework. Compared with the traditional VRPTW minimizing the total distance traveled, it calls for employing extra resources with non-decreasing resource extension functions (see Irnich, 2008). The increased number of resources naturally increases the number of dominance criteria used to eliminate the paths that are not leading to optimality and makes the pricing problem more time consuming.

Let us illustrate the behavior of these functions with an example.

**Example 4.2.1.** Let us consider a partial path $P = (0 = v_0, v_1, v_2, v_3)$. The time windows of $v_1$, $v_2$, $v_3$ are given by $[8, 15]$, $[10, 16]$, $[20, 23]$, respectively, and furthermore we establish that $\bar{t}_{01} = 1$, $\bar{t}_{12} = 3$, and $\bar{t}_{23} = 2$. We start by computing the earliest service start time $\mu_1$ and the latest feasible start time $\nu_1$ from the depot for $v_1$, and obtain $\mu_1 = \max\{8, -\infty\} = 8$ and $\nu_1 = 15 - 1 = 14$. Condition I is satisfied, since $\mu_1 < \nu_1 + \theta_1$ where $\theta_1 = 1$. Thus, the service start time function $T_1(T_0)$ is expressed by the following:

$$T_1(T_0) = \begin{cases} 8, & \text{if } T_0 \leq 7 \\ T_0 + 1, & \text{if } 7 \leq T_0 \leq 14. \end{cases}$$

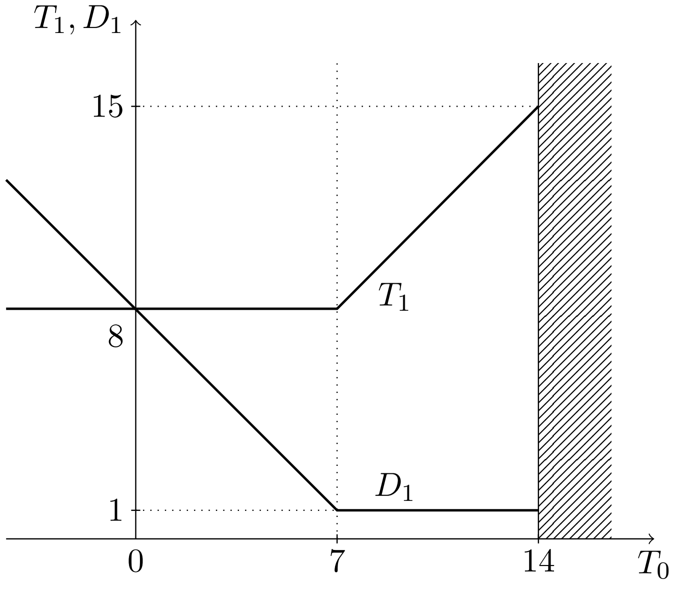


Figure 4.1: Service start time function $T_1$ and duration function $D_1$.

Similarly, the total duration function is given by

$$D_1(T_0) = \begin{cases} 8 - T_0, & \text{if } T_0 \leq 7 \\ 1, & \text{if } 7 \leq T_0 \leq 14. \end{cases}$$

It is easy to see that the behavior of these two functions is caused by the fact that there is a positive waiting time at $v_1$ if $T_0 < 7$. Both functions are depicted in Figure 4.1.

Before extending the partial path to $v_2$, we need to update $\mu_2$ and $\nu_2$ using equations (4.3) and (4.2), respectively: $\mu_2 = \max\{10, 8+3\} = 11$ and $\nu_2 = \min\{14, 16-4\} = 12$, where $\theta_2 = 4$. Condition I is satisfied again, since $\mu_2 < \nu_2 + \theta_2$. Therefore $T_2(T_0)$ is written as the following:

$$T_2(T_0) = \begin{cases} 11, & \text{if } T_0 \leq 7 \\ T_0 + 4, & \text{if } 7 \leq T_0 \leq 12, \end{cases}$$

while $D_2(T_0)$ is written as:

$$D_2(T_0) = \begin{cases} 11 - T_0, & \text{if } T_0 \leq 7 \\ 4, & \text{if } 7 \leq T_0 \leq 12. \end{cases}$$

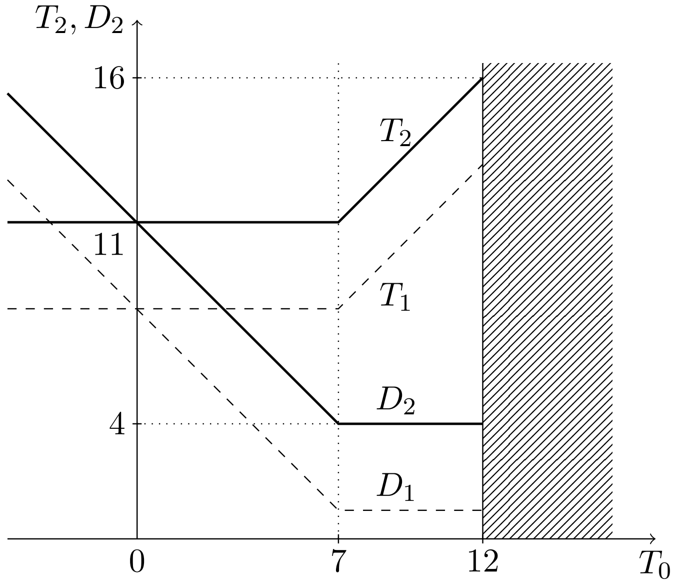


Figure 4.2: Service start time function $T_2$ and duration function $D_2$.

These functions are depicted in Figure 4.2.

Extending to vertex $v_3$ yields $\mu_3 = \max\{20, 11+2\} = 20$ and $\nu_3 = \min\{12, 23-6\} = 12$. Since $\theta_3 = 6$ and $\mu_3 > \nu_3 + \theta_3$, Condition II holds. Thus, $T_3(T_0) \equiv 20$ and $D_3(T_0) = 20 - T_0$ for $T_0 \leq 12$, as shown in Figure 4.3.

We can observe that, given the arrangement of the time windows, it is necessary for the vehicle to wait until $\mu_3$ before serving customer $v_3$, differently from the previous cases. Furthermore, departing any earlier of the latest feasible time $\nu_3$ is going to cause additional waiting time. $\square$

To conclude, we define the total cost function $C_i(T_0)$ for vertex $v_i$ in a partial path $P$ as the difference between the total duration and the cumulated sum of the dual prices, i.e., $C_i(T_0) \coloneqq D_i(T_0) - \sum_{k=0}^{i-1} \eta_k$.

We can now discuss how to use these functions of $T_0$ in a label structure that allows to fathom partial paths that cannot lead to the optimal solution.

## 4.3 A New Label Structure

Let us consider two different partial paths $P_1 = (0 = v_0^1, v_1^1, \ldots, v_{i-1}^1, v_i)$ and $P_2 = (0 = v_0^2, v_1^2, \ldots, v_{i-1}^2, v_i)$ starting from source 0 and ending at the

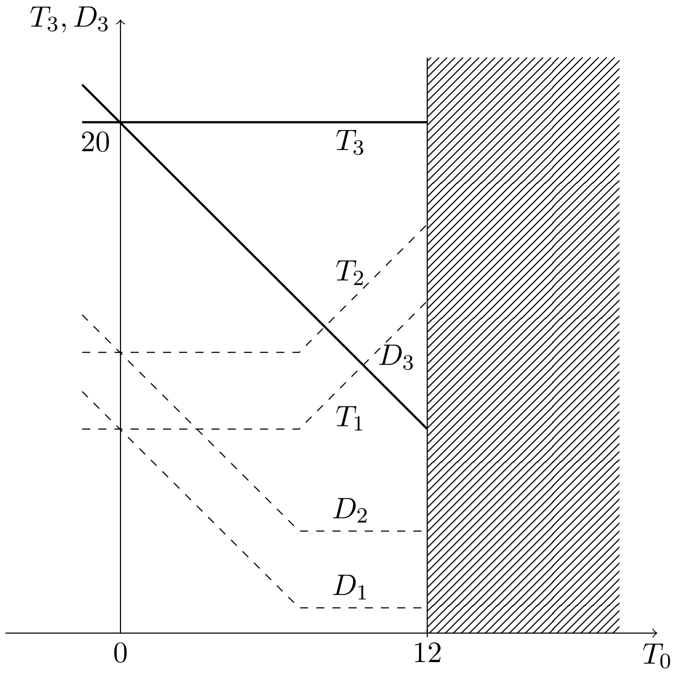


Figure 4.3: Service start time function $T_3$ and duration function $D_3$.

same vertex $v_i$. If one were to use a label structure akin to the one we have defined in Chapter 3 for the VRPTW, these paths would be represented with labels $l_1 = (C_i^1, D_i^1, T_i^1, q_i^1, (E_i^k)_k^1)$ and $l_2 = (C_i^2, D_i^2, T_i^2, q_i^2, (E_i^k)_k^2)$, respectively. In order for $P_1$ to dominate $P_2$, it would then suffice to check whether $C_i^1 \leq C_i^2$, $D_i^1 \leq D_i^2$, $T_i^1 \leq T_i^2$, $q_i^1 \leq q_i^2$, and $E_i^{k,1} \leq E_i^{k,2}$ for each $k \in N$.

While the comparisons of the vehicle load and elementarity resources pose no issue, since these resources take numerical values, the other ones are not straightforward since they compare functions of $T_0$. Furthermore, even if $C_i^1(T_0) \leq C_i^2(T_0)$, $D_i^1(T_0) \leq D_i^2(T_0)$, and $T_i^1(T_0) \leq T_i^2(T_0)$ for all $T_0$, one could not immediately guarantee that the same would hold for index $j$ when extending both partial paths to the same vertex $v_j$. In particular, in order to discard path $P_2$, it has to hold that $\nu_i^2 \leq \nu_i^1$ and $\nu_j^2 \leq \nu_j^1$, otherwise functions $C_j^2(T_0)$, $D_j^2(T_0)$, and $T_j^2(T_0)$ would be defined for values $T_0 \in ]\nu_j^1, \nu_j^2]$, which would prevent the elimination of the second label.

Küçükaydın et al. (2014) are able to determine sufficient conditions for dominance, which we condense in the following propositions.

**Proposition 4.3.1.** Let $\nu_i^2 \geq \nu_i^1$ and $D_i^1(T_0) \leq D_i^2(T_0)$, for $T_0 \leq \nu_i^2$. Then, when extending to the same vertex $v_j$, the following holds:

$$\begin{aligned}
&\nu_j^1 \geq \nu_j^2, && \\
&D_j^1(T_0) \leq D_j^2(T_0) && \forall\, T_0 \leq \nu_j^2, \\
&T_i^1(T_0) \leq T_i^2(T_0) && \forall\, T_0 \leq \nu_i^2.
\end{aligned}$$

Furthermore, if $\sum\limits_{k=0}^{i-1} \eta_k^1 \geq \sum\limits_{k=0}^{i-1} \eta_k^2$, then

$$C_j^1(T_0) \leq C_j^2(T_0) \qquad \forall\, T_0 \leq \nu_j^2.$$

□

Thus, in order to define label dominance rules it only remains to establish the conditions under which $D_i^1(T_0) \leq D_i^2(T_0)$, for $T_0 \leq \nu_i^2$. Given the previously discussed structure of the duration function, it suffices to determine whether $\mu_i^1 = D_i^1(0) \leq D_i^2(0) = \mu_i^2$ and $\zeta_i^1 \leq \zeta_i^2$, where $\zeta_i$ is defined as the minimum value of $D_i(T_0)$. The authors obtain the following result regarding $\zeta_i$ when extending a label to a new vertex.

**Proposition 4.3.2.** Let $P$ be a partial path ending at vertex $v_i$ with duration function $D_i(T_0)$ and $\zeta_i$ its minimum value. Then the following holds when extending $P$ to vertex $v_j$:

$$\zeta_j = \max\left\{\zeta_i + \bar{t}_{i,j}, \mu_j - \nu_j\right\}.$$

□

We can now establish the following label structure $\lambda_i$ for a partial path $P = (0 = v_0, v_1, \ldots, v_i)$.

$$\lambda_i \coloneqq \left(\delta_i, \nu_i, \mu_i, \zeta_i, q_i, (E_i^k)_k\right), \tag{4.6}$$

where $\delta_i \coloneqq \sum_{k=0}^{i-1} \eta_k$. The associated REFs are then expressed as follows:

$$\delta_j = \delta_i + \eta_j, \tag{4.7}$$

$$\nu_j = \min\left\{\nu_i, b_j - \theta_j\right\}, \tag{4.8}$$

$$\mu_j = \max\left\{a_j, \mu_i + \bar{t}_{ij}\right\}, \tag{4.9}$$

$$\zeta_j = \max\left\{\zeta_i + \bar{t}_{ij}, \mu_j - \nu_j\right\}, \tag{4.10}$$

$$q_j = q_i + d_j, \tag{4.11}$$

$$E_j^k = \begin{cases} E_i^k + 1, & \text{if } k = j, \\ E_i^k, & \text{if } k \neq j, \end{cases} \qquad \forall k \in N, \tag{4.12}$$

where $\theta_j$ is updated as $\theta_j = \theta_i + \bar{t}_{ij}$. The new label $\lambda_j$ is considered feasible if and only if $\mu_j \leq b_j$, $\zeta_j \leq S$, $q_j \leq Q$ and $E_j^k \leq 1$, for all $k \in N$. Finally, label $\lambda_i^1$ dominates $\lambda_i^2$ if $\delta_i^1 \geq \delta_i^2$, $\nu_i^1 \geq \nu_i^2$, $\mu_i^1 \leq \mu_i^2$, $\zeta_i^1 \leq \zeta_i^2$, $q_i^1 \leq q_i^2$ and $E_i^{k,1} \leq E_i^{k,2}$ for all $k \in N$. However, it is possible to reduce the number of comparisons, and thus to accelerate the labeling algorithm, without losing guarantee of optimality by applying stronger domination rules.

**Proposition 4.3.3.** Let $\lambda_i^1$ and $\lambda_i^2$ be two feasible labels. Then, $\lambda_i^1$ dominates $\lambda_i^2$ if $\delta_i^1 + \min\{\nu_i^1 - \nu_i^2, 0\} \geq \delta_i^2$, $\mu_i^1 \leq \mu_i^2$, $\zeta_i^1 \leq \zeta_i^2$, $q_i^1 \leq q_i^2$ and $E_i^{k,1} \leq E_i^{k,2}$ for all $k \in N$. $\square$

With Proposition 4.3.3, the number of comparisons that are necessary in order to verify label dominance decreases by one, i.e., we now need to perform $4 + n$ comparisons.

## 4.4 Bounded Bidirectional Algorithm

As we have seen in the previous chapter, the bounded bidirectional procedure by Righini and Salani (2006) is more efficient than the monodirectional one, given the reduced number of labels that are generated. Thus, we now describe how to perform backward label extensions for our problem, as well as the details concerning label concatenation.

### 4.4.1 Backward Label Extensions

Clearly, the same issues discussed previously arise in this case as well. In particular, the duration, total cost, and service start time resources are functions of a decision variable, which in this case is the time of arrival at the depot $T_{n+1}$.

Thus, given a backward partial path $P^b = (n+1 = v_0^b, \ldots, v_i^b)$ starting at the sink and ending at a vertex $v_i^b$, the service start times as functions of $T_{n+1}$ can be expressed as

$$T_i^b(T_{n+1}) = \min\left\{b_i, T_{i-1}^b(T_0) + \bar{t}_{i,i-1}\right\}, \tag{4.13}$$

analogously to equation (4.1). Let us now define the *earliest feasible arrival time* at the depot, which for path $P^b$ is defined as $\sigma_i := \max_{1 \leq k \leq i}\{a_k + \theta_k^b\}$, where $\theta_i^b := \sum_{k=1}^{i} \bar{t}_{k,k-1}$, and the *latest feasible service start time* at customer $v_i$, $\rho_i := \min\{b_i, \rho_{i-1} - \bar{t}_{i,i-1}\}$, where $\rho_0 = +\infty$. Then, the service start time at vertex $v_i$ is defined as follows:

- Condition I: If $\sigma_i - \theta_i^b < \rho_i$

$$T_i^b(T_{n+1}) = \begin{cases} T_{n+1} - \theta_i^b, & \text{if } \sigma_i \leq T_{n+1} \leq \rho_i + \theta_i^b \\ \rho_i, & \text{if } T_{n+1} \geq \rho_i + \theta_i^b \end{cases}$$

- Condition II: Else if $\sigma_i - \theta_i^b \geq \rho_i$

$$T_i^b(T_{n+1}) = \rho_i \text{ for } T_{n+1} \geq \sigma_i.$$

Let us now introduce a large enough positive constant $M$ and apply the following inversions. We can rewrite equation (4.13) thusly:

$$M - T_i^b(T_{n+1}) = \max\left\{M - b_i, (M - T_i^b(T_{n+1})) + \bar{t}_{i,i-1}\right\} \tag{4.14}$$

Additionally, we can introduce the backward time windows for each vertex as $[a_i^b, b_i^b] := [M - b_i, M - a_i]$. Finally, let us define $T_0^b := M - T_{n+1}$ and $T_i^b(T_0^b) := M - T_i^b(T_{n+1})$. It is then easy to see that it is possible to use the same label structure and associated rules from Section 4.3 and obtain feasible label extensions. Indeed, since it holds that $T_i^b(T_0^b) \leq M - a_i$ for each $v_i$, it follows that $T_i^b(T_{n+1}) \geq a_i$. Furthermore, the earliest feasible arrival time at the depot $\sigma_i$ can be derived as $M - \nu_i^b$, where $\nu_i^b := \min_{1 \leq k \leq i}\{M - a_k - \theta_k^b\}$ is the backward equivalent to the latest feasible vehicle departure time (4.2), and the latest feasible service start time $\rho_i$ is $M - \mu_i^b$, where $\mu_i^b := \max\{M - b_i, \mu_{i-1}^b + \bar{t}_{i,i-1}\}$ corresponds to (4.3). The other elements of the label structure can be easily derived applying the same reasoning of Section 4.3.

To summarize, after having computed all the backward time windows $[a_i^b, b_i^b]$, we perform label extensions using the structure

$$\lambda_i^b := \left(\delta_i^b, \nu_i^b, \mu_i^b, \zeta_i^b, q_i^b, (E_i^{k,b})_k\right), \tag{4.15}$$

using the following REFs when extending to vertex $v_j$:

$$\delta_j^b = \delta_i^b + \eta_j, \tag{4.16}$$

$$\nu_j^b = \min\left\{\nu_i^b, b_j^b - \theta_j^b\right\}, \tag{4.17}$$

$$\mu_j^b = \max\left\{a_j^b, \mu_i^b + \bar{t}_{ji}\right\}, \tag{4.18}$$

$$\zeta_j^b = \max\left\{\zeta_i^b + \bar{t}_{ji}, \mu_j^b - \nu_j^b\right\}, \tag{4.19}$$

$$q_j^b = q_i^b + d_j, \tag{4.20}$$

$$E_j^{k,b} = \begin{cases} E_i^{k,b} + 1, & \text{if } k = j, \\ E_i^{k,b}, & \text{if } k \neq j, \end{cases} \qquad \forall k \in N, \tag{4.21}$$

where $\theta_j^b$ is updated as $\theta_j^b = \theta_i^b + \bar{t}_{ji}$. The feasibility and dominance rules are the same ones described in Section 4.3 and Proposition 4.3.3.

### 4.4.2 Bounding and LabelConcatenation

As we have mentioned earlier, when performing bidirectional label extensions it is necessary to choose a criterion to stop extending labels such that the optimal path is obtained without generating an excessive number of labels. In our case, we use the same criterion that is used for the classic VRPTW, i.e., we use total duration as the critical resource (see Section 3.2). In particular, Küçükaydın et al. (2014) propose to extend a forward label $\lambda_i^f$ only if $\zeta_i \leq S/2$, analogously to the approach of Righini and Salani (2006), and apply the same for backward labels.

In order to obtain a complete feasible path, the authors choose furthermore to concatenate a forward label $\lambda_i^f$ and a backward label $\lambda_i^b$. Clearly, to satisfy the capacity and elementarity constraints, it suffices to impose $q_i^f + q_i^b - d_i \leq Q$ and $E_i^{k,f} + E_i^{k,b} \leq 1$ for each $k \in N \setminus \{v_i\}$, respectively. They also derive the following result to ensure feasibility with respect to the service start times.

**Proposition 4.4.1.** The concatenation of $\lambda_i^f$ and $\lambda_i^b$ is feasible with respect to the time constraints if and only if the earliest service start time at $v_i$ is not greater than the latest feasible service start time, i.e., $\mu_i \leq \rho_i$. □

### 4.4.3 Computing the Total Waiting Time

Before we can conclude, we have to observe that even after identifying a feasible concatenation, computing the overall path duration is not straightforward, since the total waiting time $w$ of the resulting path is not necessarily the sum of the waiting times $w^f$ and $w^b$ of the forward partial path and the backwards path, respectively. However, it is possible to use the concatenation theorem of Savelsbergh (1992) in order to compute $w$.

Here, Savelsbergh (1992) introduces the *forward time slack* $F$ for a feasible path, which indicates how much the service start time of a visited vertex can be shifted forward without making the path infeasible with regards to the time resource. Let $P_1^f = (0 = v_0, \ldots, v_i)$ and $P_2^b = (v_j, \ldots, v_m = n+1)$ be a feasible forward partial path and a feasible backward partial path with forward time slacks $F_1$ and $F_2$, respectively. Then, the vehicle departure time $T_0$ of the concatenated path $P$ can be shifted forward by

$$F = \min\left\{F_1, F_2 + w_1 + T_j^b - (T_i + \bar{t}_{ij})\right\}. \tag{4.22}$$

Table 4.1: Total waiting time $w$ on a concatenated path.

| | $\xi_j \geq 0$ | $\xi_j < 0$ |
|---|---|---|
| $w_2 = 0$ | $w_1$ | $w_1 + \max\{0, -\xi_j - B_2\}$ |
| $w_2 > 0$ | $w_1 + \max\{0, w_2 - \xi_j\}$ | $w_1 + w_2 - \xi_j$ |

Additionally, we need to define $\xi_j := T_i + \bar{t}_{ij} - T_j^b$, which is the change of the service start time at $v_j$, and the *backward time slack* $B_2$ of path $P_2^b$, which indicates how much the service start time of vertex $v_j$ can be shifted backward without creating additional waiting time on $P_2^b$. Then, there are four possible ways to determine $w$, shown in Table 4.1, depending on the sign of $\xi_j$ and whether $w_2$ is positive or zero.

In order to obtain $F$, $w_1$, $w_2$, $\xi_j$ and $B_2$, we assume that $T_i = \mu_i$ and $T_j^b = \rho_j$, and therefore that $T_0 = \mu_i - \theta_i$ if Condition I holds, or that $T_0 = \nu_i$ otherwise. Since we set $T_j^b$ equal to the latest feasible service start time $\rho_j$, it follows that $F_2 = 0$. We can also observe that $\xi_j$ can be at most 0, since $\mu_i \leq \rho_i \leq \rho_j - \bar{t}_{ij}$. Indeed,

$$\xi_j = T_i + \bar{t}_{ij} - T_j^b = \mu_i + \bar{t}_{ij} - \rho_j \leq 0.$$

We can compute the other quantities as follows:

$$\begin{aligned} F_1 &= \max\{0, \theta_i - (\mu_i - \nu_i)\}, \\ w_1 &= \max\{0, (\mu_i - \nu_i) - \theta_i\}, \\ B_2 &= \max\{0, \theta_j^b - (\mu_j^b - \nu_j^b)\}, \\ w_2 &= \max\{0, (\mu_j^b - \nu_j^b) - \theta_j^b\}. \end{aligned}$$

Indeed, if the difference between the sum of the travel and service times $\theta_i$ is greater or equal than the difference between the earliest service time $\mu_i$ and the latest departure time $\nu_i$, then the forward time slack of the forward partial path is equal to $\theta_i - (\mu_i - \nu_i)$. Otherwise, if $(\mu_i - \nu_i) > \theta_i$, then the waiting time of the forward partial path is equal to $(\mu_i - \nu_i) - \theta_i$. We can compute analogously the quantities associated with the backward partial path. Then, we use (4.22) to obtain $F$. Finally, if $F \geq w$, we can set the final vehicle departure time at $T_0 + F$, obtaining a total waiting time of 0. Otherwise, the final waiting time is equal to $w - F$. Additionally, the optimal start time from the depot can actually be chosen from the interval $[T_0 + \min\{F, w\}, T_0 + F]$. Once the total waiting time is determined, it is trivial to determine the minimum total duration of the concatenated path, and thus its total cost.

## 4.5 Conclusions

In this chapter, we have discussed the theory regarding the VRPTWWTC and have presented the results that are necessary to implement an exact bounded bidirectional labeling algorithm. Although these results are not the focus of the next chapters, which concentrate more on the empiric analysis of BP algorithms that mainly manipulate the elementarity resources of a label, they are an important component of our study since we carry out computational experiments on this problem variant as well as the classic VRPTW.

# Chapter 5

# Parametrization of Branch-and-Price Algorithms

As we have seen in Chapter 3, the introduction of NGRR inspired several recent articles that propose various hybrid algorithms that merge this technique with DSSR, such as the work of Martinelli et al. (2014) and Dayarian et al. (2015b). This line of research that consists in efficiently manipulating the elementarity constraints, is quite promising, given the clear impact of the associated resources on label domination. However, to the best of our knowledge, there has been no attempt to compare the efficiency of these labeling algorithms in a comprehensive manner.

Furthermore, it is important to observe that these algorithms, including DSSR and NGRR, rely on many components and parameters, whose values have been manually chosen by the authors on the basis of preliminary computational experiments. Recently, several tools for automatic parameter tuning have been developed with the intention of obtaining the most effective parameter configurations through a systematic approach. In particular, López-Ibáñez et al. (2016) develop an iterated racing procedure, called `irace`, in order to automatically configure optimization algorithms given a set of tuning instances and parameter value ranges.

Thus, the aim of the following two chapters is to rigorously compare the efficiency of seven pricing algorithms proposed in the literature within a BP framework. Our approach is composed of the following phases:

1. In the first phase, we identify several algorithmic strategies and numerical attributes, some of which are naturally shared by multiple algorithms, and we represent them in terms of parameters;
2. In the second phase, we let the seven algorithms undergo a tuning

process with the help of `irace` in order to obtain the best possible parameter configuration for each one separately;

3. In the last and third phase, the tuned algorithms are run on benchmark instances and the results are compared with the aid of statistical tests in order to determine whether there are outperforming algorithms.

The first phase is presented in this chapter, while the remaining two are the subject of Chapter 6. In Section 5.1, we discuss the parametrization of DSSR and NGRR. Section 5.2 describes the hybrid labeling algorithms and how they inherit the parameters of the previous two algorithms. Finally, Section 5.3 describes the main features of the BP framework in which each labeling algorithm is embedded.

## 5.1 Features of DSSR and NGRR

In this section, we discuss the way in which we can generalize and then parametrize different algorithmic components of DSSR and NGRR. More specifically, for each algorithmic strategy or numerical value that can be chosen by the decision maker during the algorithm design phase, we assign a parameter and specify a set of options or a range of numerical values, respectively.

### 5.1.1 DSSR Initialization Strategies

As we have seen in Chapter 3, Righini and Salani (2009) explore different initialization strategies for the set $\Theta$ containing the critical vertices, i.e., the vertices on which the elementarity constraints are active during the execution of the labeling algorithm. Their idea is to determine at the initialization of the algorithm some critical vertices that are likely to be visited multiple times.

In order to determine at the initialization of the algorithm some critical vertices that are likely to be visited multiple times, they introduce the measures HCA, TCA, WHCA, and WTCA, described in Section 3.3.2.

At the start of the algorithms, the vertices are sorted in non-increasing order according to the selected measure and the first $m$ vertices are inserted into $\Theta$, where $m = \lceil \alpha_\Theta n \rceil$ with $\alpha_\Theta \in ]0, 1[$. Let us denote the sets constructed in this way by $HCA_m$, $TCA_m$, $WHCA_m$, and $WTCA_m$. We also consider the mixed strategy suggested by Righini and Salani (2009) in addition to these four strategies, where the critical vertex set $MIX_m$ is initialized by the intersection of these four sets, i.e., $MIX_m = HCA_m \cap TCA_m \cap WHCA_m \cap$

$WTCA_m$. In our algorithms, we decide on the strategy to initialize the set $\Theta$ by introducing two parameters: `dssr_init_s`, which controls which one of the five strategies is implemented, and `dssr_init_n`, which controls the value of $\alpha_\Theta$. Parameter `dssr_init_s` can be assigned one of the following values: `hca`, `tca`, `whca`, `wtca`, `mixed`, and `none`, the latter indicating that no initialization strategy is used.

### 5.1.2 DSSR Insertion Strategies

As mentioned in Chapter 3, in order to update the set $\Theta$ at the end of each DSSR iteration, Boland et al. (2006) examine the following strategies for inserting new vertices into $\Theta$ when the best path obtained at the end of an iteration is not elementary:

- HMO: insert only the vertex that is visited the most;
- HMO-All: insert all vertices visited the maximum number of times;
- MO-All: insert all vertices visited more than once in the optimal path.

In our algorithms, at each iteration of our CG procedure, we add at most $N_{CG}$ negative reduced cost columns into the RMP (see Section 5.3.3). In order to generalize the three strategies of Boland et al. (2006), we make the following two independent decisions:

(1) the number of negative reduced cost paths to examine so as to detect vertices visited multiple times;

(2) the number of such vertices to add into the set $\Theta$.

For the first decision given in (1), we suggest three alternatives: examining only the best path, examining the highest possible number of paths $N_{INSMAX}$, or examining an intermediate number of paths ranging from one to $N_{INSMAX}$. $N_{INSMAX}$ is determined as the number of negative reduced cost paths generated in the current iteration, unless this number is higher than $N_{CG}$, which is the maximum number of columns that can be inserted in the RMP (see Section 5.3.3), in which case $N_{INSMAX}$ is set to $N_{CG}$. An alternative is chosen by assigning one of the following respective values to a parameter `dssr_path_s`: *one-path* (`1-p`), *all-paths* (`all-p`), or *in-between* (`in-btw`). If the third alternative is chosen, the algorithm examines at most $\lceil \alpha_P N_{CG} \rceil$ paths, where the value of $\alpha_P \in \, ]0,1[$ is controlled by parameter `dssr_path_n`.

For the second decision given in (2), we define three alternatives: inserting only the vertex visited the most (analogously to HMO), inserting all vertices visited more than once (analogously to MO-All), or inserting an intermediate number of vertices. A parameter `dssr_node_s` determines the alternative to be employed and can assume the respective values: *one-node* (`1-n`), *all-nodes* (`all-n`), or *in-between* (`in-btw`). Let $N_V$ be the set of vertices to insert with the second alternative. For the third alternative, we sort all the $N_V$ candidate vertices in decreasing order of the number of times they are visited and we insert the first $\lceil \alpha_V N_V \rceil$ vertices into the critical vertex set. The value $\alpha_V \in ]0, 1[$ is determined by parameter `dssr_node_n`.

### 5.1.3 NGRR

The effectiveness of NGRR hinges on the fact that it attempts to prevent the generation of cycles that have a low total cost, since routes with expensive cycles are unlikely to appear in the optimal solution. Thus, the criteria used to construct the *ng*-neighborhoods $M_i$ strongly influence the performance of the algorithm.

We suggest the following three construction strategies, each using a different metric $D_k(i, j)$ $(k = 1, 2, 3)$ depending on the travel time $t_{ij}$. The strategy to be implemented is chosen by assigning the corresponding value to parameter `ng_m`.

1. The travel time (`tt`): $\Delta_1(i, j) \coloneqq t_{ij}$;
2. The *cheap cycle risk* (`ccr`): $\Delta_2(i, j) \coloneqq (\bar{t}_{ij} + \bar{t}_{ji})/o(i, j)$, where $o(i, j)$ shows the overlap of time windows of $i$ and $j$;
3. The mixed measure (`mix`): $\Delta_3(i, j) \coloneqq \alpha_M \Delta_1(i, j) + (1 - \alpha_M)\Delta_2(i, j)$, where $\alpha_M \in ]0, 1[$.

The first strategy (`tt`) simply constructs the *ng*-neighborhoods using only the travel times $t_{ij}$, as it is done in the existing studies adopting NGRR, whereas the second one (`ccr`) utilizes the ratio of the duration of a 2-cycle formed by vertices $i$ and $j$ to the overlap of their time windows. The observation behind this measure is that the smaller the duration of the 2-cycle is compared to the overlap, the easier it is to feasibly visit $i$ and $j$ multiple times. Lastly, the metric employed by the third strategy (`mix`) is an affine combination of the first two strategies. The value of $\alpha_M$ is controlled by an additional parameter `ng_mix`. Finally, if a 2-cycle with vertices $i$ and $j$ is guaranteed to be infeasible, then we let $D_k(i, j) \coloneqq +\infty$ for $k = 1, 2, 3$. That is to say, we set $D_k(i, j)$ equal to $\infty$ for $k = 1, 2, 3$, if either $a_i + \bar{t}_{ij} > b_j$ or $a_j + \bar{t}_{ji} > b_i$ or $\max\{a_i + \bar{t}_{ij}, a_j\} + \bar{t}_{ji} > b_i$ and $\max\{a_j + \bar{t}_{ji}, a_i\} + \bar{t}_{ij} > b_j$.

The size of *ng*-neighborhoods is another critical issue influencing the effectiveness of the algorithm. Obviously, the larger the size of the neighborhoods is, the less is the probability of having cycles in the optimal path. For this purpose, we let $|M_i| = \lceil \alpha_S n \rceil$ with $\alpha_S \in ]0,1[$ for each vertex $i \in N$. The value of $\alpha_S$, is controlled by parameter `ng_s`. At the initialization of BP, all vertices are sorted in increasing order according to one of the three strategies given above and the first $\lceil \alpha_S n \rceil - 1$ neighbors are added to $M_i$.

## 5.2 Hybrid Algorithms

Both DSSR and NGRR rely on the relaxation of elementarity conditions. However, each algorithm relaxes the state space by employing a different approach. The DSSR follows what we call a *global* approach by maintaining a single critical vertex set for all vertices, while NGRR uses a *local* approach by defining an individual critical vertex set for each single vertex of the graph. With the aim of studying the effectiveness of these approaches, we investigate five additional algorithms, namely *global ng-DSSR*, *local ng-DSSR*, *local DSSR*, *elementary global ng-DSSR*, and *elementary local ng-DSSR*. Each one of these algorithms, most of which we have already described in Section 3.5, hybridizes DSSR and NGRR in a different way while adhering to either the global or the local approach. Additionally, the first two hybrid algorithms may provide non-elementary *ng*-paths at the end, whereas the others yield only elementary paths.

The algorithms can be described as follows.

- **Global *ng*-DSSR (NG-DSSR-G):** This first hybrid algorithm basically combines *ng*-route relaxation and DSSR by utilizing both a critical vertex set $\Theta$ and *ng*-neighborhoods $M_i$ for each vertex $i \in N$. It implements the basic *ng*-route relaxation procedure with the difference that visiting vertex $j$ from vertex $i$ is prohibited only when $j$ belongs to both the critical vertex set $\Theta$ and the *ng*-neighborhood $M_i$ of $i$, in other words only when $j \in \Theta \cap M_i$. Furthermore, a vertex $k$ visited multiple times is designated as critical and inserted in the set $\Theta$ only if it belongs to an invalid cycle. The algorithm incorporates all parameters of DSSR and *ng*-route relaxation described in the previous sections.
- **Local *ng*-DSSR (NG-DSSR-L):** As we have seen earlier, this procedure was originally described by Martinelli et al. (2014) and Dayarian et al. (2015b), and forgoes the use of the critical set $\Theta$ while calling forth the notion of *applied* neighborhoods $\overline{M}_i \subseteq M_i$ for all $i \in N$, which are initially empty. The *ng*-neighborhoods $\{M_i\}_{i \in N}$ are used to check at the end of each DSSR iteration whether the optimal paths are *ng*-paths. If an invalid *ng*-path is detected, the applied neighborhoods

$\{\overline{M}_i\}_{i \in N}$ are updated according to the rule described in Section 3.5, which we repeat here.

Let $C = (i - k_1 - \cdots - k_l - i)$ be an invalid cycle, i.e., a cycle that cannot appear in an *ng*-path. Then the repeated vertex $i$ must belong to the *ng*-neighborhood $M_{k_h}$ of all $k_h$ with $h = 1, \ldots, l$, i.e., $i \in M_{k_h}$ for all $h = 1, \ldots, l$. However, for such a cycle to occur, there exists at least one $h \in \{1, \ldots, l\}$ such that $i \notin \overline{M}_{k_h}$. Hence, vertex $i$ is included in the applied neighborhood $\overline{M}_{k_h}$ of each $k_h$ with $h = 1, \ldots, l$ so as to prevent cycle $C$ re-emerging in any path in subsequent iterations.

We label this algorithm as local, since we do not have a global critical vertex set such as $\Theta$. As in Dayarian et al. (2015b), due to the high likelihood of the recreation of the same invalid cycles at different nodes of the BP tree, the applied neighborhoods are initialized as empty sets only at the root node and maintained throughout the execution of the entire BP algorithm.

This algorithm inherits all the parameters described previously, except the following ones: `dssr_init_s` and `dssr_init_n`, since the applied neighborhoods are initialized as empty sets; `dssr_node_s` and `dssr_node_n`, since we apply the update procedure described above for every invalid cycle we examine.

- **Local DSSR (DSSR-L):** This hybrid algorithm is derived by applying the local approach exclusively to DSSR, and is also introduced in Martinelli et al. (2014). To this end, we define a collection of critical sets $\{\Theta_i\}_{i \in V}$ analogous to the applied neighborhoods used for NG-DSSR-L. We initialize these sets employing the initialization strategies of the global critical vertex set $\Theta$ used for DSSR. Furthermore, the update procedure is similar to the one used in the NG-DSSR-L algorithm. The main difference lies in the detection of critical vertices: every repeated vertex giving rise to a cycle is designated as critical, that is, if a cycle $C = (i - k_1 - \cdots - k_l - i)$ is detected, then vertex $i$ is added to the critical vertex set $\Theta_{k_h}$ for all $k_h$ with $h = 1, \ldots, l$. Differently from NG-DSSR-L, the critical sets are clearly not restricted to be subsets of any neighborhoods.

  This algorithm uses all the parameters defined for DSSR except the parameters `dssr_node_s` and `dssr_node_n`, since vertex insertion strategies for DSSR do not apply in the same way.

- **Elementary Global** and **Local *ng*-DSSR (NG-DSSR-G-E and NG-DSSR-L-E):** These two algorithms are extensions of NG-DSSR-G and NG-DSSR-L and based on a technique proposed by Dayarian et al. (2015a) for guaranteeing the elementarity of paths. The NG-DSSR-G and NG-DSSR-L algorithms terminate when the best path obtained is an *ng*-path, which may not be elementary. Conversely, the respective extensions do not terminate until the best path obtained is elementary: in case the best path is a non-elementary *ng*-path, the algorithms simply switch to the respective variant of DSSR (differently from the approach of Dayarian et al. (2015a), where the second phase is always the classic DSSR) in order to achieve elementarity. These two exact algorithms inherit the same parameters of their respective *ng*-DSSR versions.

Since the aforementioned studies generally conclude that each of the considered procedures has been found to perform better than the standard labeling algorithm for the ESPPRC, we do not take it into account in our computational study.

## 5.3 Shared Algorithmic Features

Each labeling algorithm is embedded in a BP framework in order to solve the VRPTW and the VRPTWWTC. Furthermore, each algorithm uses the label structure, the associated REFs, and the dominance rules described in Chapter 3 when solving the VRPTW, and the ones in Chapter 4 when solving the VRPTWWTC. We set $c_{ij} = t_{ij}$ for each arc $(i, j) \in A$ when solving the VRPTW, as is common practice in the literature.

There are additional aspects shared by all algorithms that we describe in this section.

### 5.3.1 Heuristic Dynamic Programming Algorithm

In order to quickly find promising columns at the start of a CG iteration, we devise two heuristic DP algorithms for solving the pricing problem, one for the VRPTW and the other for the VRPTWWTC. Each heuristic is based on the respective version of bounded bidirectional DP, and simply consist in a relaxation of the label dominance rules. Consequently, the number of labels generated reduces significantly, since the number of conditions that need to be tested is considerably cut down.

The heuristic for the VRPTW relaxes all dominance criteria, except the one related to the reduced cost. Therefore, for each customer $i$, the only existing labels are the ones associated to partial paths ending at $i$ with the least reduced cost.

The heuristic for the VRPTWWTC is based on relaxing the dominance rules given in Proposition 4.3.3. The DP heuristic omits the dominance criteria related to the earliest feasible arrival time $\mu_i$, the minimum possible duration $\zeta_i$, the accumulated vehicle load $q_i$, and the elementarity resources $(E_i^k)_{k \in N}$ of a partial path arriving at customer $i$. Thus, it only takes into consideration the improved dominance criterion introduced in Proposition 4.3.3, which compares the sum $\delta_i$ of dual prices and the latest feasible starting time $\nu_i$ from the depot. Since these heuristics do not take into account the set of elementarity resources, which are the main element that the algorithms under our consideration manipulate in order to accelerate the label extension procedure, we do not analyze their overall impact on the BP procedure.

### 5.3.2 Strengthened Bounding for Bidirectional Label Extension

In Chapter 3, we discussed how bounding label extensions is critical for the effectiveness of bidirectional DP, since it prevents the generation of unnecessary labels (thus reducing computational cost), without losing the guarantee of optimality, i.e., the guarantee of generating the optimal path. As we have seen, bounding is performed by extending only the labels such that $\tau_i \leq T/2$ in the case of the VRPTW, where $\tau_i$ is the total duration for label $l_i$ and $T$ is the latest feasible arrival time at the depot (see Section 3.2.2), and $\zeta_i \leq S/2$ in the case of the VRPTWWTC, where $\zeta_i$ is the minimum total duration for label $\lambda_i$ and $S$ is the maximum allowed duration (see Section 4.4). Under this rule, it is then possible to retain labels such that $\tau_i > T/2$ and $\zeta_i > S/2$ respectively (such labels are only prevented from being extended further.)

In our study, we observed that it is possible to adopt a slightly stronger bounding rule, which allows us to discard a higher number of labels without losing guarantee of optimality. According to this bounding rule, we can safely discard every label such that $\tau_i > T/2$ for the VRPTW and $\zeta_i > S/2$ for the VRPTWWTC. The following propositions prove the validity for this rule for each problem variant we consider.

**Proposition 5.3.1.** Let $P = (0 = v_0, v_1, \ldots, v_{m-1}, v_m = n+1)$ be a feasible path. Then, there exists $k \in \{0, \ldots, m\}$ such that $P$ can be concatenated along arc $(v_k, v_{k+1})$ with the forward label $l_k^f$ associated with $v_k$ having $\tau_k^f \leq T/2$ and the backwards label $l_{k+1}^b$ associated with $v_{k+1}$ having $\tau_{k+1}^b \leq T/2$.

*Proof.* By the way of contradiction, assume that for each $k \in \{0, \ldots, m-1\}$, it holds that $\tau_k^f > T/2$ or $\tau_{k+1}^b > T/2$. Let $k^* \in \{0, \ldots, m-1\}$ be the maximal index such that $\tau^f_{k^*} \leq T/2$. We then assume that $\tau^b_{k^*+1} > T/2$, and it holds that

$$\tau^f_{k^*+1} = \max\{a_{k^*+1}, \tau^f_{k^*} + s_{k^*} + t_{k^*,k^*+1}\} > T/2.$$

If $\tau^f_{k^*+1} = \tau^f_{k^*} + s_{k^*} + t_{k^*,k^*+1} > T/2$, then we derive from equation (3.27) that, since we assume that $\tau^b_{k^*+1} > T/2$, the total duration $\overline{\tau}$ of path $P$ would be

$$\overline{\tau} = \tau^f_{k^*} + s_{k^*} + t_{k^*,k^*+1} + s_{k^*+1} + \tau^b_{k^*+1} > T,$$

which contradicts our hypothesis that $P$ is feasible.

Let us now consider the case $\tau^f_{k^*+1} = a_{k^*+1}$. Recall that we impose that $\tau^b_{k+1} \leq T - a^b_{k+1}$ for each $k \in \{0, \ldots, m-1\}$ in order to obtain a feasible backward label. Then, since $a_{k^*+1} = \tau^f_{k^*+1} > T/2$, it holds that

$$\tau^b_{k^*+1} \leq T - a^b_{k^*+1} = T - a_{k^*+1} - s_{k^*+1} \leq T/2,$$

contradicting again our hypothesis. $\square$

**Proposition 5.3.2.** Let $P = (0 = v_0, v_1, \ldots, v_{m-1}, v_m = n+1)$ be a feasible path. Then, there exists $k \in \{0, \ldots, m\}$ such that $P$ can be concatenated along arc $(v_k, v_{k+1})$ with the forward label $\lambda^f_k$ associated with $v_k$ having $\zeta^f_k \leq S/2$ and the backwards label $\lambda^b_{k+1}$ associated with $v_{k+1}$ having $\zeta^b_{k+1} \leq S/2$.

*Proof.* Let $k^* \in \{1, \ldots, m\}$ be the maximal index such that $\zeta^f_{k^*-1} \leq S/2$. Thus, $\zeta^f_{k^*} > S/2$. It suffices to show that $\zeta^b_{k^*} \leq S/2$, given the symmetry of the label extension procedures. Assume that this is not the case, i.e., $\zeta^b_{k^*} > S/2$, and let us recall that $\zeta^f_k = \max\{\theta_k, \mu_k - \nu_k\}$, and analogously, $\zeta^b_k = \max\{\theta^b_k, \mu^b_k - \nu^b_k\}$, for all $k \in \{0, \ldots, m\}$.

We consider the following three cases:

1. $\zeta^f_{k^*} = \theta_{k^*}$ and $\zeta^b_{k^*} = \theta^b_{k^*}$;
2. $\zeta^f_{k^*} = \theta_{k^*}$ and $\zeta^b_{k^*} = \mu^b_{k^*} - \nu^b_{k^*}$;
3. $\zeta^f_{k^*} = \mu_{k^*} - \nu_{k^*}$ and $\zeta^b_{k^*} = \mu^b_{k^*} - \nu^b_{k^*}$.

We do not need to prove the case in which $\zeta^f_{k^*} = \mu_{k^*} - \nu_{k^*}$ and $\zeta^b_{k^*} = \theta^b_{k^*}$ since, because of the symmetry of the extension procedures, it can be proven analogously to case 2.

**Case 1**. The total duration $\overline{\tau}$ of path $P$ would be

$$\overline{\tau} \geq \theta_{k^*} + \theta^b_{k^*} = \zeta^f_{k^*} + \zeta^b_{k^*} > S/2 + S/2,$$

which contradicts that $P$ is a feasible path.

**Case 2**. Since $\zeta^f_{k^*} = \theta_{k^*}$, there is no waiting time on the forward partial path, while the opposite is true for the backward partial path since $\zeta^b_{k^*} = \mu^b_{k^*} - \nu^b_{k^*}$. We can determine that the forward time slack of the forward partial path is $F_1 = \theta_{k^*} - (\mu_{k^*} - \nu_{k^*}) > 0$ and that the total forward time slack of the path is $F = \min\{F_1, -\xi_{(k^*+1)}\}$, according to equation (4.22) (the forward time slack of the backward partial path $F_2$ and the waiting time on the forward partial path $w_1$ are both 0), which implies that $\xi_{(k^*+1)} + F \leq 0$. We know that in this problem, in order for the concatenation to be feasible we need $\xi_{(k^*+1)} \leq 0$. This, together with the fact that $w_1 = 0$ and the waiting time on the backwards partial path $w_2$ is positive, implies that the total waiting on the path is $w - F = w_2 - \xi_{(k^*+1)} - F$, according to Table 4.1. The total duration $\overline{\tau}$ of path $P$ is therefore

$$\begin{aligned}\overline{\tau} &= \theta_{k^*} + \theta^b_{k^*} + w_2 - \xi_{(k^*+1)} - F\\ &\geq \theta_{k^*} + \theta^b_{k^*} + w_2 = \theta_{k^*} + \mu^b_{k^*} - \nu^b_{k^*}\\ &> S/2 + S/2.\end{aligned}$$

**Case 3**. We know that in order to have a feasible concatenation between the partial paths, we need to impose $\mu_{k^*} \leq M - \mu^b_{k^*}$, since $M - \mu^b_{k^*}$ is the latest feasible departure time from vertex $v_{k^*}$. We also know that the difference between the earliest feasible arrival time at the depot $M - \nu^b_{k^*}$ and the latest feasible departure time from the depot $\nu_{k^*}$ is a lower bound to the overall duration $\overline{\tau}$ of the path $P$, i.e., $M - \nu^b_{k^*} - \nu_{k^*} \leq \overline{\tau} \leq S$. However, then we have

$$\begin{aligned}S/2 < \mu^b_{k^*} - \nu^b_{k^*} &\leq M - \mu_{k^*} - \nu^b_{k^*}\\ &\leq M - \mu_{k^*} + S + \nu_{k^*} - M = S - (\mu_{k^*} - \nu_{k^*}) < S/2.\end{aligned}$$

□

### 5.3.3 Additional Branch-and-Price Features

In order to decide the maximum number of negative reduced cost columns that are generated during the label concatenation phase, we define the quantity $N_C = \lceil \alpha_C n \rceil$, where the value of the multiplier $\alpha_C \in \, ]0, 10000]$ is controlled by parameter `n_conc`. We determined the upper bound of this range through preliminary experiments. Moreover, at each iteration of the CG procedure, we impose the maximum number of negative reduced cost columns that can be inserted in the RMP as $N_{CG} = \lceil \alpha_{CG} N_C \rceil$, with the value of the multiplier $\alpha_{CG} \in \, ]0, 1]$ being controlled by parameter `n_col`.

Table 5.1: Algorithmic parameters.

| Parameter | DSSR | NGRR | NG-DSSR-G | NG-DSSR-L | DSSR-L | NG-DSSR-G-E | NG-DSSR-L-E | Range |
|---|---|---|---|---|---|---|---|---|
| tree_trav[†] | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | {breadth, depth, best} |
| n_col | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | $]0,1]$ |
| n_conc | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | $]0,10000]$ |
| dssr_init_s | ✓ |  | ✓ |  | ✓ | ✓ |  | {none, hca, tca, whca, wtca, mix} |
| dssr_init_n | ✓ |  | ✓ |  | ✓ | ✓ |  | $]0,1[$ |
| dssr_path_s | ✓ |  | ✓ | ✓ | ✓ | ✓ | ✓ | {1-p, in-btw, all-p} |
| dssr_path_n | ✓ |  | ✓ | ✓ | ✓ | ✓ | ✓ | $]0,1[$ |
| dssr_node_s | ✓ |  | ✓ |  |  | ✓ |  | {1-n, in-btw, all-n} |
| dssr_node_n | ✓ |  | ✓ |  |  | ✓ |  | $]0,1[$ |
| ng_m |  | ✓ | ✓ | ✓ |  | ✓ | ✓ | {tt, ccr, mix} |
| ng_s |  | ✓ | ✓ | ✓ |  | ✓ | ✓ | $]0,1[$ |
| ng_mix |  | ✓ | ✓ | ✓ |  | ✓ | ✓ | $]0,1[$ |

[†] Parameter without root node duplicate.

We implement several strategies to traverse the BB tree, namely depth-first search, breadth-first search, and best-first search. In best-first search, we choose to evaluate first the nodes whose parent node yields the smallest lower bound, and in case of *ex aequo* we choose the highest number of arc flow variables set to 1, since such a node is expected to be evaluated faster. Depth-first search consists in fully exploring each branch of the tree before backtracking. In order to implement it, all the nodes that need to be evaluated are recorded into a stack, which is a data structure that follows the last-in-first-out principle. Thus, each time a child node is generated after having processed the current node, it is going to be the next node to be evaluated. Conversely, breadth-first search traverses the tree layer-wise, exploring all neighboring nodes before going deeper down the tree. To implement it, it suffices to use a queue, which follows the first-in-first-out principle, instead of a stack. Hence, if the current node generates children, these are going to be evaluated after all the nodes that are currently in the queue. In both depth-first and breadth-first search, in the case two child nodes are generated, we evaluate first the one for which the relevant arc-flow variable has been fixed to 1. A parameter named `tree_trav` controls the choice of the tree traversal strategy.

Table 5.1 outlines all the parameters of our BP algorithms. For each of those, except for `tree_trav`, we use an additional parameter that controls the associated quantity when solving the root node.

Finally, to solve the RMP at each CG iteration, we use a commercial MIP solver, CPLEX 12.6.3. In order to obtain an upper bound early in the execution of the algorithm, we call CPLEX to solve the RMP with the integrality constraints immediately after obtaining a valid lower bound at the root node, using all the columns available in the pool. In order to guarantee that all the paths used in the solution providing this upper bound are elementary when using *ng*-route algorithms, we use the set partitioning formulation instead of the set covering one. We also implement the option to use CPLEX to *recover* an upper bound, i.e., search for an integer solution with the columns generated so far, if the algorithm fails to solve the RMP at the root node within its time limit. In this case, an additional time limit of three hours is provided to CPLEX to attempt recovery. In the next chapter, we specify when this option is used.

## 5.4 Conclusions

In this chapter, we have discussed the details of the algorithms under our consideration, i.e., DSSR, NGRR, and five hybrid procedures. More specifically, we have specified how DSSR and NGRR can be parametrized, the way in which the hybrid algorithms inherit features and parameters from these two procedures, and the common aspects of the BP framework in which they are embedded.

As we are going to see in the next chapter, the two different paradigms, in which these algorithms manipulate the elementarity constraints during label extensions (which we denote as the *global* and the *local* approaches), are of key importance within the scope of this study.

# Chapter 6

# Computational Experiments

In this chapter, we first introduce the experimental methodology we have adopted in order to compare the algorithms described in the previous chapter. Then, we present and discuss the obtained results.

As mentioned in Chapter 5, our methodology is composed of two phases: a *parameter tuning* phase and a *benchmarking* phase. Section 6.1 describes the former, in which we use a software package for automated tuning in order to obtain the best possible parameter configuration for each algorithm under our consideration, for both VRPTW and the VRPTWWTC. In Section 6.2 we describe the latter phase, in which we first run each algorithm on a set of benchmark instances, and then we analyze the results with statistical tests.

Sections 6.3, 6.4 and 6.5 present additional analysis. More specifically, the first one compares the parameter values of the best configurations obtained for the more performant algorithms according to the benchmarking phase, the second one performs additional tuning and analysis of two of the best algorithms for the VRPTWWTC, with respect to the number of customers in an instance, and the third presents a comparison of the VRPTW and the VRPTWWTC with regard to the respective objective function values and elapsed computing times.

## 6.1 Tuning Phase

It is often the case in the literature that algorithmic parameters are manually chosen by the authors after several preliminary experiments. As algorithmic complexity increases, so does the potential number of parameters, and consequently manual tuning becomes less practical. In recent years, several techniques for automated parameter tuning have arisen, which allow for a more systematic approach to obtain good configurations. In our view, adopting such a technique is necessary in order to rigorously compare different algorithms with a substantial number of parameters, since the poor performance of an algorithm might be due to a bad configuration, rather than a fundamental flaw in its design. This section describes the method and toolset we use in order to obtain a good parameter configuration for each of the algorithms under our consideration, for both problem variants.

### 6.1.1 The `irace` Package

The main tool we utilize for parameter tuning is `irace`, developed by López-Ibáñez et al. (2016), a software package developed using the R programming language for automated algorithmic configuration that implements the *iterated racing procedure.*

In order to obtain good parameter configurations for a given algorithm and its parameter space, `irace` randomly samples numerous possible configurations and tests them on a set of training instances provided by the user. It relies on a user-provided measure in order to compare the performance of two configurations on the same instance. At each iteration, `irace` performs statistical tests to eliminate underperforming configurations and resamples new ones derived from the best configurations obtained so far. More specifically, `irace` adopts the non-parametric Friedman's two-way analysis of variance by ranks and the associated post-hoc analysis of Conover (1999), with a significance level of 0.05. The former is a family-wise test, in the sense that it works on all the candidate configurations simultaneously; if the test is positive, in other words, if it reports a significant difference in quality among the configurations, the post-hoc pair-wise analysis is performed in order to determine the best configurations.

Additionally, `irace` implements Student's t-test as an alternative to Friedman's test. The former is recommended when the magnitude of the difference between the performances of two configurations has to be taken into account, and thus when the best configurations should obtain the best average performance, while the latter is recommended when the best performing configuration should be among the best in as many instances as possible. This is because Friedman's test is non-parametric, i.e., no assumption is made on the distribution of data. Indeed, it performs the analysis on a ranking of the data: for each instance, the performance of each tested configuration is ranked from the smallest to the largest, with the smallest one having the best performance, as explained in Section 6.1.3. The configuration that performs best in as many instances as possible has thus the lowest average or total rank, computed over all the instances.

There are several possible stopping conditions for the procedure. It can either stop when a minimum number of surviving configurations is reached, or when a computational budget has been depleted, where the budget can either be total computing time, or the maximum number of experiments `irace` is allowed to run. Here, an experiment is defined as the execution of a specific parameter configuration on a specific instance. Upon termination, `irace` returns a set of configurations, called *elite* configurations, which are statistically equivalent to each other with regard to their performance. The number of elite configurations returned by `irace` is determined by the size of the parameter space. They are provided in increasing order according to the ranking statistic used by Friedman's test.

### 6.1.2 The Training Instance Set

In the field of machine learning, it is well-known that separating the data set used for training a machine from the data used to test its quality is necessary in order to avoid the phenomenon called *over-fitting*, which consists in the overestimation of the performance of the machine. In the case of `irace`, the analogous phenomenon is called *over-tuning*, and consists in the over specialization of the tuned algorithm for a specific set of instances (Birattari, 2009).

In order to avoid over-tuning, we need to have different sets of instances for tuning and benchmarking, and thus we consider the instances provided by Gehring and Homberger (2001) for tuning purposes and the widely used Solomon's instances for the VRPTW (Solomon, 1987) for benchmarking. Both instance sets are structured according to the same criteria. First, they are divided into three classes, namely clustered (C), random (R), and random-clustered (RC) instances, based on the spatial distribution of customers on the Euclidean plane. In addition, they are divided into two series depending on the length of the time windows of the customers and the planning horizon. The instances of Series 1 present narrower windows and a shorter horizon than the ones of Series 2, and are therefore easier to solve with exact algorithms.

In order to build a diversified tuning set, and thus attempt to remove bias depending on the size or the structure of the instances, we randomly sample two instances from each class of Gehring and Homberger's instances by considering either the first 25 or the first 50 customers. Due to hardware and time constraints, we only consider instances of Series 1 with at most 50 customers. As a result, we obtain six tuning instances. While determining these instances, for each candidate instance, a preliminary test was made with a random configuration of one of the BP algorithms in order to perceive its computing time. A candidate instance was not included in the tuning set if its computing time was under ten seconds. The purpose of this is to avoid including instances that can be solved easily by all algorithms, since these cannot offer meaningful information when performing the statistical tests.

It is worth to observe that, while we use Solomon's instances for benchmarking (given their widespread use in the literature) and the similarly structured Gehring and Homberger's set for the tuning phase, it is an equally valid approach to consider combinations of additional sets from those that have been proposed in the literature, such as the instances of Cordeau et al. (2001), as long as one remains aware of the danger of over-tuning.

### 6.1.3 Usage of `irace`

In order to identify the outperforming configurations, `irace` requires a scalar quantity as a performance measure for all algorithms; more specifically, a cost function for a given configuration $\theta$ on an instance $i$. Since the algorithms employ a BP framework, we have to impose a time limit. Hence, we cannot use computing time alone to determine the algorithmic performance. Otherwise, it would be impossible to differentiate between two configurations that reach the time limit but report different upper bounds. Thus, we define the following performance measure $t'(i, \theta)$ for a configuration $\theta$ on an instance $i$:

$$t'(i,\theta) = \begin{cases} t(i,\theta) & \text{if } t(i,\theta) < TL \\ TL + UB(i,\theta) & \text{if the root node is solved and } TL \text{ is reached} \\ M & \text{otherwise,} \end{cases}$$

where $t(i, \theta)$ is the elapsed time until completion in seconds, $UB(i, \theta)$ is the best upper bound obtained, and $M$ is a large positive constant. Here, we set the time limit $TL$ to 10800 seconds (three hours), and $M >> 10800$. Note that in this phase, if an algorithm fails to solve even the root node within the time limit, we do not attempt an upper bound recovery, as described in the previous chapter.

Recall that the statistical tests we adopt are non parametric, and that they are based on a ranking statistic. Hence, the only property that is required in order for our performance measure to represent the case that configuration $\theta_1$ performs better than configuration $\theta_2$ on instance $i$ is that $t'(i, \theta_1) < t'(i, \theta_2)$. In fact, the magnitude of the difference $t'(i, \theta_2) - t'(i, \theta_1)$ is

Table 6.1: VRPTW — Elementary algorithms — 3 hours.

| | | DSSR | | | | DSSR-L | | | | NG-DSSR-G-E | | | | NG-DSSR-L-E | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Class | NI | S | PS | Time | Gap (%) | S | PS | Time | Gap (%) | S | PS | Time | Gap (%) | S | PS | Time | Gap (%) |
| C1–25 | 9 | 9 | | 10.23 | | 9 | | 5.27 | | 9 | | 140.3 | | 9 | | 10.03 | |
| R1–25 | 12 | 12 | | 1.20 | | 12 | | 1.23 | | 12 | | 1.87 | | 12 | | 1.20 | |
| RC1–25 | 8 | 8 | | 5.26 | | 8 | | 3.80 | | 8 | | 18.17 | | 8 | | 4.46 | |
| C1–50 | 9 | 8 | | 13.84 | | 9 | | 105.4 | | 8 | | 82.46 | | 9 | | 184.9 | |
| R1–50 | 12 | 11 | 1 | 435.8 | 4.59 | 11 | 1 | 193.8 | 4.47 | 11 | 1 | 1015 | 4.59 | 11 | 1 | 154.5 | 3.99 |
| RC1–50 | 8 | 2 | 6 | 1863 | 12.01 | 2 | 6 | 1682 | 12.64 | 2 | 6 | 3442 | 12.11 | 2 | 6 | 1723 | 11.0 |

irrelevant for the purposes of the test. The chosen measure allows us to establish that configuration $\theta_1$ performs better than configuration $\theta_2$ on instance $i$ if one of the following occurs:

- both $\theta_1$ and $\theta_2$ terminate before the time limit and the elapsed time of $\theta_1$ is smaller than the one of $\theta_2$;
- $\theta_1$ terminates before the time limit while $\theta_2$ does not;
- both $\theta_1$ and $\theta_2$ do not terminate before the time limit but are able to evaluate the root node and obtain an upper bound, and the upper bound of $\theta_1$ is smaller than the one of $\theta_2$;
- $\theta_1$ is able to evaluate the root node and obtain an upper bound while $\theta_2$ is not.

As the stopping condition of the tuning procedure, we adopt a computational budget of 5000 experiments per algorithm. We implemented all the algorithms in Java 8 and performed the tunings on two computing clusters: the first with AMD Bulldozer 6272 processors, 2.1 GHz and 4 GB of RAM per core, and the second one with Intel E5–2650 processors, 2.0 GHz and 4 GB of RAM per core, both operating under Linux.

Finally, once the tuning phase is complete, we pick one configuration to study in the benchmarking phase among the ones returned by `irace`, which are six in number for each algorithm. For this purpose, we choose the first one returned by `irace`, i.e., the one with the lowest ranking.

## 6.2 Benchmarking Phase

In this phase, we perform up to two rounds of benchmarking and statistical analysis for each problem variant. In each round, we first run each of the considered algorithms with the obtained configuration on a set of benchmark instances, and then we analyze the results using a similar procedure to the one used by `irace` for testing configurations.

Table 6.2: VRPTW — *ng*-route algorithms — 3 hours.

| | | NGRR | | | | NG-DSSR-G | | | | NG-DSSR-L | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Class | NI | S | PS | Time | Gap (%) | S | PS | Time | Gap (%) | S | PS | Time | Gap (%) |
| C1–25 | 9 | 9 | | 1.49 | | 9 | | 3.22 | | 9 | | 2.25 | |
| R1–25 | 12 | 12 | | 0.95 | | 12 | | 2.02 | | 12 | | 1.14 | |
| RC1–25 | 8 | 8 | | 3.22 | | 8 | | 3.49 | | 8 | | 3.91 | |
| C1–50 | 9 | 9 | | 40.39 | | 9 | | 491.0 | | 9 | | 55.56 | |
| R1–50 | 12 | 11 | 1 | 495.1 | 5.22 | 10 | 2 | 65.67 | 3.75 | 11 | 1 | 127.6 | 4.34 |
| RC1–50 | 8 | 2 | 6 | 2478 | 12.27 | 2 | 6 | 3963 | 10.63 | 2 | 6 | 1683 | 11.8 |

More specifically, considering the same performance measure $t'(i, \theta)$ used in the tuning phase, we apply Friedman's test. Under the null hypothesis of this test, the algorithms have negligible performance differences, which means that when it is rejected, there is at least one algorithm that performs significantly better than at least one of the others. Once again, we adopt the standard significance level of 0.05. Thus, if this process yields a $p$-value significantly smaller than 0.05, we then reject the null hypothesis.

If this is the case, since Friedman's test is a family-wise test, in other words, since it is applied to the results of all the algorithms simultaneously, we need to carry out a post-hoc analysis. To this end, we now make use of Wilcoxon's signed-rank test (see Conover, 1999), which performs pairwise comparisons: for each pair of algorithms, if the associated $p$-value returned by the test is smaller than the significance level of 0.05, it can be concluded that one of the two algorithms is significantly more efficient than the other. It is possible to determine the most performant algorithm by taking into consideration the ranking statistic used by Friedman's test, and observing which algorithm has the lowest average rank.

It is worth pointing out that this phase is indeed very similar to a single iteration of `irace`, where there are only seven configurations to test. In fact, one might apply a different approach, where the choice of the labeling algorithms to use in the BP framework is delegated to some additional parameters. This approach would require tuning a single BP algorithm, instead of seven, with which no benchmarking phase would be needed. Furthermore, this approach could result in a more effective BP algorithm that is able to switch between different labeling algorithms as needed, for example using one algorithm while evaluating the root node and switching to another afterwards. However, the increase in the number of parameters would also require an increased tunning budget to be able to obtain high quality configurations, which would make the tuning phase very hard to implement in practice given the hardware and time limitations. Moreover, we believe that

Table 6.3: VRPTW — Benchmarks — 6 hours.

| | | DSSR-L | | | | | NG-DSSR-L-E | | | | | NGRR | | | | | NG-DSSR-L | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Class | NI | S | PS | R | Time | Gap (%) | S | PS | R | Time | Gap (%) | S | PS | R | Time | Gap (%) | S | PS | R | Time | Gap (%) |
| C1–50 | 9 | 9 | | | 105.4 | | 9 | | | 184.9 | | 9 | | | 40.39 | | 9 | | | 55.56 | |
| R1–50 | 12 | 11 | 1 | | 193.8 | 4.04 | 11 | 1 | | 154.5 | 3.87 | 11 | 1 | | 495.1 | 4.95 | 11 | 1 | | 127.6 | 4.07 |
| RC1–50 | 8 | 2 | 6 | | 1682 | 12.22 | 2 | 6 | | 1723 | 10.97 | 2 | 6 | | 2478 | 12.01 | 2 | 6 | | 1683 | 11.12 |
| C1–75 | 9 | 8 | 1 | | 835.7 | 3.05 | 8 | 1 | | 610.6 | 3.45 | 8 | 1 | | 1545 | 3.39 | 8 | 1 | | 475.4 | 3.39 |
| R1–75 | 12 | 10 | 2 | | 3679 | 2.38 | 10 | 2 | | 2079 | 2.48 | 9 | 3 | | 2447 | 2.95 | 10 | 2 | | 3633 | 2.90 |
| RC1–75 | 8 | 1 | 5 | 2 | 924.5 | 6.36 | 2 | 6 | | 9868 | 6.79 | 1 | 6 | 0 | 580.7 | 6.47 | 1 | 7 | | 985.0 | 6.20 |
| C1–100 | 9 | 9 | | | 358.7 | | 9 | | | 288.0 | | 9 | | | 2063 | | 9 | | | 296.5 | |
| R1–100 | 12 | 6 | 4 | 2 | 4738 | 2.01 | 6 | 5 | 1 | 4108 | 2.40 | 4 | 8 | | 353.3 | 2.11 | 6 | 5 | 0 | 4096 | 1.95 |
| RC1–100 | 8 | | 5 | 3 | | 3.96 | | 5 | 3 | | 4.97 | | 5 | 2 | | 6.16 | | 6 | 1 | | 4.84 |

this methodology would offer us less information on the overall effectiveness of each labeling algorithm, which is what we are interested to evaluate in this study.

In the first round of benchmarking, we run each of the seven algorithms on Solomon's instances of Series 1 with 25 and 50 customers using the obtained configuration. A time limit of three hours is allotted for each algorithm. Note that these are similar conditions to the ones of the tuning phase.

At this point, if the tests indicate that there is a significant difference in algorithmic performance, we select the best algorithms and proceed with a second round of benchmarking. While the first round is carried out under conditions that are consistent with the tuning phase, here the idea is to study the algorithms on instances of larger size while providing a higher computing time limit. Thus, we run each of the best performing algorithms on Solomon's instances of Series 1 with 25, 50, 75, and 100 customers, with the same configurations used in the first round. The time limit is set to six hours instead of three. Furthermore, upper bound recovery as defined in Section 5.3.3 is attempted only at this round, when even the root node cannot be solved within the time limit. We finally proceed with the statistical analysis as previously done.

The benchmarking is performed on a computer with an Intel i7–3930K @ 3.20Ghz processor with 6 cores and 60GB of RAM, operating under Windows 10 Pro.

### 6.2.1 Benchmarking for the VRPTW

Tables 6.1 and 6.2 synthetically present the benchmark results of the first round. For each instance class, we report the number of instances in that class in column "NI", the number of instances solved to optimality in column "S", and the number of *partially solved* instances in column "PS". Note that the instances for which the algorithm is able to solve at least the root node

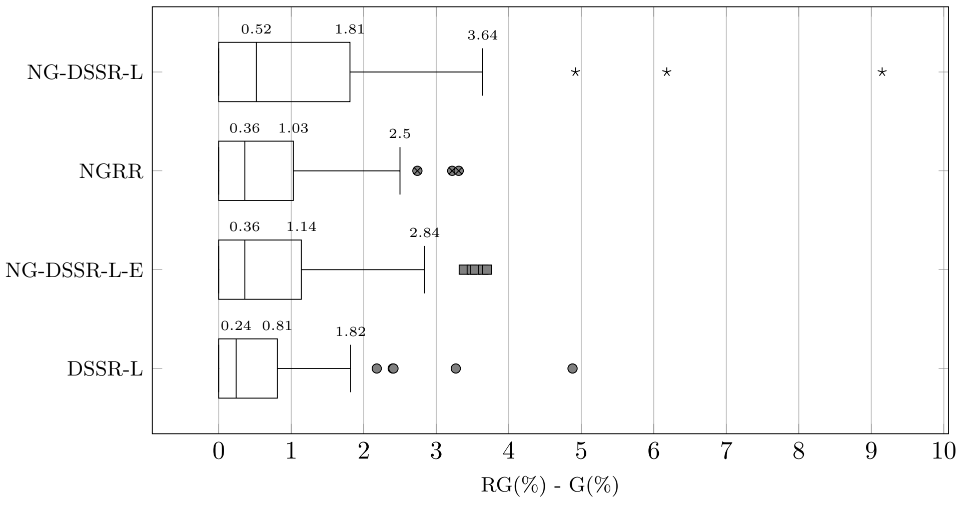


Figure 6.1: VRPTW — Differences between root gap and best gap. Median, upper quartile and upper whisker values are reported.

and obtain a feasible integer solution are called partially solved. As explained in Section 5.3.3, CPLEX is called once the root node is evaluated in order to obtain an integer solution. Furthermore, in this case, upper bound recovery does not count for considering an instance as partially solved. The column "Time" shows the average computing time calculated taking into account only the solved instances. Finally, the column "Gap (%)" gives the average gap calculated over the partially solved instances. For each instance, the gap is computed as $(UB - LB)/LB \times 100$, where $UB$ is the best upper bound obtained throughout the execution of the algorithm and $LB$ is the lower bound obtained at the root node.

Friedman's test yields a $p$-value significantly smaller than 0.05, i.e., $p < 2.2\text{e}{-16}$, thus rejecting the null hypothesis. We are now justified to proceed with Wilcoxon's test. According to its results, NG-DSSR-L and NG-DSSR-L-E outperform DSSR, NG-DSSR-G, and NG-DSSR-G-E with $p < 0.01$, while there is no statistical difference between the performances of NG-DSSR-L, NG-DSSR-L-E, DSSR-L, and NGRR. The lowest average ranking was achieved by DSSR-L, while NG-DSSR-G-E obtained the highest one. This suggests that the local approach performs significantly better than the global approach. A possible explanation for this is that the local approach allows to store more information about the cycles, since there is a set maintaining such information for each customer, instead of a single global set. Indeed,

Table 6.4: VRPTW — Best elementary route algorithms — 6 hours, 25 and 50 customers.

| | DSSR-L | | | | | | | NG-DSSR-L-E | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Time | UB | LB | RUB | G(%) | RG(%) | Nodes | Time | UB | LB | RUB | G(%) | RG(%) | Nodes |
| C101–25 | 0.26 | 191.3 | 191.3 | 191.3 | 0 | 0 | 1 | 0.53 | 191.3 | 191.3 | 191.3 | 0 | 0 | 1 |
| C102–25 | 1.18 | 190.3 | 190.3 | 190.3 | 0 | 0 | 1 | 1.5 | 190.3 | 190.3 | 190.3 | 0 | 0 | 1 |
| C103–25 | 4.05 | 190.3 | 190.3 | 190.3 | 0 | 0 | 1 | 7.7 | 190.3 | 190.3 | 190.3 | 0 | 0 | 1 |
| C104–25 | 37.69 | 186.9 | 186.9 | 186.9 | 0 | 0 | 1 | 76.4 | 186.9 | 186.9 | 186.9 | 0 | 0 | 1 |
| C105–25 | 0.44 | 191.3 | 191.3 | 191.3 | 0 | 0 | 1 | 0.55 | 191.3 | 191.3 | 191.3 | 0 | 0 | 1 |
| C106–25 | 0.41 | 191.3 | 191.3 | 191.3 | 0 | 0 | 1 | 0.52 | 191.3 | 191.3 | 191.3 | 0 | 0 | 1 |
| C107–25 | 0.49 | 191.3 | 191.3 | 191.3 | 0 | 0 | 1 | 0.51 | 191.3 | 191.3 | 191.3 | 0 | 0 | 1 |
| C108–25 | 1.04 | 191.3 | 191.3 | 191.3 | 0 | 0 | 1 | 0.81 | 191.3 | 191.3 | 191.3 | 0 | 0 | 1 |
| C109–25 | 1.84 | 191.3 | 191.3 | 191.3 | 0 | 0 | 1 | 1.77 | 191.3 | 191.3 | 191.3 | 0 | 0 | 1 |
| R101–25 | 0.17 | 617.1 | 617.1 | 617.1 | 0 | 0 | 1 | 0.28 | 617.1 | 617.1 | 617.1 | 0 | 0 | 1 |
| R102–25 | 0.54 | 547.1 | 546.3 | 547.1 | 0.14 | 0.14 | 5 | 0.56 | 547.1 | 546.3 | 547.1 | 0.14 | 0.14 | 3 |
| R103–25 | 0.47 | 454.6 | 454.6 | 454.6 | 0 | 0 | 1 | 0.51 | 454.6 | 454.6 | 454.6 | 0 | 0 | 1 |
| R104–25 | 0.45 | 416.9 | 416.9 | 416.9 | 0 | 0 | 1 | 0.62 | 416.9 | 416.9 | 416.9 | 0 | 0 | 1 |
| R105–25 | 0.25 | 530.5 | 530.5 | 530.5 | 0 | 0 | 1 | 0.31 | 530.5 | 530.5 | 530.5 | 0 | 0 | 1 |
| R106–25 | 0.52 | 465.4 | 457.3 | 465.4 | 1.77 | 1.77 | 3 | 1.24 | 465.4 | 457.3 | 468 | 1.77 | 2.34 | 7 |
| R107–25 | 0.62 | 424.3 | 424.3 | 424.3 | 0 | 0 | 1 | 0.62 | 424.3 | 424.3 | 424.3 | 0 | 0 | 1 |
| R108–25 | 2.21 | 397.3 | 396.8 | 397.3 | 0.12 | 0.12 | 5 | 1.16 | 397.3 | 396.8 | 397.3 | 0.12 | 0.12 | 5 |
| R109–25 | 0.38 | 441.3 | 441.3 | 441.3 | 0 | 0 | 1 | 0.47 | 441.3 | 441.3 | 441.3 | 0 | 0 | 1 |
| R110–25 | 1.92 | 444.1 | 438.3 | 444.7 | 1.31 | 1.45 | 19 | 2.95 | 444.1 | 438.3 | 444.1 | 1.31 | 1.31 | 25 |
| R111–25 | 1.45 | 428.8 | 427.3 | 430.1 | 0.36 | 0.66 | 9 | 1.14 | 428.8 | 427.3 | 428.8 | 0.36 | 0.36 | 5 |
| R112–25 | 5.71 | 393 | 387.1 | 393 | 1.54 | 1.54 | 19 | 4.58 | 393 | 387.1 | 393 | 1.54 | 1.54 | 19 |
| RC101–25 | 18.32 | 461.1 | 406.6 | 474.4 | 13.4 | 16.7 | 417 | 20.3 | 461.1 | 406.6 | 475.5 | 13.40 | 16.94 | 443 |
| RC102–25 | 1.57 | 351.8 | 351.8 | 351.8 | 0 | 0 | 1 | 1.03 | 351.8 | 351.8 | 351.8 | 0 | 0 | 1 |
| RC103–25 | 1.84 | 332.8 | 332.8 | 332.8 | 0 | 0 | 1 | 1.33 | 332.8 | 332.8 | 332.8 | 0 | 0 | 1 |
| RC104–25 | 2.31 | 306.6 | 306.6 | 306.6 | 0 | 0 | 1 | 2.27 | 306.6 | 306.6 | 306.6 | 0 | 0 | 1 |
| RC105–25 | 0.76 | 411.3 | 411.3 | 411.3 | 0 | 0 | 1 | 0.74 | 411.3 | 411.3 | 411.3 | 0 | 0 | 1 |
| RC106–25 | 0.83 | 345.5 | 345.5 | 345.5 | 0 | 0 | 1 | 0.93 | 345.5 | 345.5 | 345.5 | 0 | 0 | 1 |
| RC107–25 | 1.53 | 298.3 | 298.3 | 298.3 | 0 | 0 | 1 | 1.87 | 298.3 | 298.3 | 298.3 | 0 | 0 | 1 |
| RC108–25 | 3.22 | 294.5 | 294.5 | 294.5 | 0 | 0 | 1 | 7.22 | 294.5 | 294.5 | 294.5 | 0 | 0 | 1 |
| C101–50 | 0.67 | 362.4 | 362.4 | 362.4 | 0 | 0 | 1 | 0.93 | 362.4 | 362.4 | 362.4 | 0 | 0 | 1 |
| C102–50 | 2.78 | 361.4 | 361.4 | 361.4 | 0 | 0 | 1 | 3.16 | 361.4 | 361.4 | 361.4 | 0 | 0 | 1 |
| C103–50 | 9.1 | 361.4 | 361.4 | 361.4 | 0 | 0 | 1 | 22.61 | 361.4 | 361.4 | 361.4 | 0 | 0 | 1 |
| C104–50 | 920.4 | 358 | 358 | 358 | 0 | 0 | 1 | 1624 | 358 | 358 | 358 | 0 | 0 | 1 |
| C105–50 | 1.59 | 362.4 | 362.4 | 362.4 | 0 | 0 | 1 | 1.31 | 362.4 | 362.4 | 362.4 | 0 | 0 | 1 |
| C106–50 | 1.09 | 362.4 | 362.4 | 362.4 | 0 | 0 | 1 | 1.08 | 362.4 | 362.4 | 362.4 | 0 | 0 | 1 |
| C107–50 | 1.52 | 362.4 | 362.4 | 362.4 | 0 | 0 | 1 | 1.41 | 362.4 | 362.4 | 362.4 | 0 | 0 | 1 |
| C108–50 | 3.41 | 362.4 | 362.4 | 362.4 | 0 | 0 | 1 | 2.38 | 362.4 | 362.4 | 362.4 | 0 | 0 | 1 |
| C109–50 | 7.96 | 362.4 | 362.4 | 362.4 | 0 | 0 | 1 | 5.91 | 362.4 | 362.4 | 362.4 | 0 | 0 | 1 |
| R101–50 | 0.72 | 1044 | 1043.4 | 1044 | 0.06 | 0.06 | 3 | 0.72 | 1044 | 1043.3 | 1044 | 0.06 | 0.06 | 3 |
| R102–50 | 0.64 | 909 | 909 | 909 | 0 | 0 | 1 | 0.76 | 909 | 909 | 909 | 0 | 0 | 1 |
| R103–50 | 6.48 | 772.9 | 769.2 | 772.9 | 0.48 | 0.48 | 11 | 6.84 | 772.9 | 769.2 | 773.5 | 0.48 | 0.56 | 13 |
| R104–50 | 117.4 | 625.4 | 619.1 | 629 | 1.02 | 1.60 | 47 | 59.07 | 625.4 | 619.1 | 625.4 | 1.02 | 1.02 | 31 |
| R105–50 | 7.35 | 899.3 | 892.1 | 904.4 | 0.80 | 1.38 | 51 | 5.51 | 899.3 | 892.1 | 904.7 | 0.80 | 1.41 | 35 |
| R106–50 | 3.32 | 793 | 791.3 | 797.1 | 0.21 | 0.72 | 5 | 3.06 | 793 | 791.37 | 797.3 | 0.21 | 0.75 | 5 |
| R107–50 | 11.51 | 711.1 | 707.2 | 713 | 0.54 | 0.81 | 17 | 10.79 | 711.1 | 707.2 | 711.3 | 0.54 | 0.57 | 17 |
| R108–50 | 21600 | 618.7 | 594.7 | 625.4 | 4.04 | 5.16 | 11400 | 21600 | 617.7 | 594.7 | 622.2 | 3.87 | 4.62 | 14745 |
| R109–50 | 88.92 | 786.8 | 775.3 | 789.4 | 1.48 | 1.81 | 319 | 74.27 | 786.8 | 775.3 | 789.2 | 1.48 | 1.79 | 297 |
| R110–50 | 13.63 | 697 | 695.1 | 713.7 | 0.28 | 2.68 | 15 | 7.56 | 697 | 695.1 | 697 | 0.28 | 0.28 | 11 |
| R111–50 | 237.4 | 707.2 | 696.2 | 719.9 | 1.57 | 3.39 | 215 | 86.21 | 707.2 | 696.3 | 710 | 1.57 | 1.97 | 193 |
| R112–50 | 1644 | 630.2 | 614.8 | 639.3 | 2.50 | 3.98 | 1497 | 1444 | 630.2 | 614.8 | 636.9 | 2.50 | 3.59 | 1695 |
| RC101–50 | 21600 | 951.1 | 850.0 | 992.6 | 11.89 | 16.77 | 44301 | 21600 | 948 | 850 | 976.7 | 11.53 | 14.90 | 54123 |
| RC102–50 | 21600 | 830.7 | 721.8 | 830.7 | 15.09 | 15.09 | 19248 | 21600 | 822.5 | 721.8 | 828.2 | 13.95 | 14.74 | 27090 |
| RC103–50 | 21600 | 737.2 | 645.2 | 737.2 | 14.24 | 14.24 | 9492 | 21600 | 716.3 | 645.2 | 738.8 | 11.01 | 14.49 | 14162 |
| RC104–50 | 31.34 | 545.8 | 545.8 | 545.8 | 0 | 0 | 1 | 32.54 | 545.8 | 545.8 | 545.8 | 0 | 0 | 1 |
| RC105–50 | 21600 | 876.5 | 761.5 | 876.5 | 15.09 | 15.09 | 24927 | 21600 | 855.4 | 761.5 | 883.2 | 12.32 | 15.97 | 41880 |
| RC106–50 | 3332 | 723.2 | 664.4 | 724.8 | 8.85 | 9.09 | 4935 | 3413 | 723.2 | 664.4 | 729.6 | 8.85 | 9.81 | 4841 |
| RC107–50 | 21600 | 642.7 | 603.5 | 642.7 | 6.48 | 6.48 | 6674 | 21600 | 642.7 | 603.5 | 642.7 | 6.48 | 6.48 | 9087 |
| RC108–50 | 21600 | 598.1 | 541.1 | 598.1 | 10.52 | 10.52 | 2514 | 21600 | 598.1 | 541.1 | 598.1 | 10.52 | 10.52 | 4595 |

Table 6.5: VRPTW — Best *ng*-route algorithms — 6 hours, 25 and 50 customers.

| | NGRR | | | | | | | NG-DSSR-L | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Time | UB | LB | RUB | G(%) | RG(%) | Nodes | Time | UB | LB | RUB | G(%) | RG(%) | Nodes |
| C101–25 | 0.53 | 191.3 | 191.3 | 191.3 | 0 | 0 | 1 | 0.39 | 191.3 | 191.3 | 191.3 | 0 | 0 | 1 |
| C102–25 | 1.12 | 190.3 | 190.3 | 190.3 | 0 | 0 | 1 | 1.23 | 190.3 | 190.3 | 190.3 | 0 | 0 | 1 |
| C103–25 | 3.16 | 190.3 | 190.3 | 190.3 | 0 | 0 | 1 | 2.80 | 190.3 | 190.3 | 190.3 | 0 | 0 | 1 |
| C104–25 | 4.19 | 186.9 | 186.9 | 186.9 | 0 | 0 | 1 | 11.41 | 186.9 | 186.9 | 186.9 | 0 | 0 | 1 |
| C105–25 | 0.46 | 191.3 | 191.3 | 191.3 | 0 | 0 | 1 | 0.52 | 191.3 | 191.3 | 191.3 | 0 | 0 | 1 |
| C106–25 | 0.41 | 191.3 | 191.3 | 191.3 | 0 | 0 | 1 | 0.40 | 191.3 | 191.3 | 191.3 | 0 | 0 | 1 |
| C107–25 | 0.56 | 191.3 | 191.3 | 191.3 | 0 | 0 | 1 | 0.51 | 191.3 | 191.3 | 191.3 | 0 | 0 | 1 |
| C108–25 | 0.76 | 191.3 | 191.3 | 191.3 | 0 | 0 | 1 | 0.88 | 191.3 | 191.3 | 191.3 | 0 | 0 | 1 |
| C109–25 | 2.16 | 191.3 | 191.3 | 191.3 | 0 | 0 | 1 | 2.11 | 191.3 | 191.3 | 191.3 | 0 | 0 | 1 |
| R101–25 | 0.24 | 617.1 | 617.1 | 617.1 | 0 | 0 | 1 | 0.26 | 617.1 | 617.1 | 617.1 | 0 | 0 | 1 |
| R102–25 | 0.50 | 547.1 | 546.3 | 547.1 | 0.14 | 0.14 | 3 | 0.58 | 547.1 | 546.3 | 547.1 | 0.14 | 0.14 | 5 |
| R103–25 | 0.46 | 454.6 | 454.6 | 454.6 | 0 | 0 | 1 | 0.49 | 454.6 | 454.6 | 454.6 | 0 | 0 | 1 |
| R104–25 | 0.54 | 416.9 | 416.9 | 416.9 | 0 | 0 | 1 | 0.58 | 416.9 | 416.9 | 416.9 | 0 | 0 | 1 |
| R105–25 | 0.28 | 530.5 | 530.5 | 530.5 | 0 | 0 | 1 | 0.31 | 530.5 | 530.5 | 530.5 | 0 | 0 | 1 |
| R106–25 | 0.92 | 465.4 | 457.3 | 470.2 | 1.77 | 2.82 | 7 | 1.03 | 465.4 | 457.3 | 487.9 | 1.77 | 6.69 | 9 |
| R107–25 | 0.52 | 424.3 | 424.3 | 424.3 | 0 | 0 | 1 | 0.65 | 424.3 | 424.3 | 424.3 | 0 | 0 | 1 |
| R108–25 | 1.32 | 397.3 | 396.8 | 403.2 | 0.12 | 1.61 | 5 | 1.99 | 397.3 | 396.8 | 404.5 | 0.12 | 1.94 | 5 |
| R109–25 | 0.59 | 441.3 | 441.3 | 441.3 | 0 | 0 | 1 | 0.42 | 441.3 | 441.3 | 441.3 | 0 | 0 | 1 |
| R110–25 | 2.46 | 444.1 | 438.3 | 444.1 | 1.31 | 1.31 | 25 | 2.04 | 444.1 | 438.3 | 444.1 | 1.31 | 1.31 | 15 |
| R111–25 | 1.05 | 428.8 | 427.2 | 428.8 | 0.36 | 0.36 | 5 | 1.79 | 428.8 | 427.2 | 428.8 | 0.36 | 0.36 | 9 |
| R112–25 | 2.49 | 393 | 387.1 | 393 | 1.54 | 1.54 | 15 | 3.53 | 393 | 387.1 | 393 | 1.54 | 1.54 | 17 |
| RC101–25 | 15.07 | 461.1 | 406.6 | 474.2 | 13.40 | 16.62 | 321 | 18.23 | 461.1 | 406.6 | 498.3 | 13.40 | 22.55 | 417 |
| RC102–25 | 0.99 | 351.8 | 351.8 | 351.8 | 0 | 0 | 1 | 1.15 | 351.8 | 351.8 | 351.8 | 0 | 0 | 1 |
| RC103–25 | 1.60 | 332.8 | 332.8 | 332.8 | 0 | 0 | 1 | 1.42 | 332.8 | 332.8 | 332.8 | 0 | 0 | 1 |
| RC104–25 | 1.78 | 306.6 | 306.6 | 306.6 | 0 | 0 | 1 | 2.05 | 306.6 | 306.6 | 306.6 | 0 | 0 | 1 |
| RC105–25 | 0.92 | 411.3 | 410.95 | 412.5 | 0.09 | 0.38 | 3 | 0.84 | 411.3 | 411.3 | 411.3 | 0 | 0 | 1 |
| RC106–25 | 0.53 | 345.5 | 345.5 | 345.5 | 0 | 0 | 1 | 0.85 | 345.5 | 345.5 | 345.5 | 0 | 0 | 1 |
| RC107–25 | 1.83 | 298.3 | 298.3 | 298.3 | 0 | 0 | 1 | 2.06 | 298.3 | 298.3 | 298.3 | 0 | 0 | 1 |
| RC108–25 | 3.04 | 294.5 | 294.5 | 294.5 | 0 | 0 | 1 | 4.64 | 294.5 | 294.5 | 294.5 | 0 | 0 | 1 |
| C101–50 | 0.98 | 362.4 | 362.4 | 362.4 | 0 | 0 | 1 | 0.92 | 362.4 | 362.4 | 362.4 | 0 | 0 | 1 |
| C102–50 | 2.52 | 361.4 | 361.4 | 361.4 | 0 | 0 | 1 | 3.27 | 361.4 | 361.4 | 361.4 | 0 | 0 | 1 |
| C103–50 | 9.52 | 361.4 | 361.4 | 361.4 | 0 | 0 | 1 | 16.81 | 361.4 | 361.4 | 361.4 | 0 | 0 | 1 |
| C104–50 | 335.6 | 358 | 358 | 358 | 0 | 0 | 1 | 467.04 | 358 | 358 | 358 | 0 | 0 | 1 |
| C105–50 | 1.28 | 362.4 | 362.4 | 362.4 | 0 | 0 | 1 | 1.30 | 362.4 | 362.4 | 362.4 | 0 | 0 | 1 |
| C106–50 | 1.41 | 362.4 | 362.4 | 362.4 | 0 | 0 | 1 | 1.11 | 362.4 | 362.4 | 362.4 | 0 | 0 | 1 |
| C107–50 | 2.79 | 362.4 | 362.4 | 362.4 | 0 | 0 | 1 | 1.64 | 362.4 | 362.4 | 362.4 | 0 | 0 | 1 |
| C108–50 | 2.35 | 362.4 | 362.4 | 362.4 | 0 | 0 | 1 | 2.57 | 362.4 | 362.4 | 362.4 | 0 | 0 | 1 |
| C109–50 | 6.95 | 362.4 | 362.4 | 362.4 | 0 | 0 | 1 | 5.36 | 362.4 | 362.4 | 362.4 | 0 | 0 | 1 |
| R101–50 | 0.81 | 1044 | 1043.3 | 1046.7 | 0.06 | 0.32 | 3 | 0.72 | 1044 | 1043.3 | 1044 | 0.06 | 0.06 | 3 |
| R102–50 | 0.71 | 909 | 909 | 909 | 0 | 0 | 1 | 0.75 | 909 | 909 | 909 | 0 | 0 | 1 |
| R103–50 | 6.57 | 772.9 | 769.2 | 776.6 | 0.48 | 0.96 | 15 | 5.69 | 772.9 | 769.23 | 774.4 | 0.48 | 0.67 | 11 |
| R104–50 | 300.7 | 625.4 | 619.1 | 629.4 | 1.02 | 1.67 | 211 | 69.08 | 625.4 | 619.1 | 633.3 | 1.02 | 2.30 | 33 |
| R105–50 | 4.05 | 899.3 | 892.1 | 903.7 | 0.80 | 1.30 | 25 | 7.52 | 899.3 | 892.1 | 904.7 | 0.80 | 1.41 | 49 |
| R106–50 | 2.05 | 793 | 791.3 | 797.3 | 0.21 | 0.75 | 3 | 2.56 | 793 | 791.3 | 797.1 | 0.21 | 0.72 | 5 |
| R107–50 | 15.89 | 711.1 | 707.2 | 716.8 | 0.54 | 1.35 | 17 | 11.68 | 711.1 | 707.2 | 711.1 | 0.54 | 0.54 | 15 |
| R108–50 | 21600 | 624.1 | 594.6 | 628.9 | 4.95 | 5.76 | 13099 | 21600 | 618.9 | 594.7 | 625.5 | 4.07 | 5.18 | 13653 |
| R109–50 | 116.3 | 786.8 | 775.3 | 791.9 | 1.48 | 2.14 | 501 | 57.27 | 786.8 | 775.3 | 787.6 | 1.48 | 1.58 | 261 |
| R110–50 | 14.65 | 697 | 695.1 | 697 | 0.28 | 0.28 | 13 | 13.68 | 697 | 695.1 | 711.6 | 0.28 | 2.38 | 23 |
| R111–50 | 2018 | 707.2 | 696.3 | 714.4 | 1.57 | 2.60 | 1885 | 156.6 | 707.2 | 696.3 | 711.4 | 1.57 | 2.17 | 237 |
| R112–50 | 2965 | 630.2 | 614.6 | 635 | 2.53 | 3.32 | 2265 | 1078 | 630.2 | 614.8 | 635 | 2.50 | 3.28 | 1439 |
| RC101–50 | 21600 | 971.1 | 850 | 990.1 | 14.24 | 16.48 | 100773 | 21600 | 946.9 | 850 | 999.4 | 11.40 | 17.57 | 52064 |
| RC102–50 | 21600 | 830.4 | 721.8 | 840.1 | 15.04 | 16.39 | 22475 | 21600 | 827.3 | 721.8 | 844.3 | 14.61 | 16.97 | 24779 |
| RC103–50 | 21600 | 727.9 | 645.2 | 737 | 12.80 | 14.21 | 23251 | 21600 | 717.5 | 645.2 | 717.5 | 11.19 | 11.19 | 10333 |
| RC104–50 | 18.55 | 545.8 | 545.8 | 545.8 | 0 | 0 | 1 | 46.74 | 545.8 | 545.8 | 545.8 | 0 | 0 | 1 |
| RC105–50 | 21600 | 860.4 | 761.5 | 885.6 | 12.98 | 16.29 | 52519 | 21600 | 856.9 | 761.5 | 878.2 | 12.52 | 15.32 | 31866 |
| RC106–50 | 4937 | 723.2 | 664.4 | 741.4 | 8.85 | 11.58 | 5833 | 3319 | 723.2 | 664.4 | 734.1 | 8.85 | 10.49 | 4865 |
| RC107–50 | 21600 | 642.7 | 603.5 | 642.7 | 6.48 | 6.48 | 6750 | 21600 | 642.7 | 603.5 | 642.7 | 6.48 | 6.48 | 8364 |
| RC108–50 | 21600 | 598.1 | 541.1 | 598.1 | 10.52 | 10.52 | 3100 | 21600 | 598.1 | 541.1 | 598.1 | 10.52 | 10.52 | 3868 |

Table 6.6: VRPTW — Best elementary route algorithms — 6 hours, 75 and 100 customers.

| | DSSR-L | | | | | | | NG-DSSR-L-E | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Time | UB | LB | RUB | G(%) | RG(%) | Nodes | Time | UB | LB | RUB | G(%) | RG(%) | Nodes |
| C101–75 | 1.58 | 649.2 | 649.2 | 649.2 | 0.00 | 0.00 | 1 | 1.99 | 649.2 | 649.2 | 649.2 | 0.00 | 0.00 | 1 |
| C102–75 | 7.61 | 649.2 | 649.2 | 649.2 | 0.00 | 0.00 | 1 | 6.38 | 649.2 | 649.2 | 649.2 | 0.00 | 0.00 | 1 |
| C103–75 | 39.59 | 649 | 649 | 649 | 0.00 | 0.00 | 1 | 33.47 | 649 | 649 | 649 | 0.00 | 0.00 | 1 |
| C104–75 | 21600 | 642.6 | 623.58 | 642.6 | 3.05 | 3.05 | 109 | 21600 | 645.1 | 623.58 | 645.1 | 3.45 | 3.45 | 355 |
| C105–75 | 2.34 | 649.2 | 649.2 | 649.2 | 0.00 | 0.00 | 1 | 2.58 | 649.2 | 649.2 | 649.2 | 0.00 | 0.00 | 1 |
| C106–75 | 6.53 | 647.3 | 632.35 | 648.8 | 2.36 | 2.60 | 3 | 5.9 | 647.3 | 632.35 | 649.2 | 2.36 | 2.66 | 3 |
| C107–75 | 3.5 | 647.3 | 647.3 | 647.3 | 0.00 | 0.00 | 1 | 2.85 | 647.3 | 647.3 | 647.3 | 0.00 | 0.00 | 1 |
| C108–75 | 41.29 | 645.5 | 631.15 | 647.8 | 2.27 | 2.64 | 9 | 19.24 | 645.5 | 631.15 | 645.5 | 2.27 | 2.27 | 7 |
| C109–75 | 6582 | 639.2 | 625 | 639.2 | 2.27 | 2.27 | 647 | 4812 | 639.2 | 625 | 639.6 | 2.27 | 2.34 | 829 |
| R101–75 | 0.7 | 1425.1 | 1425.1 | 1425.1 | 0.00 | 0.00 | 1 | 0.83 | 1425.1 | 1425.1 | 1425.1 | 0.00 | 0.00 | 1 |
| R102–75 | 1.82 | 1268.5 | 1268.5 | 1268.5 | 0.00 | 0.00 | 1 | 1.41 | 1268.5 | 1268.5 | 1268.5 | 0.00 | 0.00 | 1 |
| R103–75 | 4.12 | 1013.2 | 1013.2 | 1013.2 | 0.00 | 0.00 | 1 | 3.55 | 1013.2 | 1013.2 | 1013.2 | 0.00 | 0.00 | 1 |
| R104–75 | 14425 | 807.2 | 796.59 | 818 | 1.33 | 2.69 | 1509 | 6948 | 807.2 | 796.59 | 836.7 | 1.33 | 5.04 | 855 |
| R105–75 | 1.51 | 1161.6 | 1161.6 | 1161.6 | 0.00 | 0.00 | 1 | 1.34 | 1161.6 | 1161.6 | 1161.6 | 0.00 | 0.00 | 1 |
| R106–75 | 113.4 | 1075.6 | 1068.79 | 1075.6 | 0.64 | 0.64 | 85 | 58.81 | 1075.6 | 1068.79 | 1075.6 | 0.64 | 0.64 | 93 |
| R107–75 | 564.3 | 919.1 | 912.23 | 923.2 | 0.75 | 1.20 | 147 | 187.9 | 919.1 | 912.23 | 923.5 | 0.75 | 1.24 | 119 |
| R108–75 | 8351 | 786.5 | 775.8 | 798.1 | 1.38 | 2.87 | 1187 | 2624 | 786.5 | 775.8 | 805.7 | 1.38 | 3.85 | 561 |
| R109–75 | 21600 | 1020.8 | 994.17 | 1044.8 | 2.68 | 5.09 | 11355 | 21600 | 1020.8 | 994.17 | 1044.2 | 2.68 | 5.03 | 14970 |
| R110–75 | 21600 | 923.1 | 904.34 | 930.9 | 2.07 | 2.94 | 9559 | 21600 | 924.9 | 904.34 | 942.4 | 2.27 | 4.21 | 11680 |
| R111–75 | 2753 | 896.3 | 884.78 | 907.9 | 1.30 | 2.61 | 785 | 2743 | 896.3 | 884.78 | 910.4 | 1.30 | 2.90 | 1673 |
| R112–75 | 10577 | 815 | 803.75 | 821.5 | 1.40 | 2.21 | 2627 | 8220 | 815 | 803.75 | 828.6 | 1.40 | 3.09 | 3541 |
| RC101–75 | 924.5 | 1361.3 | 1335.9 | 1367.8 | 1.90 | 2.39 | 1963 | 1231 | 1361.3 | 1335.9 | 1391.8 | 1.90 | 4.18 | 2895 |
| RC102–75 | 21600 | 1270.6 | 1197.04 | 1270.6 | 6.15 | 6.15 | 17652 | 21600 | 1249 | 1197.04 | 1267.8 | 4.34 | 5.91 | 18298 |
| RC103–75 | 21600 | 1085.9 | 1019.94 | 1085.9 | 6.47 | 6.47 | 4344 | 21600 | 1083.5 | 1019.94 | 1083.5 | 6.23 | 6.23 | 9216 |
| RC104–75 | 21600 | 973.6† | | | | | | 21600 | 978.1 | 919.35 | 978.1 | 6.39 | 6.39 | 601 |
| RC105–75 | 21600 | 1279 | 1194.08 | 1279 | 7.11 | 7.11 | 20646 | 21600 | 1244.5 | 1194.08 | 1278.4 | 4.22 | 7.06 | 17184 |
| RC106–75 | 21600 | 1141.4 | 1078.62 | 1141.4 | 5.82 | 5.82 | 13312 | 18505 | 1128.3 | 1078.62 | 1140.4 | 4.61 | 5.73 | 13509 |
| RC107–75 | 21600 | 1027.8 | 967.19 | 1027.8 | 6.27 | 6.27 | 2574 | 21600 | 1055.1 | 967.19 | 1055.1 | 9.09 | 9.09 | 1475 |
| RC108–75 | 21600 | 948.9† | | | | | | 21600 | 970.9 | 878.97 | 970.9 | 10.46 | 10.46 | 1704 |
| C101–100 | 3.34 | 827.3 | 827.3 | 827.3 | 0.00 | 0.00 | 1 | 2.88 | 827.3 | 827.3 | 827.3 | 0.00 | 0.00 | 1 |
| C102–100 | 23.8 | 827.3 | 827.3 | 827.3 | 0.00 | 0.00 | 1 | 56.89 | 827.3 | 827.3 | 827.3 | 0.00 | 0.00 | 1 |
| C103–100 | 95.39 | 826.3 | 826.3 | 826.3 | 0.00 | 0.00 | 1 | 132.9 | 826.3 | 826.3 | 826.3 | 0.00 | 0.00 | 1 |
| C104–100 | 2918 | 822.9 | 822.9 | 822.9 | 0.00 | 0.00 | 1 | 2338 | 822.9 | 822.9 | 822.9 | 0.00 | 0.00 | 1 |
| C105–100 | 4.93 | 827.3 | 827.3 | 827.3 | 0.00 | 0.00 | 1 | 5.39 | 827.3 | 827.3 | 827.3 | 0.00 | 0.00 | 1 |
| C106–100 | 9.95 | 827.3 | 827.3 | 827.3 | 0.00 | 0.00 | 1 | 7.14 | 827.3 | 827.3 | 827.3 | 0.00 | 0.00 | 1 |
| C107–100 | 7.61 | 827.3 | 827.3 | 827.3 | 0.00 | 0.00 | 1 | 6.27 | 827.3 | 827.3 | 827.3 | 0.00 | 0.00 | 1 |
| C108–100 | 50.03 | 827.3 | 827.3 | 827.3 | 0.00 | 0.00 | 1 | 10.58 | 827.3 | 827.3 | 827.3 | 0.00 | 0.00 | 1 |
| C109–100 | 115.2 | 827.3 | 827.3 | 827.3 | 0.00 | 0.00 | 1 | 30.9 | 827.3 | 827.3 | 827.3 | 0.00 | 0.00 | 1 |
| R101–100 | 13.7 | 1637.7 | 1631.15 | 1639.6 | 0.40 | 0.52 | 15 | 12.74 | 1637.7 | 1631.15 | 1638.2 | 0.40 | 0.43 | 15 |
| R102–100 | 6.02 | 1466.6 | 1466.6 | 1466.6 | 0.00 | 0.00 | 1 | 3.73 | 1466.6 | 1466.6 | 1466.6 | 0.00 | 0.00 | 1 |
| R103–100 | 165.5 | 1208.7 | 1206.78 | 1218.4 | 0.16 | 0.96 | 25 | 559.3 | 1208.7 | 1206.78 | 1214.9 | 0.16 | 0.67 | 199 |
| R104–100 | 21600 | 980.3 | 956.97 | 980.3 | 2.44 | 2.44 | 745 | 21600 | 984.8 | 956.97 | 984.8 | 2.91 | 2.91 | 1726 |
| R105–100 | 384.4 | 1355.3 | 1346.14 | 1363.3 | 0.68 | 1.27 | 325 | 1232 | 1355.3 | 1346.14 | 1367.9 | 0.68 | 1.62 | 1121 |
| R106–100 | 12311 | 1234.6 | 1226.91 | 1241.1 | 0.63 | 1.16 | 1313 | 7944 | 1234.6 | 1226.91 | 1249.1 | 0.63 | 1.81 | 3533 |
| R107–100 | 21600 | 1080.6 | 1053.26 | 1080.6 | 2.60 | 2.60 | 631 | 21600 | 1064.6 | 1053.26 | 1070 | 1.08 | 1.59 | 4557 |
| R108–100 | 21600 | 962.7† | | | | | | 21600 | 963.3† | | | | | |
| R109–100 | 15549 | 1146.9 | 1134.28 | 1158.6 | 1.11 | 2.14 | 3241 | 14897 | 1146.9 | 1134.28 | 1155.9 | 1.11 | 1.91 | 4763 |
| R110–100 | 21600 | 1070.9 | 1055.88 | 1070.9 | 1.42 | 1.42 | 2302 | 21600 | 1086.1 | 1055.88 | 1086.1 | 2.86 | 2.86 | 4302 |
| R111–100 | 21600 | 1051 | 1034.73 | 1051 | 1.57 | 1.57 | 1509 | 21600 | 1050.9 | 1034.73 | 1058.5 | 1.56 | 2.30 | 4320 |
| R112–100 | 21600 | 967.1† | | | | | | 21600 | 959.8 | 926.72 | 959.8 | 3.57 | 3.57 | 1446 |
| RC101–100 | 21600 | 1620.6 | 1584.09 | 1655.1 | 2.30 | 4.48 | 14633 | 21600 | 1657.5 | 1584.09 | 1657.7 | 4.63 | 4.65 | 14600 |
| RC102–100 | 21600 | 1507.2† | | | | | | 21600 | 1506.2 | 1406.44 | 1506.2 | 7.09 | 7.09 | 6918 |
| RC103–100 | 21600 | 1278.4 | 1225.65 | 1278.4 | 4.30 | 4.30 | 1136 | 21600 | 1302.9† | | | | | |
| RC104–100 | 21600 | 1174.2 | 1101.81 | 1174.2 | 6.57 | 6.57 | 96 | 21600 | 1162.7† | | | | | |
| RC105–100 | 21600 | 1536.7 | 1471.94 | 1548.8 | 4.40 | 5.22 | 7513 | 21600 | 1534.6 | 1471.94 | 1556.1 | 4.26 | 5.72 | 2611 |
| RC106–100 | 21600 | 1390.1† | | | | | | 21600 | 1395.1 | 1318.8 | 1395.1 | 5.79 | 5.79 | 4415 |
| RC107–100 | 21600 | 1209.8 | 1183.37 | 1209.8 | 2.23 | 2.23 | 1480 | 21600 | 1220 | 1183.37 | 1220 | 3.10 | 3.10 | 3218 |
| RC108–100 | 21600 | 1154.4† | | | | | | 21600 | 1208.1† | | | | | |

† Recovered upper bound.

Table 6.7: VRPTW — Best *ng*-route algorithms — 6 hours, 75 and 100 customers.

| | NGRR | | | | | | | NG-DSSR-L | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Time | UB | LB | RUB | G(%) | RG(%) | Nodes | Time | UB | LB | RUB | G(%) | RG(%) | Nodes |
| C101–75 | 1.75 | 649.2 | 649.2 | 649.2 | 0.00 | 0.00 | 1 | 1.88 | 649.2 | 649.2 | 649.2 | 0.00 | 0.00 | 1 |
| C102–75 | 8.33 | 649.2 | 649.2 | 649.2 | 0.00 | 0.00 | 1 | 5.54 | 649.2 | 649.2 | 649.2 | 0.00 | 0.00 | 1 |
| C103–75 | 38.82 | 649 | 649 | 649 | 0.00 | 0.00 | 1 | 43.95 | 649 | 649 | 649 | 0.00 | 0.00 | 1 |
| C104–75 | 21600 | 644.3 | 623.1 | 644.3 | 3.39 | 3.39 | 14 | 21600 | 644.3 | 623.1 | 644.3 | 3.39 | 3.39 | 137 |
| C105–75 | 3.38 | 649.2 | 649.2 | 649.2 | 0.00 | 0.00 | 1 | 2.63 | 649.2 | 649.2 | 649.2 | 0.00 | 0.00 | 1 |
| C106–75 | 7.31 | 647.3 | 632.3 | 648.9 | 2.36 | 2.62 | 3 | 6.32 | 647.3 | 632.3 | 648.9 | 2.36 | 2.62 | 3 |
| C107–75 | 3.49 | 647.3 | 647.3 | 647.3 | 0.00 | 0.00 | 1 | 2.73 | 647.3 | 647.3 | 647.3 | 0.00 | 0.00 | 1 |
| C108–75 | 28.39 | 645.5 | 631.1 | 648.1 | 2.27 | 2.69 | 7 | 12.43 | 645.5 | 631.1 | 648.9 | 2.27 | 2.81 | 3 |
| C109–75 | 12268 | 639.2 | 625 | 649.4 | 2.27 | 3.90 | 1187 | 3727 | 639.2 | 625 | 639.6 | 2.27 | 2.34 | 613 |
| R101–75 | 0.7 | 1425.1 | 1425.1 | 1425.1 | 0.00 | 0.00 | 1 | 0.82 | 1425.1 | 1425.1 | 1425.1 | 0.00 | 0.00 | 1 |
| R102–75 | 1.18 | 1268.5 | 1268.5 | 1268.5 | 0.00 | 0.00 | 1 | 1.75 | 1268.5 | 1268.5 | 1268.5 | 0.00 | 0.00 | 1 |
| R103–75 | 3.28 | 1013.2 | 1013.2 | 1013.2 | 0.00 | 0.00 | 1 | 3.48 | 1013.2 | 1013.2 | 1013.2 | 0.00 | 0.00 | 1 |
| R104–75 | 9704 | 807.2 | 796.5 | 821.4 | 1.34 | 3.12 | 2400 | 21600 | 820.5 | 796.6 | 820.5 | 3.00 | 3.00 | 3867 |
| R105–75 | 1.17 | 1161.6 | 1161.6 | 1161.6 | 0.00 | 0.00 | 1 | 1.32 | 1161.6 | 1161.6 | 1161.6 | 0.00 | 0.00 | 1 |
| R106–75 | 64.95 | 1075.6 | 1068.8 | 1075.6 | 0.64 | 0.64 | 79 | 57.05 | 1075.6 | 1068.8 | 1075.9 | 0.64 | 0.67 | 95 |
| R107–75 | 1209 | 919.1 | 912.04 | 925.4 | 0.77 | 1.46 | 421 | 440.5 | 919.1 | 912.2 | 923.5 | 0.75 | 1.24 | 213 |
| R108–75 | 21600 | 786.5 | 775.8 | 805.9 | 1.38 | 3.88 | 2589 | 3027 | 786.5 | 775.8 | 802.5 | 1.38 | 3.44 | 663 |
| R109–75 | 21600 | 1036.7 | 994.1 | 1036.7 | 4.28 | 4.28 | 15339 | 21600 | 1022 | 994.1 | 1043.3 | 2.80 | 4.94 | 13733 |
| R110–75 | 21600 | 933.1 | 904.3 | 933.1 | 3.18 | 3.18 | 10322 | 21044 | 923.1 | 904.3 | 929.4 | 2.07 | 2.77 | 12473 |
| R111–75 | 4556 | 896.3 | 884.8 | 900 | 1.30 | 1.72 | 1977 | 3320 | 896.3 | 884.8 | 905.9 | 1.30 | 2.39 | 1697 |
| R112–75 | 6485 | 815 | 803.7 | 817.1 | 1.40 | 1.66 | 1207 | 8440 | 815 | 803.7 | 839 | 1.40 | 4.39 | 3389 |
| RC101–75 | 580.6 | 1361.3 | 1335.9 | 1362.9 | 1.90 | 2.02 | 1381 | 985 | 1361.3 | 1335.9 | 1389.3 | 1.90 | 4.00 | 2063 |
| RC102–75 | 21600 | 1264.5 | 1197 | 1265.1 | 5.64 | 5.69 | 28053 | 21600 | 1256.6 | 1197 | 1286.2 | 4.98 | 7.45 | 18014 |
| RC103–75 | 21600 | 1098.1 | 1019.9 | 1098.1 | 7.66 | 7.66 | 22629 | 21600 | 1117.5 | 1019.9 | 1117.5 | 9.57 | 9.57 | 6318 |
| RC104–75 | 21600 | 999.3 | 919.2 | 999.3 | 8.71 | 8.71 | 10410 | 21600 | 961.7 | 919.3 | 961.7 | 4.61 | 4.61 | 876 |
| RC105–75 | 21600 | 1272.5 | 1194.1 | 1280.9 | 6.57 | 7.27 | 30565 | 21600 | 1243.4 | 1194.1 | 1269.2 | 4.13 | 6.29 | 17715 |
| RC106–75 | 21600 | 1135.1 | 1078.6 | 1137.8 | 5.24 | 5.49 | 17295 | 21600 | 1124 | 1078.6 | 1163.3 | 4.21 | 7.85 | 14606 |
| RC107–75 | 21600 | 1015.6 | 967.2 | 1038.9 | 5.01 | 7.41 | 8769 | 21600 | 1017.7 | 967.2 | 1017.7 | 5.22 | 5.22 | 7742 |
| RC108–75 | 21600 | | | | | | | 21600 | 973.2 | 878.9 | 973.2 | 10.72 | 10.72 | 913 |
| C101–100 | 3.06 | 827.3 | 827.3 | 827.3 | 0.00 | 0.00 | 1 | 2.94 | 827.3 | 827.3 | 827.3 | 0.00 | 0.00 | 1 |
| C102–100 | 42.35 | 827.3 | 827.3 | 827.3 | 0.00 | 0.00 | 1 | 19.75 | 827.3 | 827.3 | 827.3 | 0.00 | 0.00 | 1 |
| C103–100 | 467.4 | 826.3 | 826.3 | 826.3 | 0.00 | 0.00 | 1 | 239 | 826.3 | 826.3 | 826.3 | 0.00 | 0.00 | 1 |
| C104–100 | 17967 | 822.9 | 822.9 | 822.9 | 0.00 | 0.00 | 1 | 2343 | 822.9 | 822.9 | 822.9 | 0.00 | 0.00 | 1 |
| C105–100 | 5.64 | 827.3 | 827.3 | 827.3 | 0.00 | 0.00 | 1 | 5.31 | 827.3 | 827.3 | 827.3 | 0.00 | 0.00 | 1 |
| C106–100 | 8.85 | 827.3 | 827.3 | 827.3 | 0.00 | 0.00 | 1 | 7.55 | 827.3 | 827.3 | 827.3 | 0.00 | 0.00 | 1 |
| C107–100 | 7.93 | 827.3 | 827.3 | 827.3 | 0.00 | 0.00 | 1 | 6.46 | 827.3 | 827.3 | 827.3 | 0.00 | 0.00 | 1 |
| C108–100 | 15.03 | 827.3 | 827.3 | 827.3 | 0.00 | 0.00 | 1 | 9.13 | 827.3 | 827.3 | 827.3 | 0.00 | 0.00 | 1 |
| C109–100 | 47.68 | 827.3 | 827.3 | 827.3 | 0.00 | 0.00 | 1 | 34.58 | 827.3 | 827.3 | 827.3 | 0.00 | 0.00 | 1 |
| R101–100 | 22.21 | 1637.7 | 1631.1 | 1644.2 | 0.40 | 0.80 | 33 | 16.07 | 1637.7 | 1631.1 | 1647.5 | 0.40 | 1.00 | 19 |
| R102–100 | 3.83 | 1466.6 | 1466.6 | 1466.6 | 0.00 | 0.00 | 1 | 3.76 | 1466.6 | 1466.6 | 1466.6 | 0.00 | 0.00 | 1 |
| R103–100 | 286.8 | 1208.7 | 1206.7 | 1210.6 | 0.16 | 0.32 | 57 | 550.4 | 1208.7 | 1206.7 | 1213.7 | 0.16 | 0.57 | 151 |
| R104–100 | 21600 | 982.4 | 956.8 | 984.8 | 2.67 | 2.92 | 1092 | 21600 | 977.4 | 956.9 | 977.4 | 2.13 | 2.13 | 1975 |
| R105–100 | 1100 | 1355.3 | 1346.1 | 1371.4 | 0.68 | 1.88 | 1029 | 305.1 | 1355.3 | 1346.1 | 1366.9 | 0.68 | 1.54 | 313 |
| R106–100 | 21600 | 1240.3 | 1226.9 | 1254 | 1.09 | 2.21 | 8677 | 6166 | 1234.6 | 1226.9 | 1248 | 0.63 | 1.72 | 2613 |
| R107–100 | 21600 | 1074.1 | 1053.2 | 1074.1 | 1.98 | 1.98 | 1820 | 21600 | 1064.6 | 1053.2 | 1067.7 | 1.08 | 1.37 | 3350 |
| R108–100 | 21600 | 936.6 | 913.4 | 936.6 | 2.54 | 2.54 | 251 | 21600 | | | | | | |
| R109–100 | 21600 | 1146.9 | 1134.3 | 1153.4 | 1.11 | 1.69 | 4127 | 17537 | 1146.9 | 1134.3 | 1156.4 | 1.11 | 1.95 | 5333 |
| R110–100 | 21600 | 1072.5 | 1055.8 | 1072.5 | 1.57 | 1.57 | 2026 | 21600 | 1069.9 | 1055.9 | 1075.4 | 1.33 | 1.85 | 4488 |
| R111–100 | 21600 | 1054.8 | 1034.7 | 1054.8 | 1.94 | 1.94 | 3185 | 21600 | 1051 | 1034.7 | 1061.5 | 1.57 | 2.59 | 3358 |
| R112–100 | 21600 | 963.6 | 926.7 | 963.6 | 3.98 | 3.98 | 383 | 21600 | 960.5 | 926.7 | 960.5 | 3.65 | 3.65 | 871 |
| RC101–100 | 21600 | 1662.4 | 1584.1 | 1665.9 | 4.94 | 5.16 | 17194 | 21600 | 1621.2 | 1584.1 | 1658.4 | 2.34 | 4.69 | 13707 |
| RC102–100 | 21600 | 1499.2 | 1406.4 | 1504.3 | 6.60 | 6.96 | 11213 | 21600 | 1526.2 | 1406.4 | 1526.2 | 8.52 | 8.52 | 5535 |
| RC103–100 | 21600 | 1306.2 | 1225.6 | 1306.2 | 6.57 | 6.57 | 7086 | 21600 | | | | | | |
| RC104–100 | 21600 | 1167.1$^{\dagger}$ | | | | | | 21600 | 1163.7 | 1101.8 | 1163.7 | 5.62 | 5.62 | 146 |
| RC105–100 | 21600 | 1556.3 | 1471.9 | 1573.8 | 5.73 | 6.92 | 12753 | 21600 | 1516.9 | 1471.9 | 1563.7 | 3.05 | 6.23 | 6641 |
| RC106–100 | 21600 | | | | | | | 21600 | 1396.7 | 1318.8 | 1396.7 | 5.91 | 5.91 | 6418 |
| RC107–100 | 21600 | 1244.6 | 1183.3 | 1244.6 | 5.17 | 5.17 | 4022 | 21600 | 1225.8 | 1183.4 | 1225.8 | 3.59 | 3.59 | 3342 |
| RC108–100 | 21600 | 1151.6$^{\dagger}$ | | | | | | 21600 | 1211.4$^{\dagger}$ | | | | | |

$^{\dagger}$ Recovered upper bound.

while a certain cycle with endpoints on customer $i$ can be likely to occur, it might be unnecessary to mark $i$ as critical for customers that are outside the cycle. This granularity also allows for an update rule that performs the least necessary work in order to prevent the further generation of encountered cycles, while maintaining the number of "activated" elementarity resources low in order to speed up the label extension phase.

Next, we run the four more performing algorithms, namely DSSR-L, NG-DSSR-L, NG-DSSR-L-E, and NGRR, on Solomon's instances of Series 1 with 25, 50, 75, and 100 customers, with a time limit of six hours. Table 6.3 summarizes these results using the same format of Tables 6.1 and 6.2, although since the four algorithms are able to solve all the instances with 25 customers within three hours, these are omitted. Furthermore, column "R" in Table 6.3 reports where appropriate the number of instances for which CPLEX is able to recover an upper bound if the algorithm fails to solve the root node within the time limit of six hours.

Tables 6.4, 6.5, 6.6, and 6.7 report the detailed results of these experiments. For each instance considered, we give the corresponding computing time in seconds (column "Time"), the best upper bound obtained (column "UB"), the lower bound obtained at the root node (column "LB"), the upper bound provided by CPLEX at the root node (column "RUB"), the integrality gap computed using the best available upper bound (column "G (%)"), the gap computed using the upper bound obtained at the root node (column "RG (%)"), and the total number of nodes in the branch-and-bound tree that have been explored by the algorithm (column "Nodes"). Moreover, the values marked with (†) in the "UB" column indicate that the upper bound is recovered by CPLEX. The time limit for CPLEX in this case is set to three hours. Note that if only the trivial columns are available in the pool, CPLEX is not called.

At this stage, Friedman's test does not indicate any significant difference between the four algorithms ($p = 0.11$). While the test is inconclusive, Table 6.3 and the detailed tables suggest that NGRR is generally able to solve less instances than the other algorithms, and when it is able to solve the same number of instances, it often achieves worse computing times on average. This is also supported by the fact that NGRR has the highest ranking statistic among all four algorithms (the lowest one is achieved by NG-DSSR-L). As we are going to see in the next section, this result is similar to what we have observed for the VRPTWWTC.

Finally, Figure 6.1 depicts the distribution of the differences between the gap computed at the root node and the gap computed using the best available upper bound, calculated only for solved or partially solved instances such

Table 6.8: VRPTWWTC — Elementary algorithms — 3 hours.

| | | DSSR | | | | DSSR-L | | | | NG-DSSR-G-E | | | | NG-DSSR-L-E | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Class | NI | S | PS | Time | Gap (%) | S | PS | Time | Gap (%) | S | PS | Time | Gap (%) | S | PS | Time | Gap (%) |
| C1–25 | 9 | 5 | 1 | 1030 | 0.28 | 6 | 3 | 874.1 | 0.20 | 5 | 1 | 1016 | 0.33 | 5 | 3 | 75.1 | 0.19 |
| R1–25 | 12 | 12 | | 103.7 | | 12 | | 11.7 | | 12 | | 145.7 | | 12 | | 23.2 | |
| RC1–25 | 8 | 8 | | 59.4 | | 8 | | 7.91 | | 8 | | 56.5 | | 8 | | 18.8 | |
| C1–50 | 9 | 4 | | 167 | | 6 | | 907.2 | | 5 | | 2065 | | 6 | 1 | 923.7 | 0.01 |
| R1–50 | 12 | 5 | 6 | 2211 | 1.17 | 9 | 3 | 1342 | 1.99 | 5 | 7 | 1102 | 1.24 | 10 | 2 | 2458 | 1.73 |
| RC1–50 | 8 | | 6 | | 6.83 | 1 | 7 | 209.7 | 6.43 | 1 | 7 | 7800 | 6.63 | 1 | 7 | 1487 | 6.39 |

Table 6.9: VRPTWWTC — *ng*-route algorithms — 3 hours.

| | | NGRR | | | | NG-DSSR-G | | | | NG-DSSR-L | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Class | NI | S | PS | Time | Gap (%) | S | PS | Time | Gap (%) | S | PS | Time | Gap (%) |
| C1–25 | 9 | 5 | 4 | 238.7 | 0.27 | 5 | 2 | 1142 | 0.31 | 5 | 4 | 109.5 | 0.41 |
| R1–25 | 12 | 12 | | 37.9 | | 12 | | 74.8 | | 12 | | 12.9 | |
| RC1–25 | 8 | 8 | | 1643 | | 8 | | 158.8 | | 8 | | 29.2 | |
| C1–50 | 9 | 6 | | 1754 | | 4 | | 180.0 | | 5 | 1 | 211.8 | 0.02 |
| R1–50 | 12 | 7 | 5 | 1362 | 2.12 | 6 | 6 | 2519 | 1.63 | 9 | 3 | 1893 | 2.08 |
| RC1–50 | 8 | 1 | 7 | 9347 | 6.17 | 1 | 7 | 2175 | 6.53 | 1 | 7 | 359.7 | 6.66 |

that the lower bound is not already the optimal solution. The horizontal axis indicates the difference in percentage points. We can observe that in general the gap at the root node is often very close to, or the same as, the best gap overall. Indeed, the lowest quartile of the difference is 0 for all the algorithms. This is an interesting result, considering the generally high number of nodes that are evaluated by the BP procedure on such instances.

### 6.2.2 Benchmarking for the VRPTWWTC

Tables 6.8 and 6.9 synthetically present the benchmark results of the first round for the VRPTWWTC. As it was the case for the VRPTW, Friedman's test yields a $p$-value significantly smaller than 0.05, i.e., $p < 2.2\text{e}{-16}$, which justifies us to proceed with Wilcoxon's test. According to its results, DSSR-L outperforms DSSR, NGRR, NG-DSSR-G, and NG-DSSR-G-E with $p < 1\text{e}{-7}$. In addition, we observe that the performances of DSSR-L, NG-DSSR-L, and NG-DSSR-L-E are not significantly different from each other. The lowest average ranking was still achieved by DSSR-L, while DSSR obtained the highest one. Once again, it can be concluded that the local approach performs significantly better than the global approach, even when solving the VRPTWWTC.

As for the VRPTW, we run the four more performing algorithms, namely DSSR-L, NG-DSSR-L, NG-DSSR-L-E, and NGRR, on Solomon's instances

Table 6.10: VRPTWWTC — Benchmarks — 6 hours.

| | | DSSR-L | | | | | NG-DSSR-L-E | | | | | NGRR | | | | | NG-DSSR-L | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Class | NI | S | PS | R | Time | Gap (%) | S | PS | R | Time | Gap (%) | S | PS | R | Time | Gap (%) | S | PS | R | Time | Gap (%) |
| C1–25 | 9 | 7 | 2 | | 3533 | 0.32 | 6 | 3 | | 3054 | 0.21 | 6 | 3 | | 3465 | 0.27 | 7 | 2 | | 5201 | 1.49 |
| R1–25 | 12 | 12 | | | 13.2 | | 12 | | | 23.1 | | 12 | | | 33.3 | | 12 | | | 13.1 | |
| RC1–25 | 8 | 8 | | | 11.2 | | 8 | | | 24.3 | | 8 | | | 1968 | | 8 | | | 96.8 | |
| C1–50 | 9 | 6 | 1 | 1 | 1701 | 0.01 | 6 | 1 | 1 | 607.2 | 0.01 | 6 | | 2 | 930.9 | | 6 | 1 | 2 | 926.8 | 0.02 |
| R1–50 | 12 | 9 | 3 | | 1528 | 2.03 | 10 | 2 | | 3601 | 2.06 | 8 | 4 | | 1676 | 2.16 | 9 | 3 | | 1898 | 2.02 |
| RC1–50 | 8 | 2 | 6 | | 10478 | 6.72 | 1 | 7 | | 1213 | 6.39 | 1 | 7 | | 8058 | 7.21 | 1 | 7 | | 737.8 | 6.70 |
| C1–75 | 9 | 5 | 1 | 3 | 4274 | 0.21 | 5 | 1 | 2 | 1787 | 0.20 | 5 | | 4 | 3614 | | 5 | | 3 | 2992 | |
| R1–75 | 12 | 3 | 9 | | 54.3 | 1.35 | 4 | 8 | | 4923 | 1.34 | 4 | 8 | | 1986 | 1.43 | 4 | 8 | | 2925 | 1.47 |
| RC1–75 | 8 | | 8 | | | 3.12 | | 8 | | | 3.03 | | 8 | | | 3.50 | 1 | 7 | | 17613 | 3.45 |
| C1–100 | 9 | 5 | | 3 | 1896 | | 5 | | 3 | 955.4 | | 5 | | 3 | 4172 | | 5 | | 4 | 1702 | |
| R1–100 | 12 | 2 | 2 | 5 | 9121 | 0.74 | 3 | 1 | 6 | 10726 | 0.80 | 2 | 2 | 3 | 7270 | 0.40 | 3 | 1 | 5 | 6492 | 0.80 |
| RC1–100 | 8 | | 2 | 6 | | 1.27 | | 2 | 6 | | 1.72 | | 3 | 5 | | 1.97 | | 5 | 2 | | 2.05 |

of Series 1 with 25, 50, 75, and 100 customers, with a time limit of six hours. Tables 6.11, 6.12, 6.13, and 6.14 report the detailed results of these experiments, while Table 6.10 summarizes these results. According to the data provided in Table 6.10, increasing the allotted time limit of BP algorithms does not significantly increase the number of solved or partially solved instances in the classes with 25 and 50 customers. Furthermore, it can be observed that NG-DSSR-L is able to solve the root node and provide a feasible solution for five RC class instances with 100 customers. This value reaches down to 2 with the other algorithms. Table 6.10 also reports the number of instances for which CPLEX is able to recover an upper bound in column "R".

We analyze the computational results reported in Tables 6.11, 6.12, 6.13, and 6.14 by utilizing again Friedman's test. The results offered by the test imply that there is a significant performance difference between the algorithms with a $p$-value of 5.093e$-$5. Then, Wilcoxon's signed-rank test is used for the post-hoc analysis, which clearly shows that NGRR is outperformed by the other three algorithms with a $p$-value smaller than 0.01, confirming the previous results. Additionally, NGRR presents the highest ranking statistic while NG-DSSR-L has the lowest one, as it occurs with the VRPTW.

It can be observed from the data provided in the detailed tables, with regard to the three best algorithms, i.e., DSSR-L, NG-DSSR-L, and NG-DSSR-L-E, that on a few instances, NG-DSSR-L offers slightly worse gaps than DSSR-L, and more rarely they are slightly better. This would be consistent with the idea that an algorithm using only elementary routes would have a tighter lower bound than one based on *ng*-routes. However, most of the time the gaps are the same, often because the algorithms find the optimal solution at the root node. There is a small difference between NG-DSSR-L-E and DSSR-L with respect to the gaps, with only a handful of instances

Table 6.11: VRPTWWTC — Best elementary route algorithms — 6 hours, 25 and 50 customers.

| | DSSR-L | | | | | | | NG-DSSR-L-E | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Time | UB | LB | RUB | G(%) | RG(%) | Nodes | Time | UB | LB | RUB | G(%) | RG(%) | Nodes |
| C101–25 | 0.54 | 2465.2 | 2465.2 | 2465.2 | 0 | 0 | 1 | 0.84 | 2465.2 | 2465.2 | 2465.2 | 0 | 0 | 1 |
| C102–25 | 4853 | 2464.2 | 2458.7 | 2464.2 | 0.22 | 0.22 | 171 | 17791 | 2464.2 | 2458.7 | 2464.2 | 0.22 | 0.22 | 167 |
| C103–25 | 21600 | 2457.5 | 2455.5 | 2457.5 | 0.08 | 0.08 | 14 | 21600 | 2457.5 | 2455.4 | 2457.5 | 0.08 | 0.08 | 2 |
| C104–25 | 21600 | 2462.1 | 2448.1 | 2462.1 | 0.57 | 0.57 | 10 | 21600 | 2454.4 | 2448.1 | 2454.4 | 0.25 | 0.25 | 2 |
| C105–25 | 2.84 | 2465.2 | 2464.3 | 2465.2 | 0.03 | 0.03 | 3 | 2.66 | 2465.2 | 2464.3 | 2465.2 | 0.03 | 0.03 | 3 |
| C106–25 | 0.58 | 2465.2 | 2465.2 | 2465.2 | 0 | 0 | 1 | 0.77 | 2465.2 | 2465.2 | 2465.2 | 0 | 0 | 1 |
| C107–25 | 9.19 | 2465.2 | 2461.3 | 2465.2 | 0.15 | 0.15 | 7 | 8.30 | 2465.2 | 2461.3 | 2465.2 | 0.15 | 0.15 | 5 |
| C108–25 | 888.3 | 2465.2 | 2461 | 2465.2 | 0.17 | 0.17 | 13 | 523.3 | 2465.2 | 2461 | 2465.2 | 0.17 | 0.17 | 15 |
| C109–25 | 18972 | 2463.8 | 2457 | 2463.8 | 0.28 | 0.28 | 239 | 21600 | 2463.8 | 2457 | 2463.8 | 0.28 | 0.28 | 297 |
| R101–25 | 0.19 | 978.9 | 978.9 | 978.9 | 0 | 0 | 1 | 0.25 | 978.9 | 978.9 | 978.9 | 0 | 0 | 1 |
| R102–25 | 2.23 | 875.2 | 872.4 | 875.2 | 0.32 | 0.32 | 3 | 2.33 | 875.2 | 872.4 | 875.2 | 0.32 | 0.32 | 3 |
| R103–25 | 9.92 | 744.2 | 742.2 | 744.2 | 0.27 | 0.27 | 9 | 10.74 | 744.2 | 742.2 | 744.2 | 0.27 | 0.27 | 9 |
| R104–25 | 8.29 | 686.8 | 686.5 | 686.8 | 0.03 | 0.03 | 3 | 12.77 | 686.8 | 686.5 | 686.8 | 0.03 | 0.03 | 5 |
| R105–25 | 0.28 | 805.4 | 805.4 | 805.4 | 0 | 0 | 1 | 0.36 | 805.4 | 805.4 | 805.4 | 0 | 0 | 1 |
| R106–25 | 3.12 | 744.5 | 735.5 | 747.2 | 1.22 | 1.59 | 3 | 4.62 | 744.5 | 735.5 | 747.2 | 1.22 | 1.59 | 5 |
| R107–25 | 5.26 | 683.9 | 683.9 | 683.9 | 0 | 0 | 1 | 3.78 | 683.9 | 683.9 | 683.9 | 0 | 0 | 1 |
| R108–25 | 63.09 | 653.4 | 645.7 | 662.1 | 1.19 | 2.54 | 37 | 170.3 | 653.4 | 645.7 | 662.1 | 1.19 | 2.54 | 113 |
| R109–25 | 1.26 | 691.3 | 691.3 | 691.3 | 0 | 0 | 1 | 1.20 | 691.3 | 691.3 | 691.3 | 0 | 0 | 1 |
| R110–25 | 11.33 | 694.1 | 688.3 | 694.1 | 0.84 | 0.84 | 11 | 16.09 | 694.1 | 688.3 | 694.1 | 0.84 | 0.84 | 13 |
| R111–25 | 6.57 | 684.6 | 683 | 684.6 | 0.22 | 0.22 | 5 | 8.72 | 684.6 | 683 | 684.6 | 0.22 | 0.22 | 5 |
| R112–25 | 34.18 | 643 | 637 | 643 | 0.93 | 0.93 | 19 | 45.72 | 643 | 637 | 643 | 0.93 | 0.93 | 17 |
| RC101–25 | 10.25 | 722.4 | 661.4 | 722.4 | 9.21 | 9.21 | 197 | 8.85 | 722.4 | 661.4 | 722.4 | 9.21 | 9.21 | 101 |
| RC102–25 | 2.84 | 602.7 | 602.7 | 602.7 | 0 | 0 | 1 | 4.69 | 601.8 | 601.8 | 601.8 | 0 | 0 | 1 |
| RC103–25 | 24.66 | 588.7 | 588.7 | 588.7 | 0 | 0 | 1 | 27.78 | 585.1 | 585.1 | 585.1 | 0 | 0 | 1 |
| RC104–25 | 28.81 | 572.4 | 572.4 | 572.4 | 0 | 0 | 1 | 44.46 | 572.4 | 572.4 | 572.4 | 0 | 0 | 1 |
| RC105–25 | 1.44 | 681.5 | 681.5 | 681.5 | 0 | 0 | 1 | 8.26 | 681.5 | 681.5 | 681.5 | 0 | 0 | 1 |
| RC106–25 | 3.00 | 595.5 | 595.5 | 595.5 | 0 | 0 | 1 | 5.21 | 595.5 | 595.5 | 595.5 | 0 | 0 | 1 |
| RC107–25 | 6.02 | 548.3 | 548.3 | 548.3 | 0 | 0 | 1 | 44.75 | 548.3 | 548.3 | 548.3 | 0 | 0 | 1 |
| RC108–25 | 12.37 | 544.5 | 544.5 | 544.5 | 0 | 0 | 1 | 50.58 | 544.5 | 544.5 | 544.5 | 0 | 0 | 1 |
| C101–50 | 4.92 | 4929.6 | 4929.6 | 4929.6 | 0 | 0 | 1 | 6.77 | 4929.6 | 4929.6 | 4929.6 | 0 | 0 | 1 |
| C102–50 | 21600 | 4897.2 | 4896.5 | 4897.2 | 0.01 | 0.01 | 3 | 21600 | 4897.2 | 4896.5 | 4897.2 | 0.01 | 0.01 | 2 |
| C103–50 | 21600 | 4869.4[†] | | | | | 0 | 21600 | | | | | | 0 |
| C104–50 | 21600 | | | | | | 0 | 21600 | 4882.5[†] | | | | | 0 |
| C105–50 | 126 | 4894.1 | 4891.3 | 4894.1 | 0.06 | 0.06 | 11 | 112.9 | 4894.1 | 4891.3 | 4894.1 | 0.06 | 0.06 | 11 |
| C106–50 | 8.99 | 4865 | 4865 | 4865 | 0 | 0 | 1 | 10.54 | 4865 | 4865 | 4865 | 0 | 0 | 1 |
| C107–50 | 32.75 | 4867.2 | 4867.2 | 4867.2 | 0 | 0 | 1 | 31.23 | 4867.2 | 4867.2 | 4867.2 | 0 | 0 | 1 |
| C108–50 | 560.2 | 4862.4 | 4862.4 | 4862.4 | 0 | 0 | 1 | 456.2 | 4862.4 | 4862.4 | 4862.4 | 0 | 0 | 1 |
| C109–50 | 9471 | 4862.4 | 4862.4 | 4862.4 | 0 | 0 | 1 | 3025 | 4862.4 | 4862.4 | 4862.4 | 0 | 0 | 1 |
| R101–50 | 2.91 | 1739.9 | 1736.2 | 1740.6 | 0.21 | 0.25 | 11 | 5.25 | 1739.9 | 1736.2 | 1740.6 | 0.21 | 0.25 | 17 |
| R102–50 | 12.35 | 1509.5 | 1509.5 | 1509.5 | 0 | 0 | 1 | 12.35 | 1509.5 | 1509.5 | 1509.5 | 0 | 0 | 1 |
| R103–50 | 689.2 | 1312 | 1310.5 | 1315.6 | 0.11 | 0.39 | 17 | 690.3 | 1312 | 1310.5 | 1312.6 | 0.11 | 0.16 | 19 |
| R104–50 | 1645 | 1132.3 | 1124.8 | 1132.8 | 0.67 | 0.71 | 23 | 9268 | 1132.3 | 1124.8 | 1135 | 0.67 | 0.91 | 209 |
| R105–50 | 33.67 | 1432.7 | 1422.3 | 1435.3 | 0.73 | 0.91 | 45 | 56.85 | 1432.7 | 1422.3 | 1435.3 | 0.73 | 0.91 | 57 |
| R106–50 | 144.9 | 1294.8 | 1293.4 | 1294.8 | 0.10 | 0.10 | 11 | 126.1 | 1294.8 | 1293.4 | 1294.8 | 0.10 | 0.10 | 9 |
| R107–50 | 21600 | 1218.5 | 1206.3 | 1218.5 | 1.01 | 1.01 | 467 | 18330 | 1218.5 | 1206.3 | 1225.7 | 1.01 | 1.60 | 505 |
| R108–50 | 21600 | 1132.1 | 1100.2 | 1132.1 | 2.89 | 2.89 | 376 | 21600 | 1130.4 | 1100.2 | 1130.4 | 2.74 | 2.74 | 3981 |
| R109–50 | 899.14 | 1286.8 | 1275.3 | 1289.2 | 0.90 | 1.09 | 241 | 1107 | 1286.8 | 1275.3 | 1289.2 | 0.90 | 1.09 | 347 |
| R110–50 | 849.6 | 1200 | 1195.8 | 1205.4 | 0.35 | 0.80 | 21 | 529.7 | 1200 | 1195.8 | 1205.4 | 0.35 | 0.80 | 25 |
| R111–50 | 9478 | 1210.6 | 1200.1 | 1211.7 | 0.87 | 0.96 | 251 | 5881 | 1210.6 | 1200.1 | 1211.7 | 0.87 | 0.96 | 223 |
| R112–50 | 21600 | 1139.3 | 1114.8 | 1139.3 | 2.19 | 2.19 | 549 | 21600 | 1130.2 | 1114.8 | 1146.3 | 1.38 | 2.82 | 1077 |
| RC101–50 | 21600 | 1465.8 | 1358.9 | 1465.8 | 7.86 | 7.86 | 45774 | 21600 | 1465.8 | 1358.9 | 1465.8 | 7.86 | 7.86 | 55838 |
| RC102–50 | 21600 | 1304.7 | 1212.3 | 1307 | 7.62 | 7.81 | 4783 | 21600 | 1307 | 1212.3 | 1307 | 7.81 | 7.81 | 2739 |
| RC103–50 | 21600 | 1237.4 | 1151.4 | 1237.4 | 7.47 | 7.47 | 233 | 21600 | 1227.3 | 1146.9 | 1230 | 7.01 | 7.25 | 102 |
| RC104–50 | 343 | 1052 | 1052 | 1052 | 0 | 0 | 1 | 1212 | 1052 | 1052 | 1052 | 0 | 0 | 1 |
| RC105–50 | 21600 | 1384.1 | 1277.6 | 1384.1 | 8.33 | 8.33 | 12471 | 21600 | 1380 | 1277.6 | 1387.5 | 8.01 | 8.60 | 7387 |
| RC106–50 | 20613 | 1223.2 | 1164.4 | 1224 | 5.05 | 5.12 | 5463 | 21600 | 1223.2 | 1164.4 | 1224 | 5.05 | 5.12 | 2687 |
| RC107–50 | 21600 | 1142.7 | 1103.5 | 1142.7 | 3.54 | 3.54 | 498 | 21600 | 1140.3 | 1101.1 | 1140.3 | 3.55 | 3.55 | 133 |
| RC108–50 | 21600 | 1098.1 | 1041.1 | 1098.1 | 5.47 | 5.47 | 196 | 21600 | 1098.1 | 1041.1 | 1098.1 | 5.47 | 5.47 | 44 |

[†] Recovered upper bound.

Table 6.12: VRPTWWTC — Best *ng*-route algorithms — 6 hours, 25 and 50 customers.

| | NGRR | | | | | | | NG-DSSR-L | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Time | UB | LB | RUB | G(%) | RG(%) | Nodes | Time | UB | LB | RUB | G(%) | RG(%) | Nodes |
| C101–25 | 0.63 | 2465.2 | 2465.2 | 2465.2 | 0 | 0 | 1 | 0.67 | 2465.2 | 2465.2 | 2465.2 | 0 | 0 | 1 |
| C102–25 | 16335 | 2464.2 | 2457.8 | 2464.2 | 0.26 | 0.26 | 329 | 17988 | 2464.2 | 2457.2 | 2464.2 | 0.29 | 0.29 | 189 |
| C103–25 | 21600 | 2457.5 | 2454.2 | 2457.5 | 0.13 | 0.13 | 79 | 21600 | 2484.3 | 2450.2 | 2484.3 | 1.39 | 1.39 | 521 |
| C104–25 | 21600 | 2454.4 | 2445.6 | 2454.4 | 0.36 | 0.36 | 42 | 21600 | 2482.2 | 2443.2 | 2490.4 | 1.60 | 1.93 | 29 |
| C105–25 | 2.98 | 2465.2 | 2464.4 | 2465.2 | 0.03 | 0.03 | 3 | 2.93 | 2465.2 | 2464.4 | 2465.2 | 0.03 | 0.03 | 3 |
| C106–25 | 0.73 | 2465.2 | 2465.2 | 2465.2 | 0 | 0 | 1 | 0.80 | 2465.2 | 2465.2 | 2465.2 | 0 | 0 | 1 |
| C107–25 | 16.72 | 2465.2 | 2461.4 | 2465.2 | 0.15 | 0.15 | 9 | 10.85 | 2465.2 | 2461.4 | 2465.2 | 0.15 | 0.15 | 9 |
| C108–25 | 970.7 | 2465.2 | 2458.3 | 2465.2 | 0.28 | 0.28 | 53 | 546.6 | 2465.2 | 2461 | 2465.2 | 0.17 | 0.17 | 17 |
| C109–25 | 21600 | 2463.8 | 2456.3 | 2463.8 | 0.31 | 0.31 | 1187 | 17860 | 2463.8 | 2457 | 2463.8 | 0.28 | 0.28 | 245 |
| R101–25 | 0.27 | 978.9 | 978.9 | 978.9 | 0 | 0 | 1 | 0.24 | 978.9 | 978.9 | 978.9 | 0 | 0 | 1 |
| R102–25 | 2.31 | 875.2 | 872.4 | 875.2 | 0.32 | 0.32 | 3 | 2.57 | 875.2 | 872.4 | 875.2 | 0.32 | 0.32 | 3 |
| R103–25 | 14.83 | 744.2 | 742.2 | 744.2 | 0.27 | 0.27 | 15 | 9.30 | 744.2 | 742.2 | 744.2 | 0.27 | 0.27 | 9 |
| R104–25 | 14.79 | 686.8 | 686.5 | 686.8 | 0.03 | 0.03 | 7 | 6.61 | 686.8 | 686.5 | 686.8 | 0.03 | 0.03 | 3 |
| R105–25 | 0.51 | 805.4 | 805.4 | 805.4 | 0 | 0 | 1 | 0.43 | 805.4 | 805.4 | 805.4 | 0 | 0 | 1 |
| R106–25 | 4.18 | 744.5 | 735.5 | 750.6 | 1.22 | 2.05 | 5 | 4.34 | 744.5 | 735.5 | 747.2 | 1.22 | 1.59 | 5 |
| R107–25 | 4.40 | 683.9 | 683.9 | 683.9 | 0 | 0 | 1 | 2.98 | 683.9 | 683.9 | 683.9 | 0 | 0 | 1 |
| R108–25 | 114.9 | 653.4 | 645.3 | 662.1 | 1.24 | 2.59 | 63 | 27.20 | 653.4 | 645.3 | 667 | 1.24 | 3.35 | 15 |
| R109–25 | 5.62 | 691.3 | 691.3 | 691.3 | 0 | 0 | 1 | 1.11 | 691.3 | 691.3 | 691.3 | 0 | 0 | 1 |
| R110–25 | 49.24 | 694.1 | 682.6 | 694.1 | 1.67 | 1.67 | 133 | 15.80 | 694.1 | 686.7 | 694.1 | 1.07 | 1.07 | 15 |
| R111–25 | 22.53 | 684.6 | 680.4 | 684.6 | 0.62 | 0.62 | 21 | 18.79 | 684.6 | 680.4 | 684.6 | 0.62 | 0.62 | 19 |
| R112–25 | 165.6 | 643 | 631.5 | 643 | 1.81 | 1.81 | 145 | 67.94 | 643 | 636.3 | 643 | 1.04 | 1.04 | 37 |
| RC101–25 | 17.09 | 722.4 | 661.4 | 722.4 | 9.21 | 9.21 | 225 | 10.13 | 722.4 | 661.4 | 722.4 | 9.21 | 9.21 | 123 |
| RC102–25 | 2.94 | 601.8 | 601.8 | 601.8 | 0 | 0 | 1 | 2.94 | 601.8 | 601.8 | 601.8 | 0 | 0 | 1 |
| RC103–25 | 392.4 | 585.1 | 573.1 | 649 | 2.09 | 13.24 | 93 | 5.09 | 585.1 | 585.1 | 585.1 | 0 | 0 | 1 |
| RC104–25 | 5499 | 572.4 | 558.7 | 587.8 | 2.45 | 5.21 | 3477 | 59.47 | 572.4 | 572.4 | 572.4 | 0 | 0 | 1 |
| RC105–25 | 6.42 | 681.5 | 681.1 | 682.7 | 0.05 | 0.23 | 5 | 2.80 | 681.5 | 681.1 | 681.5 | 0.05 | 0.05 | 3 |
| RC106–25 | 3.54 | 595.5 | 595.5 | 595.5 | 0 | 0 | 1 | 2.51 | 595.5 | 595.5 | 595.5 | 0 | 0 | 1 |
| RC107–25 | 150 | 548.3 | 546.9 | 548.3 | 0.24 | 0.24 | 39 | 10.34 | 548.3 | 548.3 | 548.3 | 0 | 0 | 1 |
| RC108–25 | 9673 | 544.5 | 535.5 | 546.3 | 1.67 | 2.01 | 2451 | 681 | 544.5 | 543.3 | 544.5 | 0.22 | 0.22 | 9 |
| C101–50 | 6.32 | 4929.6 | 4929.6 | 4929.6 | 0 | 0 | 1 | 4.84 | 4929.6 | 4929.6 | 4929.6 | 0 | 0 | 1 |
| C102–50 | 21600 | 4897.2[†] | | | | | 0 | 21600 | 4897.2 | 4896.5 | 4897.2 | 0.02 | 0.02 | 3 |
| C103–50 | 21600 | 4886.2[†] | | | | | 0 | 21600 | 4869.4[†] | | | | | 0 |
| C104–50 | 21600 | | | | | | 0 | 21600 | 4890.3[†] | | | | | 0 |
| C105–50 | 124.7 | 4894.1 | 4891.4 | 4894.1 | 0.06 | 0.06 | 9 | 63.48 | 4894.1 | 4891.4 | 4894.4 | 0.06 | 0.06 | 9 |
| C106–50 | 9.94 | 4865 | 4865 | 4865 | 0 | 0 | 1 | 6.3 | 4865 | 4865 | 4865 | 0 | 0 | 1 |
| C107–50 | 57.63 | 4867.2 | 4867.2 | 4867.2 | 0 | 0 | 1 | 25.63 | 4867.2 | 4867.2 | 4867.2 | 0 | 0 | 1 |
| C108–50 | 393.6 | 4862.4 | 4862.4 | 4862.4 | 0 | 0 | 1 | 1024 | 4862.4 | 4862.4 | 4862.4 | 0 | 0 | 1 |
| C109–50 | 4993 | 4862.4 | 4862.4 | 4862.4 | 0 | 0 | 1 | 4435 | 4862.4 | 4862.4 | 4862.4 | 0 | 0 | 1 |
| R101–50 | 10.39 | 1739.9 | 1736.2 | 1740.6 | 0.21 | 0.25 | 35 | 2.32 | 1739.9 | 1736.2 | 1740.6 | 0.21 | 0.25 | 5 |
| R102–50 | 15.62 | 1509.5 | 1509.5 | 1509.5 | 0 | 0 | 1 | 7.79 | 1509.5 | 1509.5 | 1509.5 | 0 | 0 | 1 |
| R103–50 | 1388 | 1312 | 1310.5 | 1312.6 | 0.12 | 0.16 | 47 | 639.2 | 1312 | 1310.5 | 1312.6 | 0.12 | 0.16 | 19 |
| R104–50 | 6627 | 1132.3 | 1124.8 | 1139.9 | 0.67 | 1.35 | 199 | 8880 | 1132.3 | 1124.8 | 1132.3 | 0.67 | 0.67 | 197 |
| R105–50 | 69.64 | 1432.7 | 1422.3 | 1435.3 | 0.73 | 0.91 | 73 | 51.4 | 1432.7 | 1422.3 | 1435.3 | 0.73 | 0.91 | 55 |
| R106–50 | 195.9 | 1294.8 | 1293.5 | 1295.1 | 0.10 | 0.13 | 11 | 64.82 | 1294.8 | 1293.5 | 1295.1 | 0.10 | 0.13 | 5 |
| R107–50 | 21600 | 1223 | 1206.4 | 1223 | 1.38 | 1.38 | 2313 | 21600 | 1220.7 | 1206.4 | 1222.5 | 1.19 | 1.34 | 921 |
| R108–50 | 21600 | 1127.2 | 1099.8 | 1127.2 | 2.49 | 2.49 | 2478 | 21600 | 1129.2 | 1099.8 | 1129.2 | 2.68 | 2.68 | 1118 |
| R109–50 | 1634 | 1286.8 | 1274.1 | 1290.7 | 1.00 | 1.31 | 501 | 825.8 | 1286.8 | 1275.4 | 1289.2 | 0.90 | 1.09 | 313 |
| R110–50 | 3464 | 1200 | 1194 | 1210.2 | 0.51 | 1.36 | 291 | 2076 | 1200 | 1195.9 | 1217.4 | 0.35 | 1.80 | 115 |
| R111–50 | 21600 | 1210.6 | 1186.8 | 1211.7 | 2.00 | 2.10 | 2148 | 4532 | 1210.6 | 1199.6 | 1215 | 0.92 | 1.29 | 155 |
| R112–50 | 21600 | 1140.2 | 1109.8 | 1144.5 | 2.74 | 3.13 | 3645 | 21600 | 1139.2 | 1114.6 | 1139.3 | 2.21 | 2.21 | 1099 |
| RC101–50 | 21600 | 1475.5 | 1359 | 1496.6 | 8.57 | 10.13 | 67197 | 21600 | 1478.4 | 1359 | 1494.4 | 8.79 | 9.97 | 71436 |
| RC102–50 | 21600 | 1306.4 | 1210.2 | 1307 | 7.95 | 8.00 | 3284 | 21600 | 1313.7 | 1212.3 | 1316.2 | 8.36 | 8.57 | 2971 |
| RC103–50 | 21600 | 1260.9 | 1130 | 1277.9 | 11.59 | 13.09 | 2126 | 21600 | 1232.5 | 1144.5 | 1232.5 | 7.69 | 7.69 | 323 |
| RC104–50 | 8057 | 1052 | 1050.4 | 1052 | 0.15 | 0.15 | 73 | 737.8 | 1052 | 1052 | 1052 | 0 | 0 | 1 |
| RC105–50 | 21600 | 1382.3 | 1277.7 | 1382.3 | 8.19 | 8.19 | 5104 | 21600 | 1380 | 1277.7 | 1389.3 | 8.01 | 8.74 | 3706 |
| RC106–50 | 21600 | 1223.2 | 1164.4 | 1224 | 5.05 | 5.12 | 3486 | 21600 | 1223.2 | 1164.4 | 1224 | 5.05 | 5.12 | 3052 |
| RC107–50 | 21600 | 1140.3 | 1100.3 | 1140.3 | 3.64 | 3.64 | 1236 | 21600 | 1140.3 | 1101.2 | 1140.3 | 3.55 | 3.55 | 165 |
| RC108–50 | 21600 | 1098.1 | 1041.2 | 1098.1 | 5.47 | 5.47 | 412 | 21600 | 1098.1 | 1041.2 | 1098.1 | 5.47 | 5.47 | 37 |

[†] Recovered upper bound.

Table 6.13: VRPTWWTC — Best elementary route algorithms — 6 hours, 75 and 100 customers.

| | DSSR-L | | | | | | | NG-DSSR-L-E | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Time | UB | LB | RUB | G(%) | RG(%) | Nodes | Time | UB | LB | RUB | G(%) | RG(%) | Nodes |
| C101–75 | 19.71 | 7399.2 | 7399.2 | 7399.2 | 0 | 0 | 1 | 17.18 | 7399.2 | 7399.2 | 7399.2 | 0 | 0 | 1 |
| C102–75 | 20348 | 7399.2 | 7399.2 | 7399.2 | 0 | 0 | 1 | 21600 | 7399.2[†] | | | | | 0 |
| C103–75 | 21600 | 7399.2[†] | | | | | 0 | 21600 | 7399.2[†] | | | | | 0 |
| C104–75 | 21600 | 7398.3[†] | | | | | 0 | 21600 | | | | | | 0 |
| C105–75 | 80.62 | 7399.2 | 7399.2 | 7399.2 | 0 | 0 | 1 | 27.55 | 7399.2 | 7399.2 | 7399.2 | 0 | 0 | 1 |
| C106–75 | 807.5 | 7397.3 | 7382.3 | 7397.3 | 0.20 | 0.20 | 3 | 254.9 | 7397.3 | 7382.3 | 7397.3 | 0.20 | 0.20 | 3 |
| C107–75 | 112.8 | 7397.3 | 7397.3 | 7397.3 | 0 | 0 | 1 | 108.1 | 7397.3 | 7397.3 | 7397.3 | 0 | 0 | 1 |
| C108–75 | 21600 | 7396.3 | 7381.1 | 7396.3 | 0.21 | 0.21 | 7 | 8527 | 7396.3 | 7381.1 | 7396.3 | 0.21 | 0.21 | 3 |
| C109–75 | 21600 | 7389.2[†] | | | | | 0 | 21600 | 7389.6 | 7375 | 7389.6 | 0.20 | 0.20 | 2 |
| R101–75 | 12.31 | 2435.1 | 2434.8 | 2435.1 | 0.01 | 0.01 | 7 | 7.21 | 2435.1 | 2434.8 | 2435.1 | 0.01 | 0.01 | 3 |
| R102–75 | 124.3 | 2169.1 | 2169.1 | 2169.1 | 0 | 0 | 1 | 78.38 | 2169.1 | 2169.1 | 2169.1 | 0 | 0 | 1 |
| R103–75 | 21600 | 1824.6 | 1819.2 | 1824.6 | 0.29 | 0.29 | 78 | 19582 | 1824.6 | 1819.2 | 1825.6 | 0.29 | 0.35 | 141 |
| R104–75 | 21600 | 1564.8 | 1544 | 1564.8 | 1.34 | 1.34 | 11 | 21600 | 1574.7 | 1544 | 1574.9 | 1.98 | 2.00 | 32 |
| R105–75 | 26.23 | 1931.1 | 1930.1 | 1931.1 | 0.05 | 0.05 | 3 | 25.33 | 1931.1 | 1930.1 | 1931.1 | 0.05 | 0.5 | 3 |
| R106–75 | 21600 | 1847.3 | 1830.8 | 1849.2 | 0.90 | 1.00 | 184 | 21600 | 1846.8 | 1830.8 | 1849.7 | 0.87 | 1.03 | 390 |
| R107–75 | 21600 | 1686.9 | 1664.4 | 1686.9 | 1.35 | 1.35 | 12 | 21600 | 1685.4 | 1664.4 | 1685.4 | 1.26 | 1.26 | 69 |
| R108–75 | 21600 | 1550.7 | 1520.3 | 1550.7 | 2.00 | 2.00 | 11 | 21600 | 1532.8 | 1520.3 | 1551.6 | 0.82 | 2.06 | 23 |
| R109–75 | 21600 | 1780.6 | 1744.5 | 1780.6 | 2.07 | 2.07 | 537 | 21600 | 1780.7 | 1744.5 | 1781.4 | 2.07 | 2.11 | 1906 |
| R110–75 | 21600 | 1684.7 | 1654.4 | 1684.7 | 1.83 | 1.83 | 42 | 21600 | 1686.2 | 1654.4 | 1686.2 | 1.92 | 1.92 | 431 |
| R111–75 | 21600 | 1654.1 | 1636 | 1654.1 | 1.10 | 1.10 | 28 | 21600 | 1650 | 1636 | 1650 | 0.85 | 0.85 | 47 |
| R112–75 | 21600 | 1573.9 | 1553.7 | 1573.9 | 1.30 | 1.30 | 12 | 21600 | 1568.8 | 1553.7 | 1568.8 | 0.97 | 0.97 | 40 |
| RC101–75 | 21600 | 2159.1 | 2117.7 | 2163.7 | 1.95 | 2.17 | 9900 | 21600 | 2158.5 | 2117.7 | 2163.7 | 1.92 | 2.17 | 9880 |
| RC102–75 | 21600 | 2008 | 1947.8 | 2008 | 3.09 | 3.09 | 486 | 21600 | 2006.8 | 1947.8 | 2006.8 | 3.03 | 3.03 | 3190 |
| RC103–75 | 21600 | 1833.3 | 1764.2 | 1833.3 | 3.91 | 3.91 | 20 | 21600 | 1836.4 | 1764.2 | 1836.4 | 4.09 | 4.09 | 112 |
| RC104–75 | 21600 | 1710.6 | 1665.4 | 1710.6 | 2.71 | 2.71 | 4 | 21600 | 1689.8 | 1665.4 | 1689.8 | 1.46 | 1.46 | 2 |
| RC105–75 | 21600 | 2023 | 1967.9 | 2023 | 2.80 | 2.80 | 1566 | 21600 | 2027.1 | 1967.9 | 2033.6 | 3.01 | 3.34 | 5096 |
| RC106–75 | 21600 | 1897.9 | 1834.2 | 1897.9 | 3.47 | 3.47 | 686 | 21600 | 1899.6 | 1834.2 | 1900.3 | 3.56 | 3.60 | 3078 |
| RC107–75 | 21600 | 1768.9 | 1717.1 | 1768.9 | 3.01 | 3.01 | 17 | 21600 | 1773.2 | 1717.1 | 1773.2 | 3.26 | 3.26 | 209 |
| RC108–75 | 21600 | 1693.9 | 1628.9 | 1693.9 | 3.99 | 3.99 | 19 | 21600 | 1692.2 | 1628.9 | 1692.2 | 3.89 | 3.89 | 170 |
| C101–100 | 42.22 | 9827.3 | 9827.3 | 9827.3 | 0 | 0 | 1 | 51.26 | 9827.3 | 9827.3 | 9827.3 | 0 | 0 | 1 |
| C102–100 | 21600 | 9827.3[†] | | | | | 0 | 21600 | 9827.3[†] | | | | | 0 |
| C103–100 | 21600 | 9851.4[†] | | | | | 0 | 21600 | | | | | | 0 |
| C104–100 | 21600 | 9871.2[†] | | | | | 0 | 21600 | 9846.6[†] | | | | | 0 |
| C105–100 | 172.8 | 9827.3 | 9827.3 | 9827.3 | 0 | 0 | 1 | 114.9 | 9827.3 | 9827.3 | 9827.3 | 0 | 0 | 1 |
| C106–100 | 3001 | 9827.3 | 9827.3 | 9827.3 | 0 | 0 | 1 | 714.3 | 9827.3 | 9827.3 | 9827.3 | 0 | 0 | 1 |
| C107–100 | 216.8 | 9827.3 | 9827.3 | 9827.3 | 0 | 0 | 1 | 220.7 | 9827.3 | 9827.3 | 9827.3 | 0 | 0 | 1 |
| C108–100 | 6048 | 9827.3 | 9827.3 | 9827.3 | 0 | 0 | 1 | 3675 | 9827.3 | 9827.3 | 9827.3 | 0 | 0 | 1 |
| C109–100 | 21600 | | | | | | 0 | 21600 | 9827.3[†] | | | | | 0 |
| R101–100 | 859.4 | 2886.5 | 2881.2 | 2888.6 | 0.18 | 0.26 | 193 | 1872 | 2886.5 | 2881.2 | 2888.6 | 0.18 | 0.26 | 423 |
| R102–100 | 17382 | 2593.3 | 2592.5 | 2593.3 | 0.03 | 0.03 | 15 | 17652 | 2593.3 | 2592.5 | 2593.3 | 0.03 | 0.03 | 27 |
| R103–100 | 21600 | 2247.7[†] | | | | | 0 | 21600 | 2251.2[†] | | | | | 0 |
| R104–100 | 21600 | 1987.2[†] | | | | | 0 | 21600 | | | | | | 0 |
| R105–100 | 21600 | 2375.8 | 2363.1 | 2383.4 | 0.54 | 0.86 | 889 | 12654 | 2375.8 | 2363.1 | 2378.7 | 0.54 | 0.66 | 593 |
| R106–100 | 21600 | | | | | | 0 | 21600 | 2244.2[†] | | | | | 0 |
| R107–100 | 21600 | 2068.2[†] | | | | | 0 | 21600 | 2064.7[†] | | | | | 0 |
| R108–100 | 21600 | | | | | | 0 | 21600 | 1980.5[†] | | | | | 0 |
| R109–100 | 21600 | 2152.6 | 2134 | 2152.6 | 0.87 | 0.87 | 24 | 21600 | 2151 | 2134 | 2154.6 | 0.80 | 0.96 | 83 |
| R110–100 | 21600 | | | | | | 0 | 21600 | | | | | | 0 |
| R111–100 | 21600 | 2057.6[†] | | | | | 0 | 21600 | 2058.8[†] | | | | | 0 |
| R112–100 | 21600 | 2014.8[†] | | | | | 0 | 21600 | 1972.4[†] | | | | | 0 |
| RC101–100 | 21600 | 2640.7 | 2610.9 | 2667.1 | 1.14 | 2.15 | 2086 | 21600 | 2668.8 | 2610.9 | 2668.8 | 2.21 | 2.21 | 2927 |
| RC102–100 | 21600 | 2495.3[†] | | | | | 0 | 21600 | 2517.6[†] | | | | | 0 |
| RC103–100 | 21600 | 2300.2[†] | | | | | 0 | 21600 | 2351.5[†] | | | | | 0 |
| RC104–100 | 21600 | 2183.3[†] | | | | | 0 | 21600 | 2157.3[†] | | | | | 0 |
| RC105–100 | 21600 | 2552.5[†] | | | | | 0 | 21600 | 2552.9[†] | | | | | 0 |
| RC106–100 | 21600 | 2391.8[†] | | | | | 0 | 21600 | 2392.9[†] | | | | | 0 |
| RC107–100 | 21600 | 2214 | 2183.3 | 2214 | 1.40 | 1.40 | 7 | 21600 | 2210.3 | 2183.3 | 2210.3 | 1.23 | 1.23 | 2 |
| RC108–100 | 21600 | 2142.7[†] | | | | | 0 | 21600 | 2126.7[†] | | | | | 0 |

[†] Recovered upper bound.

Table 6.14: VRPTWWTC — Best *ng*-route algorithms — 6 hours, 75 and 100 customers.

| | NGRR | | | | | | | NG-DSSR-L | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Time | UB | LB | RUB | G(%) | RG(%) | Nodes | Time | UB | LB | RUB | G(%) | RG(%) | Nodes |
| C101–75 | 20.21 | 7399.2 | 7399.2 | 7399.2 | 0 | 0 | 1 | 12.97 | 7399.2 | 7399.2 | 7399.2 | 0 | 0 | 1 |
| C102–75 | 21600 | 7414.2† | | | | | 0 | 21600 | 7399.2† | | | | | 0 |
| C103–75 | 21600 | 7405† | | | | | 0 | 21600 | 7417.3† | | | | | 0 |
| C104–75 | 21600 | 7423.7† | | | | | 0 | 21600 | 7455.1† | | | | | 0 |
| C105–75 | 95.97 | 7399.2 | 7399.2 | 7399.2 | 0 | 0 | 1 | 65.30 | 7399.2 | 7399.2 | 7399.2 | 0 | 0 | 1 |
| C106–75 | 686.3 | 7397.3 | 7382.4 | 7397.3 | 0.20 | 0.20 | 3 | 467.6 | 7397.3 | 7382.4 | 7397.3 | 0.20 | 0.20 | 3 |
| C107–75 | 216.9 | 7397.3 | 7397.3 | 7397.3 | 0 | 0 | 1 | 60.37 | 7397.3 | 7397.3 | 7397.3 | 0 | 0 | 1 |
| C108–75 | 17048 | 7396.3 | 7381.2 | 7396.3 | 0.21 | 0.21 | 7 | 14354 | 7396.3 | 7381.2 | 7399.2 | 0.21 | 0.24 | 7 |
| C109–75 | 21600 | 7389.2† | | | | | 0 | 21600 | | | | | | 0 |
| R101–75 | 8.15 | 2435.1 | 2434.9 | 2435.1 | 0.01 | 0.01 | 3 | 7.99 | 2435.1 | 2434.9 | 2435.1 | 0.01 | 0.01 | 3 |
| R102–75 | 119.2 | 2169.1 | 2169.1 | 2169.1 | 0 | 0 | 1 | 32.19 | 2169.1 | 2169.1 | 2169.1 | 0 | 0 | 1 |
| R103–75 | 7785 | 1824.6 | 1819.3 | 1824.6 | 0.29 | 0.29 | 35 | 11641 | 1824.6 | 1819.3 | 1824.6 | 0.29 | 0.29 | 61 |
| R104–75 | 21600 | 1571.9 | 1544 | 1571.9 | 1.81 | 1.81 | 39 | 21600 | 1567.5 | 1544 | 1567.5 | 1.52 | 1.52 | 54 |
| R105–75 | 31.83 | 1931.1 | 1930.2 | 1931.1 | 0.05 | 0.05 | 3 | 20.21 | 1931.1 | 1930.2 | 1931.1 | 0.05 | 0.05 | 3 |
| R106–75 | 21600 | 1846.8 | 1829.4 | 1851.6 | 0.95 | 1.21 | 348 | 21600 | 1846.8 | 1830.8 | 1848.9 | 0.87 | 0.99 | 463 |
| R107–75 | 21600 | 1683.1 | 1662.4 | 1683.1 | 1.25 | 1.25 | 69 | 21600 | 1688.6 | 1664 | 1690.5 | 1.48 | 1.59 | 65 |
| R108–75 | 21600 | 1532.8 | 1519.3 | 1541.8 | 0.89 | 1.48 | 19 | 21600 | 1538.3 | 1520.2 | 1538.3 | 1.19 | 1.19 | 21 |
| R109–75 | 21600 | 1771.7 | 1744.6 | 1775.8 | 1.56 | 1.79 | 1293 | 21600 | 1776.1 | 1744.6 | 1776.1 | 1.81 | 1.81 | 1144 |
| R110–75 | 21600 | 1683.1 | 1651 | 1683.9 | 1.95 | 2.00 | 289 | 21600 | 1681 | 1654.4 | 1681.6 | 1.61 | 1.64 | 144 |
| R111–75 | 21600 | 1654.2 | 1632.3 | 1654.2 | 1.34 | 1.34 | 127 | 21600 | 1660.4 | 1635.9 | 1660.4 | 1.50 | 1.50 | 92 |
| R112–75 | 21600 | 1579.6 | 1553.2 | 1579.6 | 1.70 | 1.70 | 132 | 21600 | 1581.3 | 1553.7 | 1581.3 | 1.78 | 1.78 | 70 |
| RC101–75 | 21600 | 2158.5 | 2117.7 | 2159.4 | 1.92 | 1.97 | 11073 | 17613 | 2158.5 | 2117.7 | 2159.4 | 1.92 | 1.97 | 9563 |
| RC102–75 | 21600 | 2001 | 1945.2 | 2001 | 2.87 | 2.87 | 1996 | 21600 | 2001.1 | 1947.8 | 2001.1 | 2.74 | 2.74 | 4586 |
| RC103–75 | 21600 | 1838.8 | 1730.4 | 1838.8 | 6.26 | 6.26 | 3131 | 21600 | 1838.5 | 1764.3 | 1838.5 | 4.21 | 4.21 | 421 |
| RC104–75 | 21600 | 1708.8 | 1645.8 | 1708.8 | 3.83 | 3.83 | 145 | 21600 | 1693.5 | 1665.3 | 1693.5 | 1.69 | 1.69 | 2 |
| RC105–75 | 21600 | 2024.6 | 1968 | 2026 | 2.88 | 2.95 | 6665 | 21600 | 2035.7 | 1968 | 2037.1 | 3.44 | 3.51 | 6500 |
| RC106–75 | 21600 | 1906.7 | 1834.3 | 1906.7 | 3.95 | 3.95 | 3531 | 21600 | 1907.7 | 1834.3 | 1907.7 | 4.00 | 4.00 | 3092 |
| RC107–75 | 21600 | 1750.6 | 1714.7 | 1768.5 | 2.09 | 3.14 | 785 | 21600 | 1788.7 | 1717.2 | 1788.7 | 4.17 | 4.17 | 42 |
| RC108–75 | 21600 | 1693.8 | 1625.2 | 1693.8 | 4.22 | 4.22 | 1402 | 21600 | 1692.1 | 1628.9 | 1692.1 | 3.88 | 3.88 | 38 |
| C101–100 | 38.22 | 9827.3 | 9827.3 | 9827.3 | 0 | 0 | 1 | 25.46 | 9827.3 | 9827.3 | 9827.3 | 0 | 0 | 1 |
| C102–100 | 21600 | 9828† | | | | | 0 | 21600 | 9851.3† | | | | | 0 |
| C103–100 | 21600 | 9856.7† | | | | | 0 | 21600 | 9833.6† | | | | | 0 |
| C104–100 | 21600 | | | | | | 0 | 21600 | 9882.5† | | | | | 0 |
| C105–100 | 140.3 | 9827.3 | 9827.3 | 9827.3 | 0 | 0 | 1 | 274.7 | 9827.3 | 9827.3 | 9827.3 | 0 | 0 | 3 |
| C106–100 | 4940 | 9827.3 | 9827.3 | 9827.3 | 0 | 0 | 1 | 1769 | 9827.3 | 9827.3 | 9827.3 | 0 | 0 | 1 |
| C107–100 | 357.3 | 9827.3 | 9827.3 | 9827.3 | 0 | 0 | 1 | 194.4 | 9827.3 | 9827.3 | 9827.3 | 0 | 0 | 1 |
| C108–100 | 15384 | 9827.3 | 9827.3 | 9827.3 | 0 | 0 | 1 | 6244 | 9827.3 | 9827.3 | 9827.3 | 0 | 0 | 1 |
| C109–100 | 21600 | 9827.3† | | | | | 0 | 21600 | 9827.3† | | | | | 0 |
| R101–100 | 249.3 | 2886.5 | 2881.2 | 2886.5 | 0.18 | 0.18 | 37 | 1557 | 2886.5 | 2881.2 | 2886.5 | 0.18 | 0.18 | 509 |
| R102–100 | 21600 | 2593.3 | 2592.5 | 2593.4 | 0.03 | 0.03 | 11 | 11110 | 2593.3 | 2592.5 | 2593.3 | 0.03 | 0.03 | 13 |
| R103–100 | 21600 | | | | | | 0 | 21600 | | | | | | 0 |
| R104–100 | 21600 | 2004.2† | | | | | 0 | 21600 | 2010.5† | | | | | 0 |
| R105–100 | 14291 | 2375.8 | 2363.1 | 2381.6 | 0.54 | 0.78 | 595 | 6807 | 2375.8 | 2363.1 | 2390.6 | 0.54 | 1.16 | 849 |
| R106–100 | 21600 | | | | | | 0 | 21600 | 2253.4† | | | | | 0 |
| R107–100 | 21600 | | | | | | 0 | 21600 | 2145.8† | | | | | 0 |
| R108–100 | 21600 | 2001† | | | | | 0 | 21600 | | | | | | 0 |
| R109–100 | 21600 | 2150.5 | 2134 | 2150.5 | 0.77 | 0.77 | 72 | 21600 | 2151 | 2134 | 2152.8 | 0.80 | 0.88 | 75 |
| R110–100 | 21600 | 2139.1† | | | | | 0 | 21600 | 2146.7† | | | | | 0 |
| R111–100 | 21600 | | | | | | 0 | 21600 | 2131.2† | | | | | 0 |
| R112–100 | 21600 | | | | | | 0 | 21600 | | | | | | 0 |
| RC101–100 | 21600 | 2668.8 | 2611 | 2668.8 | 2.21 | 2.21 | 3323 | 21600 | 2640.7 | 2611 | 2675.7 | 1.14 | 2.48 | 970 |
| RC102–100 | 21600 | 2518.4† | | | | | 0 | 21600 | 2492.8 | 2422.1 | 2492.8 | 2.92 | 2.92 | 334 |
| RC103–100 | 21600 | 2296.8† | | | | | 0 | 21600 | 2294.7† | | | | | 0 |
| RC104–100 | 21600 | 2260† | | | | | 0 | 21600 | 2163.3† | | | | | 0 |
| RC105–100 | 21600 | 2548.8 | 2492.3 | 2548.8 | 2.27 | 2.27 | 421 | 21600 | 2547.1 | 2492.3 | 2555.3 | 2.20 | 2.53 | 781 |
| RC106–100 | 21600 | 2388.5† | | | | | 0 | 21600 | 2383.8 | 2319.8 | 2383.8 | 2.76 | 2.76 | 330 |
| RC107–100 | 21600 | 2208.9 | 2178.1 | 2208.9 | 1.41 | 1.41 | 22 | 21600 | 2210 | 2183.4 | 2210 | 1.22 | 1.22 | 8 |
| RC108–100 | 21600 | 2144† | | | | | 0 | 21600 | | | | | | 0 |

† Recovered upper bound.

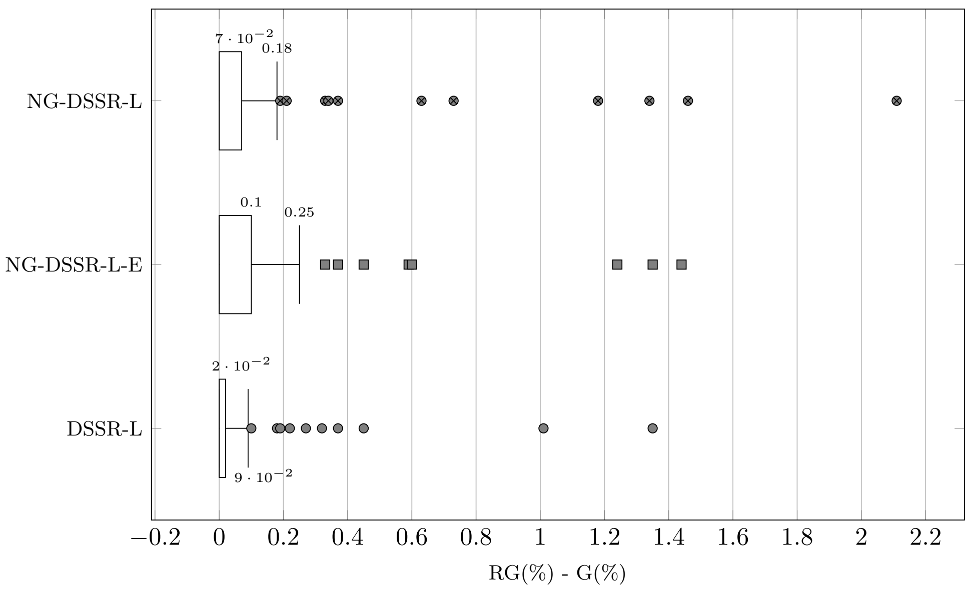


Figure 6.2: VRPTWWTC — Differences between root gap and best gap. Upper quartile and upper whisker values are reported.

reaching differences of at most 1%. Regarding the number of explored nodes of the branch-and-bound tree, DSSR-L generally evaluates fewer nodes than the other algorithms, except the RC class instances with 50 customers.

Finally, Figure 6.2 shows the distribution of the differences between the gap at the root node and the best gap overall, computed in the same way of Figure 6.1, i.e., on instances that are solved or partially solved such that the lower bound is not already the optimal solution. Here, the differences are even smaller than the ones for the VRPTW, and indeed even the median difference is 0 for all algorithms. As in the case of the VRPTW, the quality of the upper bound obtained at the root node, along with the high number of nodes that are evaluated on these instances, suggests that terminating the BP procedure once the root node is evaluated and CPLEX is called to obtain an upper bound can be the basis for an effective heuristic algorithm.

## 6.3 Configuration Analysis

In this section, we discuss the parameter configurations of the best performing algorithms for each problem variant. More precisely, we consider all

Table 6.15: VRPTW — Best parameter configurations.

| | DSSR-L | | | | NG-DSSR-L-E | | | | NGRR | | | | NG-DSSR-L | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Best | Avg | Min | Max | Best | Avg | Min | Max | Best | Avg | Min | Max | Best | Avg | Min | Max |
| `n_col_root` | 7% | 8% | 1% | 24% | 1% | 15% | 1% | 31% | 3% | 31% | 3% | 87% | 3% | 10% | 1% | 38% |
| `n_conc_root` | 3206 | 5894 | 3206 | 9589 | 5711 | 6950 | 5711 | 8777 | 8903 | 5574 | 1702 | 8903 | 4417 | 6011 | 3367 | 9734 |
| `n_col` | 54% | 56% | 36% | 80% | 9% | 23% | 3% | 43% | 45% | 44% | 4% | 79% | 76% | 68% | 43% | 90% |
| `n_conc` | 9981 | 9363 | 8518 | 9981 | 7417 | 8062 | 6867 | 9097 | 9278 | 8593 | 6919 | 9903 | 6728 | 8676 | 8094 | 9998 |
| `tree_nav` | `best` | `best` (4) | | | `breadth` | `breadth` (6) | | | `depth` | `depth` (3) | | | `breadth` | `breadth` (5) | | |
| `dssr_init_s_root` | `whca` | `whca` (6) | | | | | | | | | | | | | | |
| `dssr_init_n_root` | 12% | 24% | 12% | 37% | | | | | | | | | | | | |
| `dssr_path_s_root` | `all-p` | `all-p` (6) | | | `in-btw` | `in-btw` (3) | | | | | | | `in-btw` | `in-btw` (5) | | |
| `dssr_path_n_root` | | | | | 31% | 16% | 15% | 31% | | | | | 70% | 74% | 62% | 88% |
| `dssr_init_s` | `wtca` | `wtca` (4) | | | | | | | | | | | | | | |
| `dssr_init_n` | 21% | 21% | 15% | 32% | | | | | | | | | | | | |
| `dssr_path_s` | `all-p` | `all-p` (6) | | | `in-btw` | `in-btw` (6) | | | | | | | `all-p` | `all-p` (5) | | |
| `dssr_path_n` | | | | | 91% | 88% | 74% | 95% | | | | | | | | |
| `ng_m_root` | | | | | `mix` | `mix` (6) | | | `mix` | `mix` (6) | | | `mix` | `mix` (3) | | |
| `ng_s_root` | | | | | 81% | 64% | 54% | 81% | 22% | 31% | 22% | 38% | 43% | 53% | 43% | 74% |
| `ng_mix_root` | | | | | 10% | 8% | 1% | 25% | 46% | 53% | 32% | 74% | 94% | 89% | 84% | 94% |
| `ng_m` | | | | | `ccr` | `ccr` (3) | | | `ccr` | `ccr` (3) | | | `tt` | `tt` (4) | | |
| `ng_s` | | | | | 17% | 23% | 11% | 35% | 30% | 34% | 20% | 52% | 52% | 73% | 52% | 89% |

the configurations returned by `irace` for a given algorithm, not only the one used for benchmarking. Note that for each algorithm, `irace` recommended six configurations.

Table 6.15 summarizes the configurations of the best algorithms for the VRPTW, while Table 6.16 does it for the VRPTWWTC. In both tables, column "Best" presents the parameter value in the best configuration, i.e., the one used in the benchmarking phase. Column "Avg" presents the average value of a parameter offered by the best six configurations or the most prevalent choice for a categorical parameter such as `tree_nav`. The lowest and highest occurring values of each numerical parameter are given in column "Min" and "Max", respectively. Parameters `dssr_node_s`, `dssr_node_n`, and `ng_mix` with their root node counterparts are not included in the Table 6.16, either because they do not assume values in the configurations, or they are not defined for the considered algorithms. Table 6.15 also ignores all these parameters, except for `ng_mix_root`.

In Table 6.15, we can observe that the most frequent tree navigation strategy is breadth-first search for NG-DSSR-L and NG-DSSR-L-E, while best-first search and depth-first search are preferred by DSSR-L and NGRR, respectively, although by a smaller margin. All the algorithms tend to accept a small percentage of concatenated paths at the root node, reaching only 1% of paths for NG-DSSR-L-E, while generating an average number of paths at the concatenation phase (remember that this parameter can assume at most the value of 10000). Outside the root node, this behavior changes; indeed both the average percentage of accepted paths and the number of generated paths significantly increase for all algorithms. A possible explanation for this phenomenon is that the columns that are generated while evaluating the root node can be of relatively poor quality, or that the algorithm takes benefit from inserting in the RMP a smaller number of columns with very low negative reduced cost early on, while this effect is lessened outside the root node.

DSSR-L seems to assume a different initialization strategy when outside the root node, changing from WHCA to WTCA, while the best values for `dssr_init_n_root` and `dssr_init_n` are all grouped relatively close to 20%. Regarding the insertion strategies, all algorithms seem to prefer either the *all-paths* or the *in-between* strategy, with NG-DSSR-L-E adopting the one that allows examining the least number of paths at the root node (*in-between*, average of 16%). This indicates that the best approach might be to examine a high number of paths at the end of a DSSR iteration in order to detect as many invalid cycles as possible, especially outside the root node.

The mixed strategy is the preferred one for all the algorithms that use *ng*-neighborhoods at the root node, with varying values of the parameter `ng_mix_root`. Outside the root node this seems more varied, with the cheap cycle risk (`ccr`) measure being chosen more often by a small margin. This suggests that metrics different from the travel time used to construct the *ng*-neighborhoods can present a valid alternative and might be the subject of further research. Finally, we can observe that the sizes of the neighborhoods vary quite considerably depending on the algorithm and on the phase of the BP algorithm, which makes it hard to draw definite conclusions. NGRR and NG-DSSR-L experience a slight increase in size outside the root node, which can suggest that it might be necessary to prevent cycles in the final phase of the algorithm (recall that the higher the size of the neighborhoods, the harder it is for a valid *ng*-route to have a cycle), although NG-DSSR-L-E experiences a significant decrease.

For the VRPTWWTC, we can observe in Table 6.16 that the most frequent tree navigation strategy is still best-first search in DSSR-L, while it is depth-first search for the algorithms using *ng*-neighborhoods, differently from the case of the classic VRPTW. DSSR-L tends to create fewer paths during the concatenation phase at the root node, while including a high percentage of the concatenated paths into the RMP. At the child nodes, this behavior is reversed, i.e., the number of concatenated paths dramatically increases and a smaller percentage of the concatenated paths are inserted in the RMP. We can observe that this is virtually the opposite behavior of what occurs in the case of the VRPTW, suggesting that DSSR-L might be able to generate good columns with relatively few label concatenations early in the execution of the algorithm. On the other hand, NG-DSSR-L and NG-DSSR-L-E basically maintain the previously observed behavior, with NG-DSSR-L-E

Table 6.16: VRPTWWTC — Best parameter configurations.

| | DSSR-L | | | | NG-DSSR-L-E | | | | NG-DSSR-L | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Best | Avg | Min | Max | Best | Avg | Min | Max | Best | Avg | Min | Max |
| `n_col_root` | 82% | 42% | 10% | 81% | 4% | 10% | 1% | 35% | 27% | 45% | 26% | 98% |
| `n_conc_root` | 2820 | 3998 | 2764 | 8481 | 8752 | 6900 | 5476 | 8752 | 7910 | 5949 | 1003 | 8320 |
| `n_col` | 31% | 35% | 22% | 63% | 14% | 24% | 11% | 63% | 42% | 37% | 36% | 42% |
| `n_conc` | 9247 | 8902 | 7308 | 9867 | 3079 | 3540 | 1100 | 4519 | 9845 | 8688 | 5234 | 9923 |
| `tree_nav` | `best` | `best` (5) | | | `depth` | `depth` (6) | | | `depth` | `depth` (6) | | |
| `dssr_init_s_root` | `none` | `none` (6) | | | | | | | | | | |
| `dssr_init_n_root` | | | | | | | | | | | | |
| `dssr_path_s_root` | `in-btw` | `in-btw` (6) | | | `all-p` | `all-p` (6) | | | `1-p` | `1-p` (6) | | |
| `dssr_path_n_root` | 25% | 38% | 24% | 47% | | | | | | | | |
| `dssr_init_s` | `none` | `none` (6) | | | | | | | | | | |
| `dssr_init_n` | | | | | | | | | | | | |
| `dssr_path_s` | `in-btw` | `in-btw` (6) | | | `in-btw` | `in-btw` (6) | | | `in-btw` | `in-btw` (6) | | |
| `dssr_path_n` | 39% | 57% | 38% | 80% | 35% | 31% | 26% | 34% | 98% | 81% | 72% | 97% |
| `ng_m_root` | | | | | `tt` | `tt` (6) | | | `tt` | `tt` (5) | | |
| `ng_s_root` | | | | | 17% | 17% | 14% | 24% | 6% | 9% | 4% | 21% |
| `ng_m` | | | | | `tt` | `tt` (5) | | | `tt` | `tt` (6) | | |
| `ng_s` | | | | | 88% | 81% | 50% | 97% | 23% | 24% | 16% | 45% |

accepting only the best 4% of concatenated paths at the root node.

Another interesting finding for DSSR-L is that the critical node sets are initialized as empty, differently from what has been observed in the literature for the global DSSR algorithm and in our own analysis for the VRPTW. It is possible that the critical sets' update procedure works well enough for DSSR-L when solving the VRPTWWTC that no initialization is necessary. Additionally, we can observe that the behavior regarding the insertion strategies is largely the same as the case of the VRPTW, which corroborates the idea that the best approach is to examine as many paths as possible in order to detect invalid cycles.

When we examine *ng*-route parameters, we can observe that the size of the neighborhoods is clearly different depending on the algorithm in use, and tendentially smaller at the root node than at the child nodes. This can be observed especially for NG-DSSR-L. As mentioned before, this can imply that the algorithm permits more cycles at the root node, while being more restrictive in the rest of the tree. The explanation for this might lie in the fact that the algorithm keeps a smaller neighborhood size at the start of the algorithm in order to speed up label extension, while it is forced to prevent more cycles at the end in order to find elementary routes.

Finally, we can note that these results as a whole seem to strongly suggest that the significant differences in the parameter values for the root and child nodes justify our approach of treating these parameters separately.

## 6.4 Size-Dependent Tuning

In this section, we analyze the impact of the tuning set composition on the performances of the obtained parameter configurations for the VRPTWWTC. For that purpose, we further tune DSSR-L and NG-DSSR-L in the following way. First, we tune each algorithm using a tuning set consisting of six instances with only 25 customers. We denote the best obtained configurations as DSSR-L-25 and NG-DSSR-L-25. Later, we tune each algorithm using a tuning set consisting of six instances with only 50 customers, obtaining configurations DSSR-L-50 and NG-DSSR-L-50. In this section, we denote the previously obtained configurations as NG-DSSR-L and DSSR-L.

Analogously to the previous experiments, we run each best configuration on Solomon's benchmark instances with 25, 50, 75, and 100 customers with a time limit of six hours. We then perform the following comparisons, using again Friedman's and Wilcoxon's tests:

1. NG-DSSR-L, NG-DSSR-L-25, and NG-DSSR-L-50 considering only the results on instances with 25 customers;
2. NG-DSSR-L, NG-DSSR-L-25, and NG-DSSR-L-50 on instances with 50 customers;
3. DSSR-L, DSSR-L-25, and DSSR-L-50 on instances with 25 customers;
4. DSSR-L, DSSR-L-25 and DSSR-L-50 on instances with 50 customers;
5. NG-DSSR-L, NG-DSSR-L-25 and NG-DSSR-L-50 on instances with 75 and 100 customers;
6. DSSR-L, DSSR-L-25 and DSSR-L-50 on instances with 75 and 100 customers.

In the first four comparisons, we want to analyze whether tuning an algorithm using a set of instances of a certain size affects significantly its performance on instances of the same size. For instance, an algorithm tuned on instances of only 25 customers, which are relatively easy, might be at a disadvantage when evaluated on instances of 50 customers. Also, an algorithm trained exclusively on instances of 50 customers might have a better performance on similar instances when compared to an algorithm tuned on a more heterogeneous set. Among these comparisons, only test 2 reports significant findings. Here, Friedman's test reports a difference in the performance of the configurations ($p < 0.04$), and Wilcoxon's signed-rank test reports that NG-DSSR-L-50 is significantly better than NG-DSSR-L-25 ($p < 0.05$), while not being significantly different than NG-DSSR-L. This suggests that an algorithm tuned on easier instances can indeed perform more poorly on harder ones than one tuned on such instances.

In the latter two comparisons, we want instead to analyze whether tuning an algorithm using sets of instances of a certain size affects its performance on instances of larger sizes. Note that we cannot perform tests such as the previous ones when taking into account large instances, since tuning the algorithms on instances larger than 50 customers would require a prohibitively long computing time. Indeed, in order to have meaningful results it would be necessary to increase considerably the time limit given to solve each instance. In Test 6, Friedman's test reports a significant difference among the configurations with $p < 0.05$, whereas Wilcoxon's test gives evidence that DSSR-L-50 is better than DSSR-L-25, but under a different significance level ($p < 0.08$). Overall, these tests offer some evidence suggesting that, when tuning algorithms, the best approach would be to construct a tuning set with at least some instances, if not all, with the biggest possible size for which it is feasible to perform the tuning process.

## 6.5 A Comparison between the VRPTW and the VRPTWWTC

To conclude this chapter, we present here a comparative analysis of the VRPTW and the VRPTWWTC with regard to the respective objective function values and computing times. To perform this analysis, we take into account the results of one of the best performing algorithms, NG-DSSR-L, obtained on the 61 Solomon's instances that can be solved to optimality for both problems in the second benchmarking round.

We consider first the comparison with respect to the objective function values. In the VRPTW, the aim is to minimize the total travel time, while in the VRPTWWTC the objective function to minimize is total route duration. For each instance $i$, the optimal solution $S_i^{VRPTW}$ of the VRPTW is prone to have a greater total waiting time than the optimal solution $S_i^{WTC}$ of the VRPTWWTC, since waiting times are not part of the objective function, while $S_i^{WTC}$ might present a greater total travel time than $S_i^{VRPTW}$, in an effort to minimize the overall duration. Let us denote the total travel time of a solution $S$ with $T(S)$, and its total waiting time with $W(S)$. Hence, for each instance $i$, we consider the difference of the total travel times

$$\mathit{\Delta} T_i \coloneqq T(S_i^{VRPTW}) - T(S_i^{WTC}),$$

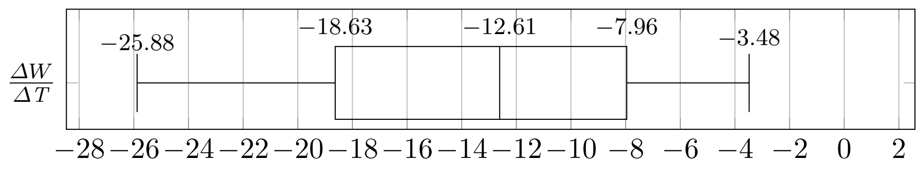


Figure 6.3: Waiting time gain per travel time spent

the difference of the total waiting times

$$\Delta W_i \coloneqq W(S_i^{VRPTW}) - W(S_i^{WTC}),$$

and their ratio $\frac{\Delta W_i}{\Delta T_i}$, which denotes the number of waiting time units reduced for each unit of travel time increase in $S_i^{WTC}$. Here, we are interested in observing the distribution of $\frac{\Delta W_i}{\Delta T_i}$ over each instance $i$, reported in Figure 6.5. Out of the 61 Solomon's instances that can be solved to optimality for both problems, 22 are such that $T(S_i^{VRPTW}) = T(S_i^{WTC})$, i.e., $\frac{\Delta W_i}{\Delta T_i}$ is not defined. Due to the high spread of outliers, we omit them from the figure. We can observe that the median ratio is −12.61, which is indeed quite a large value: a travel time increase of 1% in $S_i^{WTC}$ with respect to $S_i^{VRPTW}$ leads to a decrease in total waiting time of more than 12%. On the other hand, that means that in order to minimize total travel time in the VRPTW, one has to allow a significant increase in the total waiting times.

Finally, we consider the relative difference of the elapsed computing time. For each instance $i$, we denote with $CPU_i^{VRPTW}$ and $CPU_i^{WTC}$ the computing time of NG-DSSR-L on $i$ for the VRPTW and the VRPTWWTC, respectively. The relative difference for instance $i$ is defined as

$$\Delta CPU_i \coloneqq \frac{CPU_i^{WTC} - CPU_i^{VRPTW}}{CPU_i^{VRPTW}}.$$

Figure 6.5 depicts the distribution of the relative differences, again omitting the outliers.

We can observe that the median relative difference is quite high: the median performance of NG-DSSR-L on the VRPTWWTC is almost 15 times slower than on the VRPTW, and the right hand side of the distribution has a significantly large spread. This may be due to an inherent higher difficulty of the VRPTWWTC with respect to the VRPTW, therefore suggesting that it can be worth to investigate ways to strengthen the performance of the available algorithms when solving this problem.

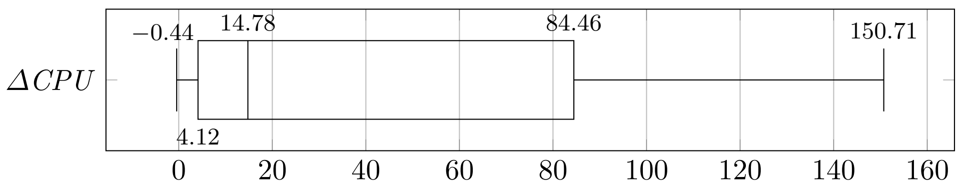


Figure 6.4: Relative elapsed time difference

## 6.6 Conclusions

In this chapter, we have described our experimental methodology, and we have presented the results of our computational experiments. Furthermore, we have discussed the parameter configurations of the best performing algorithms for each problem, and analyzed the impact of the tuning set composition on the algorithmic configurations for the VRPTWWTC.

We can summarize some of our most important results thusly. First, the algorithms following the *local* approach, which aims to guarantee route elementarity by maintaining an individual critical set for each vertex in the graph, consistently outperforms the *global* approach, where a single set is maintained. In particular, it seems that this approach is especially successful when used with the hybrid algorithms under our consideration, i.e., when the critical sets are updated dynamically. This is in contrast to standard NGRR, where the sets are determined a priori. Indeed, the hybrid algorithms DSSR-L, NG-DSSR-L and NG-DSSR-L-E either perform as well as NGRR (as in the case of the VRPTW), or outperform it (as in the case of the VRPTWWTC). This seems to corroborate the validity of the techniques proposed by Martinelli et al. (2014) and Dayarian et al. (2015b).

Furthermore, when the best performing algorithms are able to solve the root node, the upper bound obtained by CPLEX on the MIP just after the evaluation of the root node is usually very good compared to the best upper bound found throughout their execution. This suggests that a method in which only the root node is evaluated, followed by a call to a MIP solver, can be the basis to an effective matheuristic method.

Additionally, we have seen how the parametrization and tuning of complex exact algorithms can be a valid line of research by bringing to light several interesting observations. In particular, we have noted a few trending properties among the best configurations. First of all, during the evaluation of the root node the tendency is to include a relatively small part of the generated columns in the relaxed master problem at the end of each column generation iteration, consisting in the ones with the lowest reduced cost, while reversing this behavior outside the root node. Then, at the end of a decremental state space relaxation iteration, the best approach seems to evaluate as many generated columns as possible to detect invalid cycles. Finally, one might want to increase the size of existing *ng*-neighborhoods in the child nodes of the brand-and-price tree in order to prevent the generation of more cycles. Overall, we can also conclude that having different parameters when evaluating the root node of the search tree seems to be an important feature for a branch-and-price procedure, since a good configuration requires very different parameter values for this phase of the algorithm.

Lastly, we have compared the VRPTW and the VRPTWWTC with respect to their objective function values and computing times, observed on the instances that were solved to optimality by NG-DSSR-L (one of the best performing algorithms according to our analysis) for both problems in the second round of benchmarking. Here, we have observed that the median number of waiting time units reduced in the optimal solution of the VRPTWWTC for each unit of travel time increase with respect to the optimal solution of the VRPTW is quite substantial. On the other hand, the median relative difference in computing time also has a high value, denoting that the VRPTWWTC is harder to solve than the VRPTW. Thus, while using the available BP algorithms to minimize total route duration is an effective way to reduce total waiting time, thus obtaining solutions of high quality for certain practical applications, it is necessary to investigate ways to improve their performance in order to obtain competitive solution methods for this problem.

# Chapter 7

# Conclusions

In this study, we present a comparative analysis of several labeling algorithms embedded in a branch-and-price framework to solve two types of routing problems with time windows. The first problem is the classical vehicle routing problem with time windows, which has been an important object of study in operational research literature for decades. We give a brief overview of this vast body of work, which encompasses a varied landscape of solution methods, both heuristic and exact, in Chapter 2.

In Chapter 3, we discuss more in detail one of the leading exact solution methods for this problem, branch-and-price, consisting in a branch-and-bound algorithm where each linear relaxation is solved with column generation. In particular, we present the classic dynamic programming method, called labeling algorithm, that has been proposed for the solution of the associated pricing problem, the elementary shortest path problem with resource constraints. Additionally, we introduce several well-known methods that have been developed to improve its effectiveness, such as bidirectional label extensions, decremental state space relaxation, and *ng*-route relaxation.

Chapter 4 introduces the second problem under our consideration, a special variant of the vehicle routing problem with time windows, where the objective is to minimize the total route duration. In certain applications, this objective function is arguably more realistic than the minimization of total distance traveled when time windows are present. For instance, it can be used to model cases where the wages of drivers (or other on-board personnel) per time unit worked is significant, such as when vehicles are carrying nurses for home healthcare operations, or in certain possible applications of dial-a-ride problems, such as ride-sharing services. Additionally, route duration minimization can be relevant in cases where vehicles are rented by the distribution company, or where vehicle idle times bear an implicit cost, such as fuel consumption, or energy expenditure in the case of the delivery of perishable goods, which require refrigeration. Here, we discuss the changes in the label structure and dominance rules proposed by Küçükaydın et al. (2014) that are necessary in order to use the labeling algorithm mentioned above to solve the corresponding elementary shortest path problem with resource constraints.

For both problems under our consideration, a significant part of the complexity of the branch-and-price procedure lays in the solution of the pricing problem. In recent years, several methods have been proposed to further improve the effectiveness of the labeling algorithm. More specifically, these methods consist in hybridizations of decremental state space relaxation and *ng*-route relaxation techniques. This is a natural development, since in order to speed up their execution all those techniques relax, each in a different way, the state-space regarding the elementarity resources of the generated routes, whose numerousness may hamper label dominance and thus increase computing time.

However, to the best of our knowledge no systematic analysis of the effectiveness of the hybrid algorithms, especially when compared to decremental state space relaxation and *ng*-route relaxation, has been carried out in the literature. Because of this, our aim in this study is to perform such an analysis. Hence, we do not attempt to propose an algorithm that performs competitively with respect to the state of the art, but rather to identify the underlying properties of those techniques that allow for better performance.

Additionally, it is often the case in the literature that algorithmic strategies and parameters, which form an important set of decisions on the part of the algorithm designer, are simply set manually, at most with the aid of observations or preliminary experiments. In our view, it is necessary to make these decisions more rigorously when carrying out such an analysis. Thus, we parametrize such aspects and use an automated tuner, the `irace` package, in order to obtain the best possible parameter configurations for each algorithm. In Chapter 5, we further elaborate on these concepts, while describing the implementation details of each algorithm and their shared branch-and-price framework.

Finally, in Chapter 6 we introduce the methodology we use to compare the algorithms, once the best available configuration is obtained for each one. Furthermore, we discuss the results of our computational experiments and present some further analyses. In particular, according to our results, the algorithmic paradigm we have denoted as the *local* approach significantly outperforms the alternative, and that the hybrid algorithms that adopt it may even outperform standard *ng*-route relaxation. Thus, the most efficient way to handle elementarity resources seems to consist in maintaining individual critical sets for each vertex in the graph, which are updated dynamically throughout the execution of the algorithm. This seems to corroborate the findings of Martinelli et al. (2014) and Dayarian et al. (2015b), whose proposed methods are part of the current state of the art (Pecin et al., 2017a).

Additionally, the best performing configurations we have obtained demonstrate that tuning the parameters of these complex procedures can be a delicate task. Indeed, the best parameter values might even change depending on which part of the search tree the algorithm is exploring, as the differences in the values at the root node have shown in our case. We have also observed a few recurring qualities in the best performing configurations, which might be desirable aspects of an effective BP algorithm for the considered problems.

Furthermore, we have compared the optimal objective function values for the two problems under our consideration, computed on the instances that one of the best solution methods available to us can solve to optimality for both problems, taking also into account the computing times necessary to obtain the optimal solutions. According to our results, minimizing total route duration allows the algorithm to find solutions where waiting times are reduced considerably with respect to the increase in total travel time, which can be attractive in relevant practical applications. However, the performance of the solution method is adversely affected, suggesting that it might be necessary to investigate ways to improve its effectiveness when minimizing total route duration.

There are several possible directions for future research. In recent years, branch-and-cut-and-price, which adds cut separation to the methods considered in this thesis, has proven to be very effective when solving routing problems. In fact, it has proven successful even when solving the capacitated vehicle routing problem, which is generally more difficult to solve than the version with time windows for branch-and-price. Indeed, since it is a less constrained problem, it is more difficult to fathom infeasible partial paths, and thus the algorithm uses more computational resources. Recently, Pecin et al. (2017a,b) have proposed state-of-the-art methods that combine many of the recent advancements in this field, using such techniques as subset-row inequalities (Jepsen et al., 2008), route enumeration, and strong branching, among others. We did not include these features in this thesis, since we wanted to concentrate on analyzing the performance of labeling algorithms. Indeed, certain features of state-of the-art methods can substantially complicate the structure of the pricing problem. In particular, the use of subset-row inequalities, while currently being considered an indispensable part of any state-of-the-art procedure, increases the difficulty of the pricing problem, since whenever an inequality is added to the model, an associated resource has to be added to the label structure. As we have seen, this can make label domination harder, and thus cause an overall increase in computing time. However, the set of resources associated with these inequalities is separate from the elementarity resources that are manipulated by the algorithms considered in this study. Thus, it is likely that, when a few subset-row inequalities are included, the results that we have presented in this thesis regarding the best performant labeling algorithms still hold, while thusly developed BCP algorithms would still need to be tuned taking into account all the additional parameters. Note that the best performant algorithms are the same for both problem variants under consideration, which have different sets of resources. In the VRPTW, three resources have to be considered besides the elementarity resources, with just as many comparisons for dominance, while in the VRPTWWTC there are five such resources, with four comparisons. In any case, it would be certainly interesting in a future work to study some of these additional state-of-the-art aspects using a similar approach.

The past few years have also seen the emergence of a class of heuristic solution methods that hybridize heuristics with mathematical programming techniques, called *matheuristics*. In particular, Archetti and Speranza (2014) survey the existing matheuristics for routing problems and propose to classify such methods in three groups: decomposition approaches, improvement heuristics, and branch-and-price based approaches. Technically speaking, each algorithm under our consideration can already be used as a matheuristic belonging to the third group. Indeed, it suffices to stop execution once the root node is solved and CPLEX has been called to obtain an integer solution. As we have observed in Chapter 6, the integrality gaps computed at the root node are often very close to the gaps computed with the best available upper bound, suggesting that this can indeed be a promising heuristic method. It would be interesting in the future to consider advanced techniques aimed at improving its effectiveness, since it is often the case that fully evaluating the root node can be a hard task by itself when solving large instances.

Moreover, it would be interesting to apply our methodology to another variant of the vehicle routing problem, which is harder to solve than the ones we have considered: the multi-trip vehicle routing problem with time windows. In a multi-trip routing problem, each vehicle is allowed to return to the depot between delivery routes in order to load/unload deliverables and thus potentially serve more customers during a journey. This case is particularly interesting in certain practical situations, e.g., when the fleet size is low compared to the number of vehicles that are necessary to service all customers in the single-trip variant, when each vehicle can only serve a few customers at a time due to low capacity, or when the planning horizon is relatively long compared to the average travel time between two customers. Cattaruzza et al. (2016) have presented a detailed survey of the literature on such problems. In particular, Hernandez et al. (2014, 2016) have developed effective column generation-based approaches to solve a multi-trip problem with time windows. It would be interesting to compare such algorithms to the ones considered in our study, in particular when the objective is to minimize total route duration. Indeed, François et al. (2017) argue that good solutions with respect to this objective are much more realistic in practice than those obtained when aiming to minimize the total travel time, since they can feature drastically lower waiting times for drivers.

# Bibliography

Archetti, C. and Speranza, M. G. (2014). A survey on matheuristics for routing problems. *EURO Journal on Computational Optimization*, 2(4):223–246.

Atkinson, J. B. (1994). A greedy look-ahead heuristic for combinatorial optimization: an application to vehicle scheduling with time windows. *Journal of the Operational Research Society*, 45(6):673–684.

Baker, E. and Rushinek, S. (1982). Large scale implementation of a time oriented vehicle scheduling model. Technical report, US Department of Transportation, Urban Mass Transit Administration, Washington, D.C.

Baker, E. and Schaffer, J. R. (1986). Computational experience with branch exchange heuristics for vehicle routing problems with time window constraints. *American Journal of Mathematical and Management Sciences*, 6:261—-300.

Baldacci, R., Bartolini, E., Mingozzi, A., and Roberti, R. (2010). An exact solution framework for a broad class of vehicle routing problems. *Computational Management Science*, 7(3):229–268.

Baldacci, R., Mingozzi, A., and Roberti, R. (2011). New route relaxation and pricing strategies for the vehicle routing problem. *Operations research*, 59(5):1269–1283.

Baldacci, R., Mingozzi, A., and Roberti, R. (2012). Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints. *European Journal of Operational Research*, 218(1):1–6.

Ball, M. O. (2011). Heuristics based on mathematical programming. *Surveys in Operations Research and Management Science*, 16(1):21–38.

Bard, J. F., Kontoravdis, G., and Yu, G. (2002). A branch-and-cut procedure for the vehicle routing problem with time windows. *Transportation Science*, 36(2):250–269.

Bettinelli, A., Ceselli, A., and Righini, G. (2011). A branch-and-cut-and-price algorithm for the multi-depot heterogeneous vehicle routing problem with time windows. *Transportation Research Part C: Emerging Technologies*, 19(5):723–740.

Birattari, M. (2009). *Tuning Metaheuristics*, volume 197 of *Studies in Computational Intelligence*. Springer Berlin Heidelberg, Berlin, Heidelberg.

Boland, N., Dethridge, J., and Dumitrescu, I. (2006). Accelerated label setting algorithms for the elementary resource constrained shortest path problem. *Operations Research Letters*, 34(1):58–68.

Bramel, J. and Simchi-Levi, D. (1996). Probabilistic analyses and practical algorithms for the vehicle routing problem with time windows. *Operations Research*, 44(3):501–509.

Bräysy, O. (2003). A reactive variable neighborhood search for the vehicle-routing problem with time windows. *INFORMS Journal on Computing*, 15(4):347–368.

Bräysy, O. and Gendreau, M. (2005a). Vehicle routing problem with time windows, part I: route construction and local search algorithms. *Transportation Science*, 39(1):104–118.

Bräysy, O. and Gendreau, M. (2005b). Vehicle routing problem with time windows, part II: metaheuristics. *Transportation Science*, 39(1):119–139.

Briant, O., Lemaréchal, C., Meurdesoif, P., Michel, S., Perrot, N., and Vanderbeck, F. (2008). Comparison of bundle and classical column generation. *Mathematical Programming*, 113(2):299–344.

Cattaruzza, D., Absi, N., and Feillet, D. (2016). Vehicle routing problems with multiple trips. *4OR*, 14(3):223–259.

Christofides, N., Mingozzi, A., and Toth, P. (1981a). Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Mathematical Programming*, 20(1):255–282.

Christofides, N., Mingozzi, A., and Toth, P. (1981b). State-space relaxation procedures for the computation of bounds to routing problems. *Networks*, 11(2):145–164.

Clarke, G. and Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581.

Conover, W. J. (1999). *Practical nonparametric statistics.* Wiley, Ney York, NY, third edition.

Cook, T. M. and Russell, R. A. (1978). A simulation and statistical analysis of stochastic vehicle routing with timing constraints. *Decision Sciences*, 9(4):673–687.

Cordeau, J.-F., Desaulniers, G., Desrosiers, J., Solomon, M. M., and Soumis, F. (2002). VRP with time windows. In Toth, P. and Vigo, D., editors, *The Vehicle Routing Problem*, chapter 7, pages 157—-193. SIAM, Philadelphia, PA.

Cordeau, J.-F. and Laporte, G. (2007). The dial-a-ride problem: models and algorithms. *Annals of Operations Research*, 153(1):29–46.

Cordeau, J.-F., Laporte, G., and Mercier, A. (2001). A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational Research Society*, 52(8):928–936.

Cordeau, J.-F., Laporte, G., and Mercier, A. (2004). Improved tabu search algorithm for the handling of route duration constraints in vehicle routing problems with time windows. *Journal of the Operational Research Society*, 55(5):542–546.

Cordeau, J.-F. and Maischberger, M. (2012). A parallel iterated tabu search heuristic for vehicle routing problems. *Computers & Operations Research*, 39(9):2033–2050.

Cordone, R. and Calvo, R. W. (2001). A heuristic for the vehicle routing problem with time windows. *Journal of Heuristics*, 7(2):107–129.

Dabia, S., Ropke, S., van Woensel, T., and De Kok, T. (2013). Branch and price for the time-dependent vehicle routing problem with time windows. *Transportation Science*, 47(3):380–396.

Danna, E. and Le Pape, C. (2005). Branch-and-price heuristics: A case study on the vehicle routing problem with time windows. In *Column Generation*, pages 99–129. Springer.

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

Dantzig, G. B. and Wolfe, P. (1960). Decomposition principle for linear programs. *Operations Research*, 8(1):101–111.

Dayarian, I., Crainic, T. G., Gendreau, M., and Rei, W. (2015a). A branch-and-price approach for a multi-period vehicle routing problem. *Computers & Operations Research*, 55:167–184.

Dayarian, I., Crainic, T. G., Gendreau, M., and Rei, W. (2015b). A column generation approach for a multi-attribute vehicle routing problem. *European Journal of Operational Research*, 241(3):888–906.

de Armas, J. and Melián-Batista, B. (2015). Variable Neighborhood Search for a Dynamic Rich Vehicle Routing Problem with time windows. *Computers & Industrial Engineering*, 85:120–131.

Desaulniers, G., Desrosiers, J., Spoorendonk, S., Desaulniers, G., Desrosiers, J., and Spoorendonk, S. (2011). The vehicle routing problem with time windows: state-of-the-art exact solution methods. In *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc., Hoboken, NJ, USA.

Desaulniers, G., Lessard, F., and Hadjar, A. (2008). Tabu Search, Partial Elementarity, and Generalized $k$-Path Inequalities for the Vehicle Routing Problem with Time Windows. *Transportation Science*, 42(3):387–404.

Desaulniers, G., Madsen, O. B., and Ropke, S. (2014). The vehicle routing problem with time windows. In *Vehicle Routing: Problems, Methods, and applications*, chapter 5, pages 119–159. Society for Industrial and Applied Mathematics, Philadelphia, PA.

Desaulniers, G. and Villeneuve, D. (2000). The shortest path problem with time windows and linear waiting costs. *Transportation Science*, 34(3):312–319.

Desrochers, M., Desrosiers, J., and Solomon, M. M. (1992). A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40(2):342–354.

Doerner, K. F. and Schmid, V. (2010). Survey: Matheuristics for Rich Vehicle Routing Problems. In *Hybrid Metaheuristics*, pages 206–221. Springer, Berlin, Heidelberg.

Dror, M. (1994). Note on the complexity of the shortest path models for column generation in VRPTW. *Operations Research*, 42(5):977–978.

Falkenauer, E. and Bouffouix, S. (1991). A genetic algorithm for job shop. In *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, pages 824–829. IEEE Comput. Soc. Press.

Feillet, D. (2010). A tutorial on column generation and branch-and-price for vehicle routing problems. *4OR*, 8(4):407–424.

Feillet, D., Dejax, P., Gendreau, M., and Gueguen, C. (2004). An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, 44(3):216–229.

François, V., Arda, Y., and Crama, Y. (2017). Adaptive large neighborhood search for multi-trip vehicle routing with time windows. *Working paper, Université de Liège.*

Garcia, B.-L., Potvin, J.-Y., and Rousseau, J.-M. (1994). A parallel implementation of the Tabu search heuristic for vehicle routing problems with time window constraints. *Computers & Operations Research*, 21(9):1025–1033.

Gehring, H. and Homberger, J. (2001). A parallel two-phase metaheuristic for routing problems with time windows. *Asia-Pacific Journal of Operational Research*, 18(1):35.

Gendreau, M., Hertz, A., and Laporte, G. (1992). New insertion and postoptimization procedures for the traveling salesman problem. *Operations Research*, 40(6):1086–1094.

Gilmore, P. C. and Gomory, R. E. (1961). A linear programming approach to the cutting-stock problem. *Operations Research*, 9(6):849–859.

Gilmore, P. C. and Gomory, R. E. (1963). A linear programming approach to the cutting stock problem—part II. *Operations Research*, 11(6):863–888.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549.

Hashimoto, H. and Yagiura, M. (2008). A path relinking approach with an adaptive mechanism to control parameters for the vehicle routing problem with time windows. In *EvoCOP 2008: Evolutionary Computation in Combinatorial Optimization*, pages 254–265. Springer, Berlin, Heidelberg.

Hernandez, F., Feillet, D., Giroudeau, R., and Naud, O. (2014). A new exact algorithm to solve the multi-trip vehicle routing problem with time windows and limited duration. *4OR*, 12(3):235–259.

Hernandez, F., Feillet, D., Giroudeau, R., and Naud, O. (2016). Branch-and-price algorithms for the solution of the multi-trip vehicle routing problem with time windows. *European Journal of Operational Research*, 249(2):551–559.

Holland, J. H. J. H. (1992). *Adaptation in natural and artificial systems : an introductory analysis with applications to biology, control, and artificial intelligence*. MIT Press.

Hsu, C.-I., Hung, S.-F., and Li, H.-C. (2007). Vehicle routing problem with time-windows for perishable food delivery. *Journal of Food Engineering*, 80(2):465–475.

Ioachim, I., Gélinas, S., Soumis, F., and Desrosiers, J. (1998). A dynamic programming algorithm for the shortest path problem with time windows and linear node costs. *Networks*, 31(3):193–204.

Ioannou, G., Kritikos, M., and Prastacos, G. (2001). A greedy look-ahead heuristic for the vehicle routing problem with time windows. *Journal of the Operational Research Society*, 52(5):523–537.

Irnich, S. (2008). Resource extension functions: properties, inversion, and generalization to segments. *OR Spectrum*, 30(1):113–148.

Irnich, S. and Villeneuve, D. (2006). The shortest-path problem with resource constraints and $k$-cycle elimination for $k \geq 3$. *INFORMS Journal on Computing*, 18(3):391–406.

Jepsen, M., Petersen, B., Spoorendonk, S., and Pisinger, D. (2008). Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, 56(2):497–511.

Jörnsten, K., Madsen, O. B., and Sørensen, B. (1986). Exact solution of the vehicle routing and scheduling problem with time windows by variable splitting. Technical report, Institute of Mathematical Statistics and Operations Research, Technical University of Denmark, DK-2800 Lyngby, Denmark.

Kallehauge, B. (2008). Formulations and exact algorithms for the vehicle routing problem with time windows. *Computers & Operations Research*, 35(7):2307–2330.

Kallehauge, B., Boland, N., and Madsen, O. B. (2007). Path inequalities for the vehicle routing problem with time windows. *Networks*, 49(4):273–293.

Kallehauge, B., Larsen, J., and Madsen, O. B. (2006). Lagrangian duality applied to the vehicle routing problem with time windows. *Computers & Operations Research*, 33(5):1464–1487.

Kim, S., Chang, K.-N., and Lee, J.-Y. (1995). A descent method with linear programming subproblems for nondifferentiable convex optimization. *Mathematical Programming*, 71(1):17–28.

Knight, K. W. and Hofer, J. P. (1968). Vehicle scheduling with timed and connected calls: a case study. *OR*, 19(3):299.

Kohl, N., Desrosiers, J., Madsen, O. B., Solomon, M. M., and Soumis, F. (1999). 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, 33(1):101–116.

Kohl, N. and Madsen, O. B. (1997). An optimization algorithm for the vehicle routing problem with time windows based on lagrangian relaxation. *Operations Research*, 45(3):395–406.

Kolen, A. W. J., Rinnooy Kan, A. H. G., and Trienekens, H. W. J. M. (1987). Vehicle routing with time windows. *Operations Research*, 35(2):266–273.

Kontoravdis, G. and Bard, J. F. (1995). A GRASP for the vehicle routing problem with time windows. *ORSA Journal on Computing*, 7(1):10–23.

Küçükaydın, H., Arda, Y., and Crama, Y. (2014). Optimization of the service start time for an elementary shortest path problem with time windows. *Working paper, Université de Liège.*

Laporte, G., Toth, P., and Vigo, D. (2013). Vehicle routing: historical perspective and recent contributions. *EURO Journal on Transportation and Logistics*, 2(1-2):1–4.

Letchford, A. N. and Salazar-González, J.-J. (2006). Projection results for vehicle routing. *Mathematical Programming*, 105(2-3):251–274.

Liberatore, F., Righini, G., and Salani, M. (2011). A column generation algorithm for the vehicle routing problem with soft time windows. *4OR*, 9(1):49–82.

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

Lourenço, H. R., Martin, O. C., and Stützle, T. (2010). Iterated local search: framework and applications. In Gendreau, M. and Potvin, J.-Y., editors, *Handbook of Metaheuristics*, pages 363–397. Springer, Boston, MA.

Lübbecke, M. E. and Desrosiers, J. (2005). Selected Topics in Column Generation. *Operations Research*, 53(6):1007–1023.

Lysgaard, J. (2006). Reachability cuts for the vehicle routing problem with time windows. *European Journal of Operational Research*, 175(1):210–223.

Madsen, O. B. (1976). Optimal scheduling of trucks - a routing problem with tight due times for delivery. In Strobel, H., Genser, R., and Etschmaier, M., editors, *Optimization applied to transportation systems*, pages 126–136. IIASA, International Institute for Applied System Analysis, Laxenburgh, Austria.

Marsten, R. E., Hogan, W. W., and Blankenship, J. W. (1975). The Boxstep Method for Large-Scale Optimization. *Operations Research*, 23(3):389–405.

Martinelli, R., Pecin, D., and Poggi, M. (2014). Efficient elementary and restricted non-elementary route pricing. *European Journal of Operational Research*, 239(1):102–111.

Michelini, S., Arda, Y., and Küçükaydın, H. (2018). A comparative study of labeling algorithms for vehicle routing problems with time windows and waiting time costs. *Working paper, Université de Liège.*

Mitchell, J. E. (2002). Branch-and-cut algorithms for combinatorial optimization problems. In Pardalos, P. M. and Resende, M. G., editors, *Handbook of applied optimization*, pages 65–77. Oxford University Press.

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

Moscato, P. and Cotta, C. (2010). A modern introduction to memetic algorithms. In *Handbook of Metaheuristics*, pages 141–183. Springer, Boston, MA.

Nagata, Y. (1997). Edge assembly crossover: a high-power genetic algorithm for the traveling salesman problem. *Proceedings of 7th International Conference on Genetic Algorithms, 1997*, pages 450–457.

Or, I. (1976). *Traveling salesman-type combinatorial optimization problems and their relation to the logistics of regional blood banking.* PhD thesis, Northwestern University, Evanston, IL.

Osman, I. H. (1993). Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41(4):421–451.

Padberg, M. and Rinaldi, G. (1987). Optimization of a 532-city symmetric traveling salesman problem by branch and cut. *Operations Research Letters*, 6(1):1–7.

Pecin, D., Contardo, C., Desaulniers, G., and Uchoa, E. (2017a). New enhancements for the exact solution of the vehicle routing problem with time windows. *INFORMS Journal on Computing*, 29(3):489–502.

Pecin, D., Pessoa, A., Poggi, M., and Uchoa, E. (2017b). Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation*, 9(1):61–100.

Pillac, V., Guéret, C., and Medaglia, A. L. (2013). A parallel matheuristic for the technician routing and scheduling problem. *Optimization Letters*, 7(7):1525–1535.

Pisinger, D. and Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403–2435.

Potvin, J.-Y. (2009). State-of-the-art review: evolutionary algorithms for vehicle routing. *INFORMS Journal on Computing*, 21(4):518–548.

Potvin, J.-Y. and Rousseau, J.-M. (1993). A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, 66(3):331–340.

Potvin, J.-Y. and Rousseau, J.-M. (1995). An exchange heuristic for routeing problems with time windows. *The Journal of the Operational Research Society*, 46(12):1433.

Prescott-Gagnon, E., Desaulniers, G., and Rousseau, L.-M. (2009). A branch-and-price-based large neighborhood search algorithm for the vehicle routing problem with time windows. *Networks*, 54(4):190–204.

Pullen, H. G. M. and Webb, M. H. J. (1967). A computer application to a transport scheduling problem. *The Computer Journal*, 10(1):10–13.

Repoussis, P. P., Paraskevopoulos, D. C., Tarantilis, C. D., and Ioannou, G. (2006). A Reactive Greedy Randomized Variable Neighborhood Tabu Search for the Vehicle Routing Problem with Time Windows. In *Hybrid Metaheuristics*, pages 124–138. Springer, Berlin, Heidelberg.

Repoussis, P. P., Tarantilis, C. D., and Ioannou, G. (2009). Arc-guided evolutionary algorithm for the vehicle routing problem with time windows. *IEEE Transactions on Evolutionary Computation*, 13(3):624–647.

Righini, G. and Salani, M. (2006). Symmetry helps: bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optimization*, 3(3):255–273.

Righini, G. and Salani, M. (2008). New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks*, 51(3):155–170.

Righini, G. and Salani, M. (2009). Decremental state space relaxation strategies and initialization heuristics for solving the orienteering problem with time windows with dynamic programming. *Computers & Operations Research*, 36(4):1191–1203.

Ropke, S. and Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, 40(4):455–472.

Rousseau, L.-M., Gendreau, M., and Pesant, G. (2002). Using constraint-based operators to solve the vehicle routing problem with time windows. *Journal of Heuristics*, 8(1):43–58.

Russell, R. A. (1977). An effective heuristic for the M-tour traveling salesman problem with some side conditions. *Operations Research*, 25(3):517–524.

Russell, R. A. (1995). Hybrid heuristics for the vehicle routing problem with time windows. *Transportation Science*, 29(2):156–166.

Savelsbergh, M. W. P. (1985). Local search in routing problems with time windows. *Annals of Operations Research*, 4(1):285–305.

Savelsbergh, M. W. P. (1990). An efficient implementation of local search algorithms for constrained routing problems. *European Journal of Operational Research*, 47(1):75–85.

Savelsbergh, M. W. P. (1992). The vehicle routing problem with time windows: minimizing route duration. *ORSA journal on computing*, 4(2):146–154.

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *CP-98 (Fourth International Conference* *on Principles and Practice of Constraint Programming)*, pages 417–431. Springer, Berlin, Heidelberg.

Solomon, M. M. (1986). On the worst-case performance of some heuristics for the vehicle routing and scheduling problem with time window constraints. *Networks*, 16(2):161–174.

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265.

Solomon, M. M., Baker, E. K., and Schaffer, J. R. (1988). Vehicle routing and scheduling problems with time window constraints: efficient implementations of solution improvement procedures. In Golden, B. and Assad, A., editors, *Vehicle Routing: Methods and Studies*, pages 85–106. Elsevier Science Publishers, Amsterdam, The Netherlands.

Solomon, M. M. and Desrosiers, J. (1988). Survey paper—time window constrained routing and scheduling p-roblems. *Transportation Science*, 22(1):1–13.

Suzuki, Y. (2011). A new truck-routing approach for reducing fuel consumption and pollutants emission. *Transportation Research Part D: Transport and Environment*, 16(1):73–77.

Taillard, É., Badeau, P., Gendreau, M., Guertin, F., and Potvin, J.-Y. (1997). A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 31(2):170–186.

Thompson, P. M. and Psaraftis, H. N. (1993). Cyclic Transfer Algorithm for Multivehicle Routing and Scheduling Problems. *Operations Research*, 41(5):935–946.

Trienekens, H. W. J. M. (1982). The time constrained vehicle routing problem. Technical report, Erasmus University, Rotterdam, The Netherlands.

# List of Figures

# List of Tables

# Index