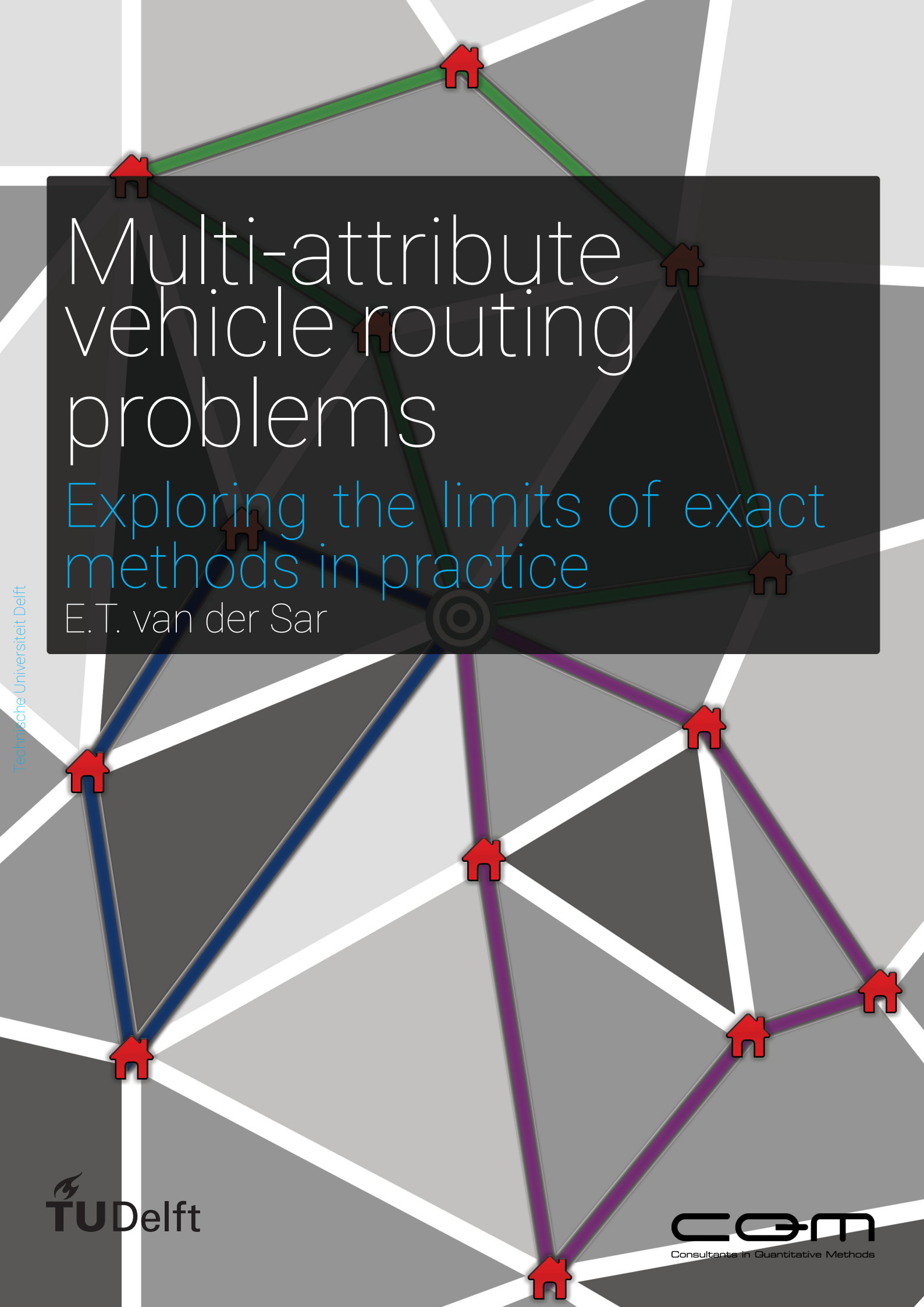


Multi-attribute vehicle routing problems

Exploring the limits of exact methods in practice

E.T. van der Sar

Technische Universiteit Delft

TU Delft

CQM
Consultants in Quantitative Methods

Multi-attribute vehicle routing problems

Exploring the limits of exact methods in practice

by

E.T. van der Sar

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday March 21, 2018 at 10:00 AM.

Student number:	4100034	
Project duration:	May 1, 2017 – March 21, 2018	
Thesis committee:	Prof. dr. ir. K.I. Aardal,	TU Delft
	Dr. ir. L.J.J. van Iersel,	TU Delft, supervisor
	Dr. ir. G.F. Nane,	TU Delft
	Ir. D. Looije,	CQM, supervisor

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Abstract

The companies CQM and EVO-it work together to help many different companies with solving their vehicle routing problems. EVO-it provides the interface of the software that the companies can use for route planning and CQM provides the technology that is used to solve the route planning problems, which concerns a lot of mathematical programming. The difficulty of solving these problems is that real-life problems usually deal with multi-attribute vehicle routing problems, which are problems with multiple characteristics, such as time windows and vehicle restrictions. These characteristics influence the solving process and the quality of the solution of the method used to solve the problem. Since these attributes can vary a lot per company, it is a difficult task for CQM to provide the best solving method for all problem types. Even though CQM has provided a very complex method that can handle many problems, it is unclear how far from the optimum solution the provided solutions are.

The goal of this project was to investigate the improvement possibilities of the current method used by CQM.

In this thesis many different solving methods for the vehicle routing problem are evaluated and their qualities are discussed. Furthermore, information from the companies that are using the software of EVO-it is collected. With this information an overview of the problem types occurring at these companies is provided. Such an overview had not been made before and will be very useful for CQM and EVO-it to make more problem type specific improvements to the solving method.

Furthermore, this thesis describes two exact methods for solving vehicle routing problems similar to the real-life problems occurring at the clients of EVO-it. One of these methods uses a three-index flow formulation and the other one uses a set partitioning formulation and is based on the method described by Baldacci and Mingozzi [5]. It is shown that this latter method can be used to solve instances with up to 40 orders. In addition, both methods can deal with several complicated attributes, including capacity constraints, distance constraints, different type of costs and heterogeneous vehicles. Moreover, it can even deal with instances where not all orders can or need to be scheduled. In such a case, the orders that are not scheduled will get a penalty. To the best of my knowledge, no previous exact method was able to deal with such instances.

Preface

This thesis is the final project of my master Applied Mathematics at Delft University of Technology.

For my final thesis project I really wanted to do something practical where I could use the mathematical knowledge that I had gathered over the years. Furthermore, I wanted to see the type of work a mathematician can do in real life. CQM was the perfect company for this experience, therefore I would like to thank CQM in general for giving me this opportunity. Furthermore, I would like to thank CQM to include me in all of the activities, the fun activities, such as a young CQM excursion, but also the more serious activities, such as meetings. This made me feel like a part of the company and made this into a great internship experience.

I would like to thank my official supervisors Daphne, from CQM, and Leo, from the TU Delft, and also Sander, who has been like a second supervisor at CQM for all the help and support you have given me. Thanks to all of you for your help giving a direction to my research. Thanks to all of you for reading many versions of my thesis and for all the feedback that I got. Furthermore, thanks for thinking along with me when I was trying to understand some difficult parts of the articles I was reading for my thesis.

Thanks to Daphne, for her help many times to solve errors and for always being available to consult. Thanks to Sander, for his positive attitude when I could only see what I did not or could not do and help me focus on what I did do. Thanks to Leo for his enthusiasm when I presented the work I had done and for the very detailed notes that he has given on my thesis report.

This research could not have been carried out without the help of Jan from EVO-it. Thanks to Jan for explaining all of the situations of the VRP-instances at EVO-it and giving me availability to the data from Ritplan. Moreover, thanks for checking the chapter I wrote about EVO-it, and thanks for giving me a place to work, that was much closer to my home than CQM.

To answer some of the questions I had about the papers by Baldacci et al. [6] and Baldacci and Mingozzi [5], I have also had personal contact with the authors prof. Baldacci and prof. Mingozzi. I would like to thank them for their quick responses and helpful answers.

Next, I would like to thank Karen and Tina for being part of my thesis committee and making it possible for me to graduate.

Also, great thanks to Maarten, my boyfriend, who has been a big support throughout this project, not to mention by giving notes on my thesis.

And of course last but not least I would like to thank my parents who have been supportive and interested in what I have been doing, not only for the past nine months but during my complete studies.

E.T. van der Sar
Delft, March 2018

List of abbreviations

CVRP:	Capacitated vehicle routing problem
DARP:	Dial-a-ride problem
MAVRP:	Multi-attribute vehicle routing problem (rich VRP)
MDVRP:	Multiple-depot vehicle routing problem
PDP:	Pick-up and delivery problem
SDVRP:	Site-dependent vehicle routing problem
TSP:	Traveling salesman problem
VRP:	Vehicle routing problem
VRPTW:	Vehicle routing problem with time windows

Table 1: Abbreviations for problems

ABC:	Artificial bee colony
ACO:	Ant colony optimization
ALNS:	Adaptive large neighbourhood search
AMP:	Adaptive memory procedure
ELS :	Evolutionairy local search
GA:	Genetic algorithm
GLS:	Guided local search
GRASP:	Greedy randomized adaptive search procedure
GTS:	Granular tabu search
ILS:	Iterated local search
MA:	Memetic algorithm
MA|PM	Memetic algorithm with population management
PR:	Path relinking
PSO:	Particle swarm optimization
SA:	Simulated annealing
SP:	Set Partitioning
SS:	Scatter search
TS:	Tabu search
UTS:	Unified tabu search
VLSN:	Very large scale neighbourhood search
VND:	Variable neigbourhood descent
VNS:	Variable neighbourhood search

Table 2: Abbreviations for methods

CQM:	Consultants in Quantitative Methods
EVO:	Eigen Vervoer Ondernemingen
ILP:	Integer Linear Programming
SEC:	Subtour elimination constraint

Table 3: Other abbreviations

Contents

1 Introduction

The vehicle routing problem (VRP) is a widespread problem for delivery companies. VRP is the problem of minimizing the costs while delivering or retrieving goods, services or people using a fleet of vehicles. Think of a grocery delivering company, a garbage pick-up company, a plumbery or door-to-door transportation for elderly or disabled people. The VRP is a generalization of the travelling salesman problem (TSP), which uses only one vehicle route to service all customers. The first article on the optimization of a VRP was by Dantzig and Ramser [23] in 1959. Since this moment a broad literature on this subject has been put forward, due to on the one hand the scientific interest as a difficult combinatorial optimization problem and on the other hand the application in many fields in practice. The vehicle routing problem can be extended in many ways, because conditions can vary a lot from one application to another. The software company EVO-it deals with all kinds of variations of the VRP and CQM provides solution methods to help them solve the problems.

This chapter will give an introduction to the company CQM, who is the commissioning company for this thesis project, and their partner in this project EVO-it. Furthermore a short introduction to the classical vehicle routing problem and an overview of the chapters to follow will be given.

1.1. CQM and RitOpt

The research in this thesis was at the request of CQM, Consultants in Quantitative Methods. CQM was founded 35 years ago and is now a company of almost 40 experts on all kinds of mathematical models that help companies with complex problems. CQM focuses on the mathematical issues of companies and consults them on scheduling, transportation, distribution, data science and other issues that can be solved, modelled or optimized in a mathematical way. A grasp out of the projects of CQM are: a predicting and optimization tool for the energy prices of greenhouse horticulture in collaboration with AgroEnergy, predicting the lifetime of LED lighting for Phillips, or scheduling railway maintenance for ProRail.

One of the projects of CQM is in collaboration with EVO-it. EVO-it provides a software program, *Ritplan*, that helps companies that are dealing with VRPs, to schedule their orders and CQM provides the so called intelligence for this program. This holds (1) geocoding, thus transforming the users input of an address to a location on the map, (2) constructing a distance matrix, in this matrix the distances between all orders are provided, and (3) building a route optimization heuristic, which can solve the vehicle routing problem to near optimality. The later is done in the background program called *RitOpt*. The challenge for the optimization program RitOpt is to be able to deal with all the different VRP variants of the customers of EVO-it.

The collaboration between CQM and EVO-it started more than 20 years ago. Since this moment many different users of Ritplan with many variants of VRPs have appeared. This has led to a lot of extensions and changes to RitOpt over the years to service all of the users of Ritplan.

At this moment RitOpt can solve many different types of VRPs. To solve these different types of VRPs, RitOpt uses a hybrid form of simulated annealing and large neighbourhood search. In the current situation RitOpt provides only two ways of solving a problem, where the main difference is in the usage of the neighbourhoods. A 'standard' heuristic is provided for less complicated problems and an 'extended' heuristic is provided for problems with pick-up and delivery, or problems with complications that the standard heuristic was not able

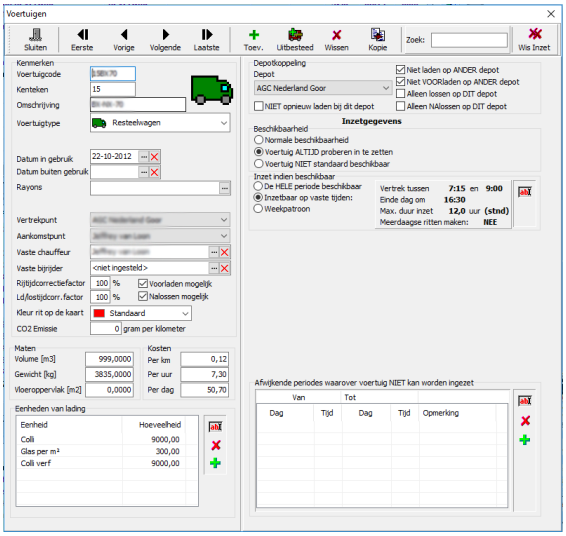
to solve very well.

Even though these methods can solve many problems, they might not always provide a very accurate solution. A heuristic that works well for problem A does not have to work well for problem B. Furthermore, CQM has started creating the heuristics in RitOpt many years ago. Since then many new solving methods have appeared in literature.

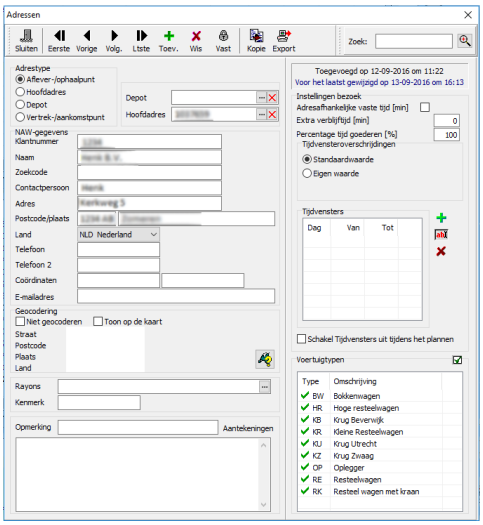
For these reasons, CQM is interested in the possibilities for improving, specializing or extending some parts of the current heuristics in RitOpt. This has lead to the research questions in Section 1.4.

1.2. EVO-it and Ritplan

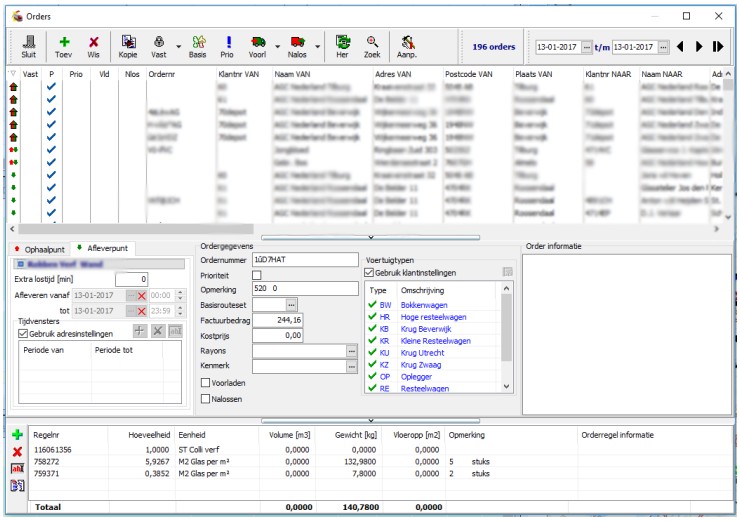
EVO-it is specialized in logistics software for companies that provide their own transport, such as Deli XL and Grolsch. EVO-it is originated from stakeholder EVO (Dutch: Eigen Vervoer Ondernemingen). With their transport scheduling software, Ritplan, EVO-it helps companies with their transport planning. Ritplan will provide a schedule for the companies that minimizes the total costs of the vehicle routes. The companies provide the information of the addresses, orders, vehicles and depots and RitPlan returns the routes for the vehicles.



(a) In this window the companies can provide input for the vehicles.



(b) In this window the companies can provide input for the addresses.



(c) In this window the companies can provide input for the orders.

Figure 1.1: Problem definition in Ritplan software.

Figure 1.2 is provided to give a visual idea of the interface of the software program of EVO-it. Figure 1.1a shows the window for the input of the vehicles. In this window the user can add vehicles and give all information on the vehicles, such as start- and endpoints, capacity, vehicle type and availability. Before providing information for the orders, the information for the addresses has to be provided. This is done in the window shown in Figure 1.1b. For each address the user has to indicate the address type, thus if it concerns a depot or a delivery/pick-up address, and which vehicle types can serve this address. Besides that, the user can also provide a time windows for the addresses. After the addresses have been provided, the user can add the orders that have to be scheduled. This is done in the window shown in Figure 1.1c. To construct an order the user needs to provide a “from" and a “to" address. When the address “from” and “to” are given for an order, the order automatically updates the data from the addresses like time windows and potentially vehicle constraints. At the bottom of the window the demand for the order can be given.

When the user has provided all information, it is time to start the planning process of Ritplan to schedule the tours for the vehicles. This is the moment the components of CQM become important. The information gets processed and RitOpt starts searching for the most optimal solution. When RitOpt has returned a solution to Ritplan, the user can see the vehicle routes of the solution found by RitOpt in the window of Ritplan. Ritplan can visualise the routes on a map as shown in Figure 1.2.

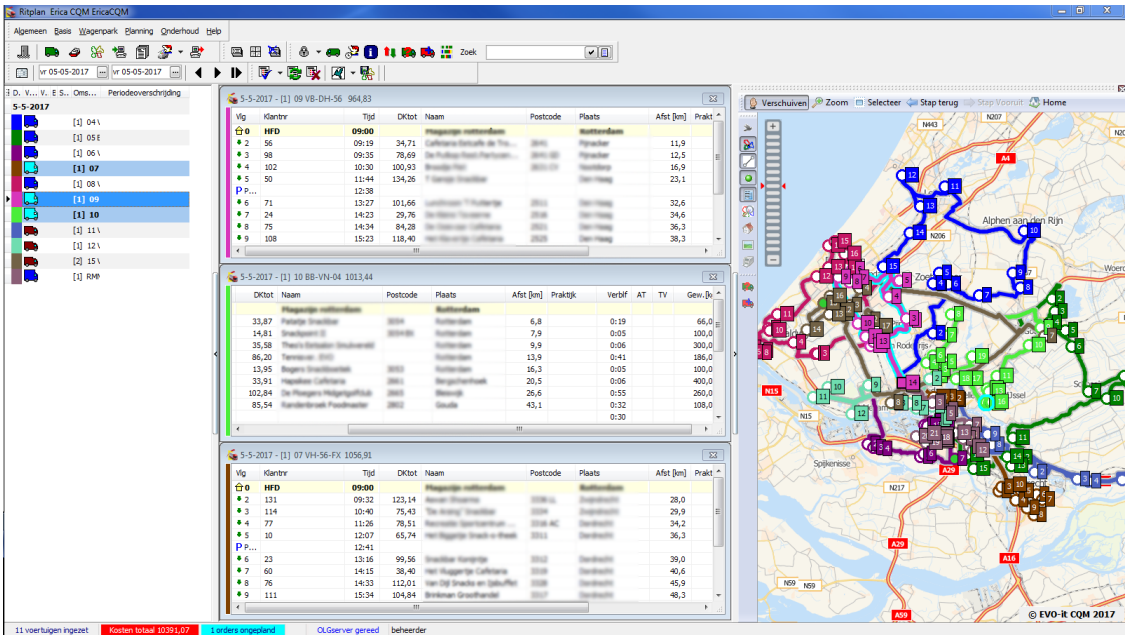


Figure 1.2: An impression of how the user can view the tours constructed in RitPlan.

1.3. The vehicle routing problem

Due to the many different applications of the VRP, many variants of the VRP have have been developed. In this section the classical VRP will be described to give an idea of the essence of the problem.

Most often the VRP is defined under capacity constraints, which is called the capacitated vehicle routing problem (CVRP). The mathematical model of the CVRP is defined as follows. Let $G = (V, A)$ be a complete graph. Where the set of vertices, $V = \{0, ..., n\}$, represent the depot, 0, and the n customers $V_c = \{1, ..., n\}$. A set of m vehicles is based at the depot and have a capacity Q. Each customer needs to be visited by one of the m vehicles and has a nonnegative demand q_i. The arcs $A = \{(i, j) : i, j \in V i \neq j\}$ are associated with a travelling cost c_{ij}. These costs can be linked to the travel distance or travel time. The goal of the VRP is to minimize the total routing costs for a set of at most m routes such that:

1. Each route starts and ends at the depot.

2. Each customer is visited by a route.

3. The total demand of a route does not exceed the capacity Q.

In the case that $c_{ij} = c_{ji}$ for all $(i, j) \in A$, the VRP is symmetric and it is usual to work with edges $E = \{(i, j) : i, j \in V\, i < j\}$. Figure 1.3 gives a visual idea of a solution to a VRP.

Since the VRP is a generalization of the TSP it is NP-hard, which means that there are no polynomial-time algorithms to solve this probem. The NP-hardness of the TSP is proven by Papadimitriou [58].

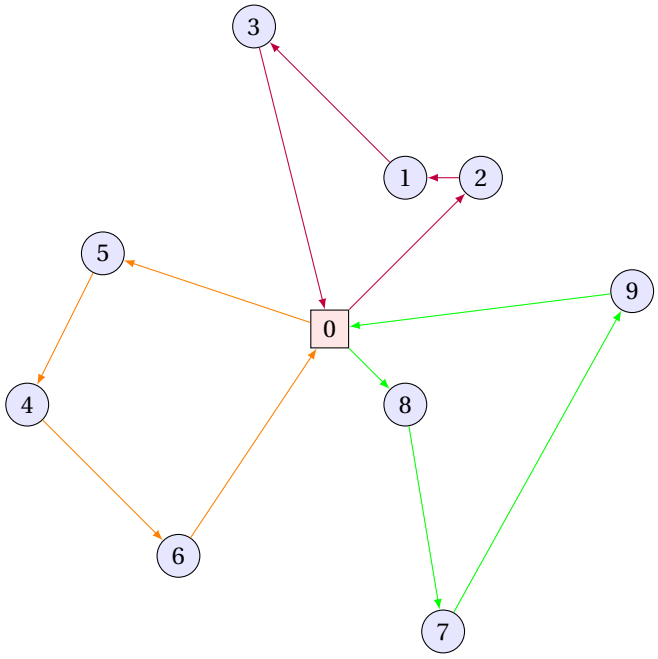


Figure 1.3: An example of a solution to a vehicle routing problem.

1.4. Research question and outline

The goal of this thesis project was to investigate the improvement possibilities of the current solving method for vehicle routing problems used by CQM. To find out where to improve, it is useful to compare the solutions of RitOpt to an exact solution.

Usually exact methods are not applied in practice because the complications of real-life problems make them extremely complex. For this reason it is common to use heuristic methods for real-life problems, which are more flexible. Nevertheless, in the past decade many developments have been made, such as new formulations with stronger bounds, making it more interesting to try to solve real-life instances using exact methods. The intend is to benchmark the solutions of the exact methods against the solutions generated by RitOpt. This does not only give insight in the potential of new exact methods, but can also validate the quality of RitOpt. For these reasons the research question of this thesis is:

Which real-life instances of vehicle routing problems at EVO-it can be solved using exact methods?

The subquestions answered in this thesis to get to this answer are:

1. *What methods are used to solve the VRP in literature?*
   This question is answered in Chapter 2. This chapter gives an overview of the exact methods and the heuristics available in the literature for the capacitated VRP (CVRP), which is in the literature the most common type of VRP. At the end of this chapter a brief summary of the qualities of the different types of solving methods is given.

2. *What type of VRPs appear at EVO-it?*
   This question is answered in Chapter 3. To answer this question the many conversations with Jan Robroeks from EVO-it have been very helpful. Furthermore, the access to the users data in Ritplan helped providing the information in this chapter.

3. *What situations should an exact method be able to tackle to solve instances at EVO-it?*
   In Section 4.1 a selection of problem types is made, based on what came forward in Chapter 3. This

section points out which situations definitely need to be included in the method to be able to solve any of the problems at EVO-it.

4. *Which exact method is most appropriate to be able to include these situations?*
   The remainder of Chapter 4 is the pursuit of an exact method to solve all of the situations that came forward in Section 4.1. This starts with the most intuitive formulation, a two-index flow formulation. Subsequently this formulation is extended to a three-index flow formulation and eventually the switch to a set partitioning formulation is made.

5. *How do the exact methods perform on instances that deal with these situations?*
   To answer this question Chapter 5 reports the computational results of the different exact methods described in Chapter 4.

Finally, the discussion in Chapter 6 will give a critical view on the methods that have been used in this thesis and subsequently several recommendations for future work will be given. The conclusions are shown in Chapter 7.

2

Solving methods for the VRP

Since the introduction of the vehicle routing problem nearly 60 years ago many extensions to the problem have been put forward. Think of vehicle routing problems with time windows (VRPTW), with multiple depots (MDVRP), or with different (heterogeneous) vehicle types (HVRP). For each variant of the VRP multiple methods to solve them have been developed. This results in an even larger set of solving methods. This chapter will provide an overview of the methods available and evaluates them on their qualities.

Since there is such a huge amount of solving methods for the VRP, this chapter will only name the most significant solving methods for the CVRP (and variants). Other overviews on solving methods can be found in Cordeau et al. [21], Laporte [45], Laporte [46], Toth and Vigo [77] and Prodhon and Prins [65] (among many others). This chapter is written based on information and references mainly from these articles.

The types of solution methods distinguished in this chapter are:

1. Exact algorithms, described in Section 2.1,
2. Construction heuristics, described in Section 2.2,
3. Local-search, described in Section 2.4.1,
4. Metaheuristics, described in Section 2.4.

After some examples are given for each type of method, the methods will be qualified with respect to the guiding criteria of Cordeau et al. [20] in Section 2.5.

2.1. Exact algorithms

Although the VRP is a generalization of the TSP, it is much more difficult to solve in practice. For the TSP there exist algorithms that can solve problems exactly up to thousands of vertices, while the best known exact algorithms for VRP can only solve problems with hundreds of vertices.

The best algorithm for the CVRP found in the literature is a *branch-cut-and-price* algorithm from Pecin et al. [59]. This algorithm is able to solve all of the standard classes of instances (A, B, E, F, M and P, which can be found at Uchoa et al. [78]), used for testing exact algorithms for the CVRP, with up to 199 customers (and even some larger problems), while previous methods were only able to solve problems up to 135 customers. The standard classes of instances provide instances from 13 upto 199 customers and were proposed by Augerat [4], by Christofides and Eilon [15], by Fisher [26] and by Christofides [14]. The algorithm of Pecin et al. [59] is based on a combination of the work in seven other recent articles. One can imagine that this method, based on seven other complicated methods, can become very comprehensive and cannot be very easily explained.

Therefore this section only contains the most important concepts of exact algorithms. The two most important mathematical formulations of the CVRP are shown: the set partitioning and the two-index flow formulation. Thereafter the concept of the main approaches used in exact algorithms: branch-and-bound, branch-and-cut and column generation, is explained.

Another exact formulations worth mentioning is the k-degree center tree Formulation of Christofides et al. [16], which will not be discussed here.

2.1.1. Set partitioning formulation

The set partitioning formulation is introduced by Balinski and Quandt [8] in 1963 and has been used widely to model the CVRP and its variants.

Let $\mathscr{R}$ be the set of all feasible routes. Each route $r \in \mathscr{R}$ is associated with a cost c_r and a binary n-vector a_r. The element a_{ir} of vector a_r is a binary coefficient that is equal to 1 if customer i is visited by route r, and 0 otherwise. Furthermore, let x_r be the binary variable equal to 1 if and only if route r is in the optimal solution. The model is

$$
\begin{aligned}
(SP) \qquad & \text{minimize} && \sum_{r\in\mathscr{R}} c_r x_r, && && (2.1)\\
& \text{subject to} && \sum_{r\in\mathscr{R}} a_{ir} x_r = 1, && \forall i \in V_c && (2.2)\\
& && \sum_{r\in\mathscr{R}} x_r = m, && && (2.3)\\
& && x_r \in \{0,1\}, && \forall r \in \mathscr{R} && (2.4)
\end{aligned}
$$

Constraints (2.2) are to make sure each customer is included in a route. Constraint (2.3) requires that m routes are selected. This can also be replaced by a less than or equal sign if not all vehicles need to be used.

Note that vector a_r only represents the customers visited by route r, but not the order of the visits. Thus, to calculate the costs c_r, a TSP must be solved. As mentioned in Section 1.3, solving the TSP is NP-hard. Furthermore, each route r is assumed to be feasible thus, in the case of the CVRP, has to at least satisfy:

$$q \cdot a_r \leq Q$$

Where $q = (q_1, ..., q_n)$ is the vector of all customer demands and Q the capacity of a vehicle.

(SP) can be solved using an integer programming solver, such as Gurobi or CPLEX. In the article of Balinski and Quandt [8] some small instances of the VRP are solved this way. It is pointed out though, that the problem has to be small enough to be able to list $\mathscr{R}$, the set of feasible routes. Unfortunately the number of potential routes is usually too large and computing the c_r coefficients requires solving exponentially many times an NP-hard problem. Therefore, the direct application of this formulation is very limited in practice. Nevertheless, the LP-relaxation, where the binary variables $x_r \in \{0,1\}$ are replaced with linear variables $x_r \geq 0 \forall r \in \mathscr{R}$, can provide a very tight lower bound on the CVRP. The model is very general and can easily take additional constraints into account. Therefore using some smart techniques like column generation, described in Subsection 2.1.5, and q-route relaxation, can make this formulation very interesting again.

2.1.2. Two-index flow formulation

The two-index vehicle flow formulation for the CVRP was introduced by Laporte et al. [48] in 1985. The formulation in this section is described for the symmetrical CVRP, thus on a graph $G = (V, E)$ with edges. Let x_{ij} denote how many times edge $(i, j) \in E$ is traversed. Thus $x_{ij} \in \{0,1\}$, $\forall i, j \in V_c$ and $x_{ij} \in \{0,1,2\}$, $\forall i, j \in V : i \vee j = 0$. Note that $x_{0j} = 2$ corresponds to a single vertex trip between the depot 0 and j.

For a subset $S \subseteq V$ let $\delta(S) := \{(i, j) \in E : i \in S, j \notin S \vee i \notin S, j \in S\}$ and $v(S)$ a lower bound on the number of vehicles needed to cover the customers in S. The two-index model for the CVRP is formulated as follows

$$
\begin{aligned}
(F) \qquad & \text{minimize} && \sum_{(i,j)\in E} c_{ij} x_{ij}, && && (2.5)\\
& \text{subject to} && \sum_{(i,j)\in\delta(\{h\})} x_{ij} = 2, && \forall h \in V_c && (2.6)\\
& && \sum_{j\in V_c} x_{0j} = 2m, && && (2.7)\\
& && \sum_{(i,j)\in\delta(S)} x_{ij} \geq 2v(S), && \forall S \subseteq V_c, |S| \geq 2 && (2.8)\\
& && x_{ij} \in \{0,1\}, && \forall (i,j) \notin \delta(0) && (2.9)\\
& && x_{ij} \in \{0,1,2\}, && \forall (i,j) \in \delta(0) && (2.10)
\end{aligned}
$$

Constraints (2.6) and (2.7) make sure each customer vertex is visited by a route and that there are exactly m routes. Note that m can also be taken as a variable. The constraints in (2.8) are the subtour elimination

constraints (SECs). These constraints make sure that the graph is connected (thus no subtours) and when the value of $v(S)$ is defined as $v(S) = \lceil \frac{q(S)}{Q} \rceil$, where $q(S)$ it the total demand of all customers in S, these constraints make sure that the total demand of a tour does not exceed the maximum capacity. Note that constraints (2.8) can be replaced with:

$$\sum_{i,j \in S} x_{ij} \leq |S| - v(S), \qquad \forall S \subseteq V_c, |S| \geq 2 \tag{2.11}$$

as this also makes sure that there are no subtours and that the capacity constraints are satisfied. The two main issues with the two-index formulation are:

1. The exponential growth of the number of SECs (2.8).
2. A poor lower bound for the LP-relaxation.

This results in a slow solving process. To be able to solve these problems, the branch-and-bound and branch-and-cut approaches have been developed.

2.1.3. Branch-and-bound

Branch-and-bound is a well known concept from combinatorial optimization and up to the late 80s it was the most effective exact approach for the CVRP.

Branch-and-bound is a tree search method that can be used on the two-index formulation. The concept of branch-and-bound is displayed in Figure 2.1 and explained below. Furthermore Figure 2.2 gives an idea of how a branching tree looks.

Step 1. **Initialization**. At the root node of the tree: Find a feasible solution for the CVRP. This can be done using an appropriate heuristic such as the Clarke and Wright savings algorithm described in Section 2.2. Let $\overline{z}$ be the value of the objective function. $\overline{z}$ is the current upper bound for the optimal solution z^* of the CVRP.

Step 2. **Relaxation**. Define a linear relaxation of the problem. For formulation (F) this can be obtained by dropping the constraints (2.8) and by using linear variables instead of integer variables:

(LF)

$$\text{minimize} \quad \sum_{(i,j) \in E} c_{ij} x_{ij}, \tag{2.12}$$

$$\text{subject to} \quad \sum_{(i,j) \in \delta(\{h\})} x_{ij} = 2, \qquad \forall h \in V_c \tag{2.13}$$

$$\sum_{j \in V_c} x_{0j} = 2m, \tag{2.14}$$

$$x_{ij} \geq 0, \qquad \forall (i,j) \in E \tag{2.15}$$

Step 3. **Pick subproblem**. Choose the next subproblem to solve and go to Step 4. The algorithms for CVRP generally adopt a best-bound-first search strategy, i.e. always branch on the node of the search tree with the smallest lower bound value. This method allows for minimization of the number of subproblems solved and is more effective than depth-first strategy.

Step 4. **Solve subproblem**. Solve the current relaxed subproblem using simplex. Let $\underline{z}$ be the optimal value of the objective function. (If the subproblem is infeasible take $\underline{z} = \infty$.) This is a lower bound on the best possible solution "branching" on this node.

Step 5. **Prune**. If $\underline{z} \geq \overline{z}$ the tree can be "*pruned* by bound" at this node. This node can not lead to a better solution than the current one available. This node is *fanthomed* or killed, nodes that can still be used for branching are referred to as *live*. Go back to Step 3.

Otherwise continue.

Step 6. **Eliminate constraints**. If any of the generated SECs are ineffective, eliminate them from the program. This has proved to be a convenient and memory saving device.

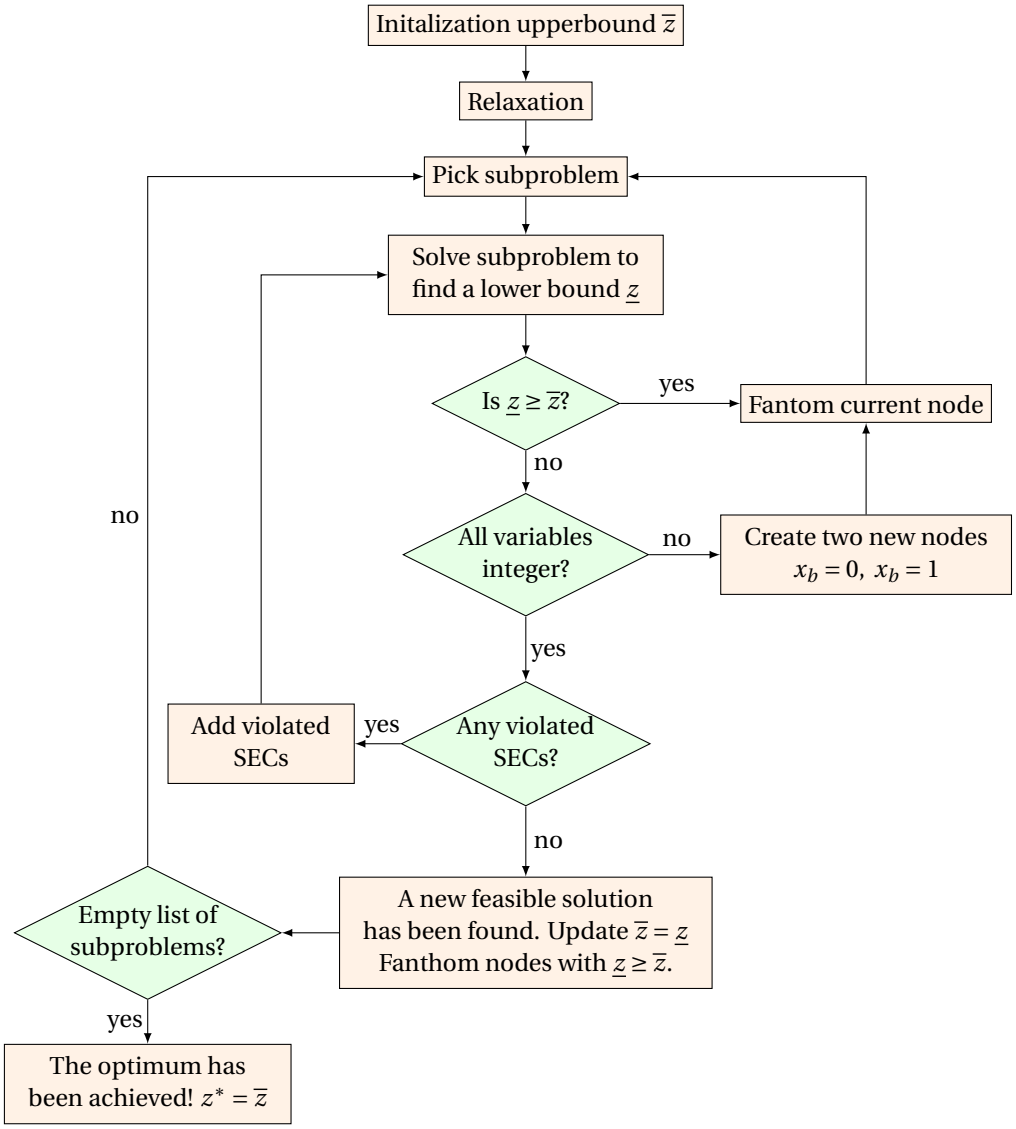


Figure 2.1: Flowchart of Branch-and-Bound algorithm for the CVRP.

Step 7. **Branch**. If not all variables are integer, determine a fractional variable x_b, on which to branch. This can for example be the one with the lowest costs or the one connected to the customer with the highest demand. Apply branching on x_b, i.e. create new subproblems with $x_b = 0$ and $x_b = 1$. The level of the tree is increased and the current node is fanthomed. Go back to Step 3.

Otherwise continue.

Step 8. **Add constraints**. Check if there are any violated SECs. If so, add the violated SEC and go back to Step 4. Otherwise continue.

Step 9. **Update upper bound $\overline{z}$**. A new feasible solution has been obtained. Update $\overline{z} = \underline{z}$. All nodes with $\underline{z} \geq \overline{z}$ can be pruned.

Step 10. **Optimum solution**. If the list of subproblems is empty the optimum solution has been found! Otherwise go to Step 3.

Besides the relaxation given in formulation (LF) there are also other formulations with other relaxation possible to apply branch-and-bound on. For example dropping the SECs and extending the graph with $m-1$ copies of the depot into a complete graph $G' = (V', E')$ results into the Assignment Problem, this formulation

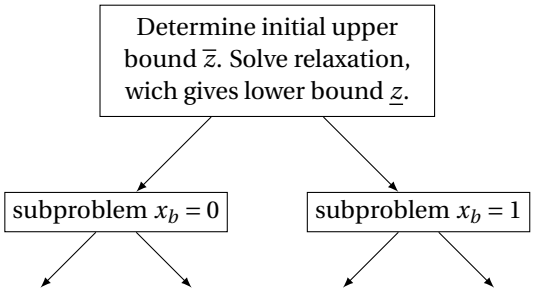


Figure 2.2: Branching tree.

was used by Laporte et al. [49]. Another example is the k-degree center tree relaxation used by Christofides et al. [16].

Using only the basic combinatorial relaxations for the CVRP unfortunately gives poor quality lower bounds. Therefore more sophisticated bounding techniques are needed and most of the exact algorithms also make use of a Lagrangian relaxation technique, illustrated in (2.16).

$$\begin{array}{lll} & \text{Problem} & \text{Relaxation} \\ \min & c^T x & \min \quad c^T x + \lambda^T (A_1 x - b_1) \\ s.t. & A_1 x \leq b_1 & s.t. \quad A_2 x \leq b_2 \\ & A_2 x \leq b_2 & \end{array} \tag{2.16}$$

Where λ is a nonnegative weight. Note that this relaxation gives a lower bound on the problem. For the CVRP, the basic relaxations can be strengthened by dualizing in a Lagrangian fashion, i.e., adding some of the SECs into the objective. Since there are exponentially many of these constraints only a limited subset will be included, which will be iteratively extended by violated constraints. Christofides et al. [16] make use of a Lagrangian relaxation. Next to that they also use a lower bound based on *q-routes*. These are (not necessarily simple) routes on which the total demand is exactly $q \in \{1, ..., Q\}$, starting from the depot traversing a sequence of customers and returning back to the depot.

2.1.4. Branch-and-cut

For a good branch-and-bound method the tightness of the lower bound is crucial. Therefore the LP-relaxation (LF), described in the previous section, is often strengthened by adding cuts, which leads to the branch-and-cut approach. This can be achieved by adding a *cutting plane* step to the problem before the branching step.

The idea of this cutting plane step is as follows. Start with $i = 0$ and let the first relaxation LF_0 be equal to the relaxation (LF).

Step 1. Solve the relaxation LF_i. Suppose the relaxation has the optimal solution x_i^* with value z_i^*.

Step 2. If this solution is valid for the two-index flow formulation (F) then an optimal solution for the CVRP has been found. Stop.

Step 3. If not, find an inequality $\alpha x \leq \beta$ valid for each x that would be feasible in formulation (F), but such that $\alpha x_i^* > \beta$. A valid inequality for formulation (F) that is violated by x_i^* is a *cutting plane*. This cutting plane *separates* x_i^* from (F).

Step 4. Add the cutting plane to the relaxation giving

$$LF_{i+1} := LF_i \cap \{x : \alpha x \leq \beta\}$$

This results to a stronger lower bound $z_i^* < z_{i+1}^* \leq z^*$. Repeat.

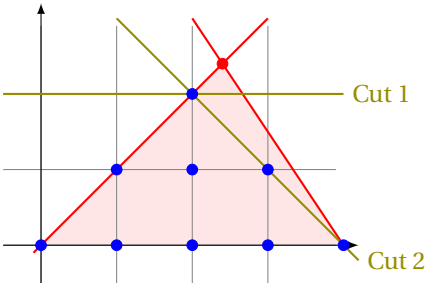


Figure 2.3: Branch-and-cut idea. Suppose the blue points are in the feasible solutions of the original problem and the red area is the relaxation of the original problem. Let the red node be the optimal solution to the relaxation. Cut 1 and Cut 2 are possible cuts to separate the valid solutions from the infeasible solution obtained by the relaxation.

The main difficulty of the cutting plane algorithm is the separation problem that needs to be solved at Step 3. Since there are infinitely many cutting planes separating x_i^*, how can one find one that is most effective? Unfortunately the separation problem is also NP-hard (see Augerat [4]) but thanks to heuristic separation procedures effective constraints can be found more easily.

An example of an algorithm that uses the cutting plane step can be found in Laporte et al. [47]. In this article Gomory cuts are used at the root node in the branch-and-cut algorithm. Since this article has been published, many additional cuts and separation procedures have been developed. For example, different types of *capacity inequalities*, *comb inequalities* which where initially proposed for the TSP, *hypotour inequalities* and *multistar inequalities*.

A good overview of different cuts and separation procedures for the branch-and-bound algorithm for the CVRP can be found in Toth and Vigo [77].

2.1.5. Column generation

Column generation is used in exact algorithms that make use of the set partitioning formulation. As said before, the main drawback of this formulation is the very large number of possible routes. Column generation is a way to cope with this problem. The concept of column generation is as follows.

1. Select a subset of routes.
2. Solve the dual problem for this subset, this is often called the *master* problem.
3. Check if any constraints of the original dual problem are violated, by solving a subproblem.
4. Add the violated constraint(s).
5. Repeat until there are no violated constraints.

This subsection is based on the concept of Agarwal et al. [1], who describe a full column generation approach for the CVRP.

First some adjustments will be made to formulation (SP). When the cost matrix c_{ij} satisfies the triangle inequality the formulation (SP) can be converted to a set covering formulation by replacing the equality sign in Equation (2.2) with a $\geq$ sign. The advantage of this is that only "inclusion-maximal" feasible routes need to be considered. If for example we have routes r_W, r_U, on subsets $W \subset U \subseteq V$, and suppose $c_{r_W} = c_{r_U}$, then only the column representing U has to be considered. This reduces the total set of routes $\mathscr{R}$ and the dual solution space.

Furthermore the LP-relaxation, (LSC), of the set covering problem will be considered, which gives

$$(LSC) \qquad \text{minimize} \quad \sum_{r\in\mathscr{R}} c_r x_r, \tag{2.17}$$

$$\text{subject to} \quad \sum_{r\in\mathscr{R}} a_{ir} x_r \geq 1, \qquad \forall i \in V_c \tag{2.18}$$

$$\sum_{r\in\mathscr{R}} x_r = m, \tag{2.19}$$

$$x_r \geq 0, \qquad \forall r \in \mathscr{R} \tag{2.20}$$

The first step of column generation is to start with a small subset of the routes $B \subset \mathscr{R}$ and solve the dual relaxation $D(B)$ of (LSC) on this subset B.

$$(D(B)) \quad \text{maximize} \quad \sum_{i \in V_c} \lambda_i + m\lambda_0, \tag{2.21}$$

$$\text{subject to} \quad \sum_{i \in V_c} a_{ir}\lambda_i + \lambda_0 \leq c_r, \qquad \forall r \in B \tag{2.22}$$

$$\lambda_i \geq 0, \qquad \forall i \in V_c \tag{2.23}$$

$$\lambda_0 \gtrless 0 \tag{2.24}$$

Suppose λ^* is the optimal solution for $D(B)$. Now if λ^* is feasible for $D(\mathscr{R})$, then it is optimal for $D(\mathscr{R})$, and we have found an optimum for $LSC(\mathscr{R})$. Otherwise one or more constraints from routes $r \in \mathscr{R} \setminus B$ are violated and these routes have to be added to B.

Checking if the solution λ^* is feasible for $\mathscr{R}$ implies computing the column a_k with the least reduced costs:

$$c_k = \min_{r \in \mathscr{R}}(c_r - \sum_{i \in V_c} a_{ir}\lambda_i^* - \lambda_0^*) \tag{2.25}$$

If $\overline{c}_k \geq 0$ then λ^* is feasible for $D(\mathscr{R})$, else column a_k is added to route set B.

Because the feasibility of a route r is assumed, the column a_k, that provides the minimum in Expression 2.25, is equal to y^*, an optimal solution of the subproblem:

$$(\text{P}) \quad \text{minimize} \quad f(y) - \sum_{i \in V_c} y_i\lambda_i^* - \lambda_0^*, \tag{2.26}$$

$$\text{subject to} \quad \sum_{i \in V_c} y_i q_i \leq Q, \qquad \forall r \in \mathscr{R} \tag{2.27}$$

$$y_i \in \{0,1\}, \qquad \forall i \in V_c \tag{2.28}$$

$$\tag{2.29}$$

Where $f(y)$ is the cost of the optimal TSP tour over the customers that are served by y: $i \in V_c : y_i = 1$. Determining $f(y)$ thus requires solving a TSP, which is NP-hard. To solve this, Agarwal et al. [1] use a branch and bound algorithm, for details the reader is referred to the original article.

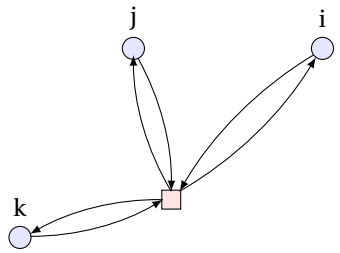
Later successful algorithms make partial use of the set partitioning formulation and combine this with other formulations and extra cuts to solve the CVRP.

2.2. Construction heuristics

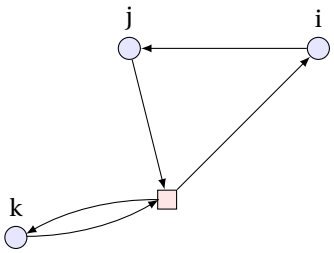
The first heuristics to solve the CVRP were construction heuristics and they are still used to produce initial solutions in many software implementations. The characteristics of construction heuristics are that they start from an empty solution and iteratively build routes in a greedy manner, inserting one or more customers until all customers are routed.

2.2.1. Clarke and Wright Savings method

The most famous example of a construction heuristic is the savings heuristic of Clarke and Wright [17] (CWS). This heuristic initially starts with a route $(0,i,0)$ for each customer $i \in V_c$ as in Figure 2.4a. For each iteration two routes are merged for which the *saving* is maximal, which is done for i and j in Figure2.4b. The saving of merging two routes is computed as follows. Let r_i be a route with customer i at the end/begin and r_j be a route with customer j at the end/begin. Merging the two routes r_i and r_j has saving $s_{ij} = c_{i0} + c_{0j} - c_{ij}$.



(a) Initial routes.



(b) Routes with customers i and j are merged.

Figure 2.4: Example of the Clarke and Wright savings algorithm.

The popularity of the savings method is due to the simplicity and speed, it is easy to understand and can be coded simple and short. Nevertheless the method is not that flexible. Although it is not difficult to add constraints in the method the accuracy of the solution may decrease quickly.

2.2.2. Two phase methods

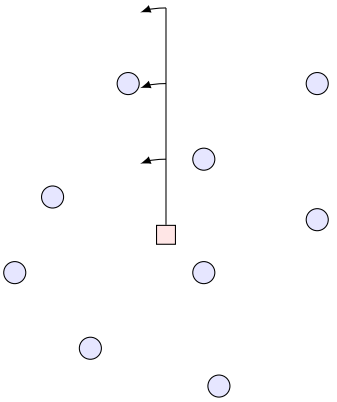
Another type of construction heuristics are the two phase methods. The two phases described in these methods are

1. Clustering, where the customers are divided into subsets, that will be served by one route.

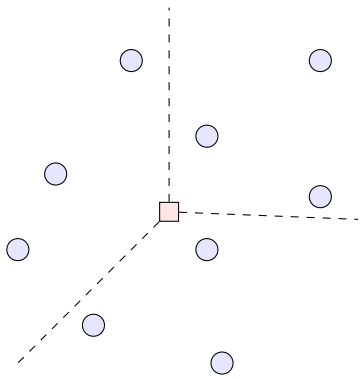
2. Routing, where the sequence of the customers on the route is established.

Two phase methods can be divided into cluster cluster-first, route-second and route-first, cluster-second methods.

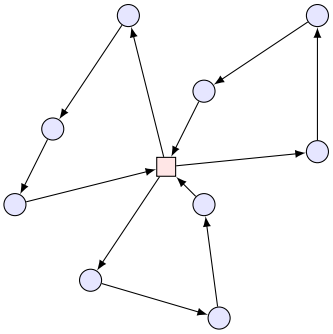
An example of a *cluster-first, route-second* method is the sweep algorithm, Gillett and Miller [31]. This algorithm starts with a random customer and assigns customers to the subset by increasing the angle around the depot. When it is no longer feasible to add new customers to the subset, for example because of the capacity constraints, a new subset is initialized. When all customers are assigned to a subset, the vehicle routes can be solved with a TSP algorithm. The sweep algorithm also has the quality of being easy to understand and to implement, but is not as fast as CWS. Furthermore, CWS usually has a better performance with respect to the accuracy, see Cordeau et al. [20]. The process of the sweep algorithm is shown in Figure 2.5



(a) Starting point sweeping.



(b) Clustered customers.



(c) Vehicle routes.

Figure 2.5: Example of the sweep algorithm with vehicles with a capacity of 3 customer requests.

A *route-first, cluster-second* method is described by Beasley [10]. A giant tour that visits all customers is constructed with a TSP algorithm and afterwards the tour is cut into smaller feasible vehicle routes with a splitting procedure using an auxiliary graph. This method did not have very good results until Prins re-evaluated the split procedure and combined it with metaheuristics in Prins [63] and Prins [64].

2.3. Local Search

Local search heuristics are applied to improve an initial solution. Starting from the initial solution S, local search explores *neighbourhood* solutions $S' \in N(S)$ which are obtained by simple modifications to the solution S, such as customer movements. The moves, $S \rightarrow S'$, can be divided into

1. Intraroute moves, which operate on a single route at a time, also known from the TSP.
2. Interroute moves, which consider multiple routes simultaneously.

There are two types of strategies for local search:

1. The *first-improvement strategy*, where a the first solution $S' \in N(S)$ for which $C(S') < C(S)$ is accepted, where $C(S)$ is the cost of solution S.
2. The *best-improvement strategy*, where all solutions $S' \in N(S)$ are evaluated and the best one S^* gets accepted if $C(S^*) < C(S)$.

One can imagine that choosing a good neighbourhood becomes more difficult with more constraints, because moves can lead to infeasible solutions more often. For this reason more complex moves and smart feasibility tests have been developed trying to maintain speed, such as precomputing maximum delay for VRPs with *time windows*, by Savelsbergh [69]. A review on local search neigbourhoods can be found in Funke et al. [29].

In this section some of the classical moves for local search neigbourhoods are explained.

Node relocation

This comes down to removing a customer from its current location and reinserting it at another. This is called the *insert* neighbourhood and is most commonly used. This move can be applied on a single route or between multiple routes, see Figure 2.6.

(a) Before node relocation.

(b) After node relocation.

Figure 2.6: Relocation of node c. These node strings can be from the same route or from two different routes.

Node exchange

Two customers are swapped from positions, which provides the *swap* neighbourhood. Swap can also be applied on a single route or between multiple routes as shown in Figure 2.7. Both swap and insert neighbourhoods are of size $O(n^2)$.

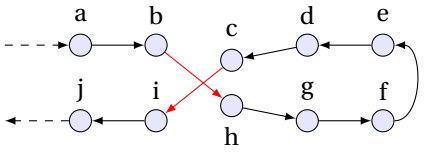
(a) Before swap.

(b) After swap.

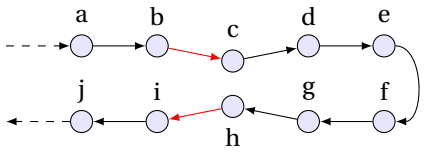
Figure 2.7: Swapping nodes x and y. The node strings can be from the same route or from two different routes.

λ-opt

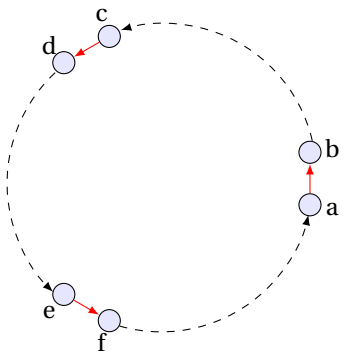
The λ-opt neighbourhood, by Lin and Kernighan [51], is obtained by removing and reinserting λ arcs on the same route. Testing all λ-opt moves costs $O(n^\lambda)$ time for n customers. For this reason the most common are 2-opt and 3-opt, shown in Figure 2.8.



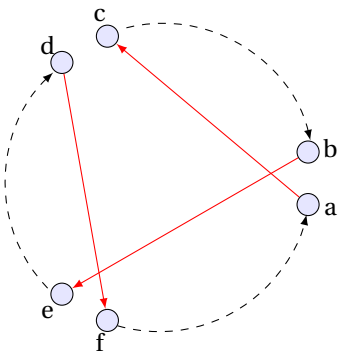
(a) Before 2-opt. Edges (b,h) and (c,i) will be removed.



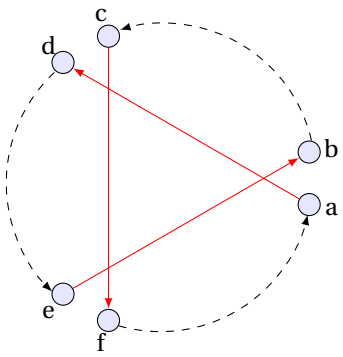
(b) After 2-opt. Edges (b,c) and (i,h) are constructed. Notice that the string h,g,f,e,d,c is reversed after 2-opt is applied.



(c) Before 3-opt. Edges (a,b), (c,d) and (e,f) will be removed.



(d) Possible outcome after 3-opt. Notice that some of the dotted arcs are reversed after 3 opt is applied.

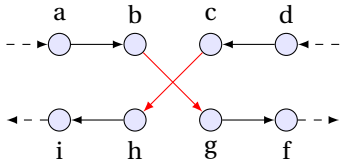


(e) Another possibility for the outcome after 3-opt. Note that there are 8 possible outcomes.

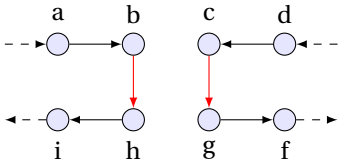
Figure 2.8: Applying 2 and 3-opt.

2-opt*

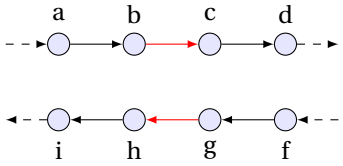
The 2-opt* neighbourhood, Potvin and Rousseau [62], is similar to 2-opt, described above, but applied on two different routes. This neighbourhood contains $O(n^2)$ solutions. As shown in Figure 2.9 this move has two possible reconstruction options.



(a) Before 2-opt*. Edges (b,g) and (c,h) will be removed.



(b) Possible outcome after 2-opt*. Edges (b,h) and (c,g) are constructed.

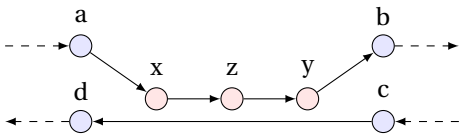


(c) Possible outcome after 2-opt*. Edges (b,c) and (h,g) are constructed.

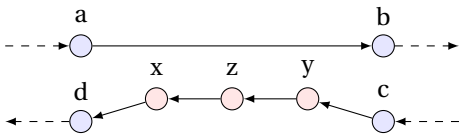
Figure 2.9: For 2-opt* there are two reconstruction options.

Or-opt

OR-opt, Or [56], relocates a string of at most λ customers. Note that this neighbourhood contains a subset of 3-opt moves. This neighbourhood can be searched in $O(\lambda n^2)$ time. $\lambda = 3$ is most commonly used.



(a) Before Or-opt.

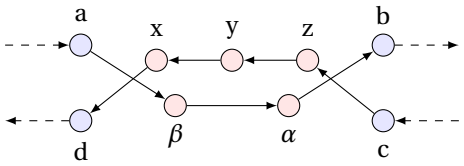


(b) After node Or-opt, with $\lambda = 3$.

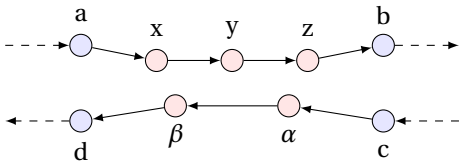
Figure 2.10: Applying Or-Opt, for $\lambda = 3$.

λ-interchange

The λ-interchange neighbourhood, Osman [57], exchanges two customer subsets of at most size λ. The subsets can have different sizes (one being potentially empty), which means this neighbourhood includes the insert, swap, λ-opt and Or-opt neighbourhoods.



(a) Before λ-interchange, for two strings of different size.



(b) After λ-interchange.

Figure 2.11: Applying λ-interchange, for $\lambda = 3$.

2.3.1. Penalized objective

When local search is applied to very constrained problems it can be helpful to widen the solution space. This can be done using a penalized objective. In this case complicating constraints are relaxed and the solution value $C(x)$ of solution x will be replaced with a penalized objective, e.g.

$$C'(x) = C(x) + \alpha Q(x) + \beta T(x) \tag{2.30}$$

where $Q(x)$ is the total capacity violation and $T(x)$ the total time window violation. α and β are coefficients that can be used to fine-tune the weights of the violation.

2.4. Metaheuristics

Metaheuristics are more generic solution schemes developed to escape from local optima. Adopting a metaheuristic to the practical problems at hand requires much less work than developing a specialized heuristic from scratch. Moreover metaheursistics can provide near-optimal solutions in a feasible time, when implemented in a good way. For this reason metaheuristics are according to most literature the most appropriate way to deal with real life VRPs.

There are four types of metaheuristics distinguished in this section, metaheuristics based on:

1. Local search, in Section 2.4.1,
2. Population search, in Section 2.4.2,
3. Learning mechanisms, in Section 2.4.3,
4. Hybrid heuristics, in Section 2.4.4.

2.4.1. Local search

Metaheuristics based on local search start with an initial solution that, for example, can be constructed by one of the construction heuristics from Section 2.2 and then define in each iteration the neighbourhood of the current solution to go from this solution to another. The success of these metaheuristics depends on the defined neighbourhoods. Local search algorithms are rarely implemented in their basic version. The best implementations are in hybrid methods. In this section some of the classical metaheuristics based on local search are described; simulated annealing, tabu search, variable neighbourhood search, iterated local search, adaptive large neighbourhood search and guided local search.

Simulated Annealing

Probably the first metaheuristic of this type used for vehicle routing is *Simulated Annealing* (SA), by Kirkpatrick et al. [43]. It tries to escape local optima by accepting neighbourhood solutions with a deterioration of the objective value, with a probability $e^{-\Delta c/\theta_t}$. Where θ_t is a *temperature* parameter, which is decreased continuously according to a cooling scheme. This way the probability of accepting a deterioration becomes smaller during the search process.

In contrast with many other metaheuristics, for SA it is not necessary to start with a good initial solution, since high temperatures in the beginning of the algorithm would give it high probability that it will escape from the initial solution. The most successful implementation of SA for the CVRP is by Osman [57]. There are a few other implementations on extensions of the CVRP, e.g. truck and trailer routing problem and time windows. SA produces good quality solutions, but it is not one of the most efficient methods and was not competitive with the tabu search implementations that were developed at the same time, with respect to accuracy and speed.

An example of a deterministic variant of Simulated Annealing is the *record-to-record* method from Li et al. [50], specific for very large instances. Instead of accepting a worse solution with a certain probability, depending on temperature θ_t, the record-to-record method accepts the solution x if it is not worse than a predefined deviation from the current best known solution x^*, thus if $f(x) \leq \theta f(x^*)$.

Tabu search

Tabu search (TS), Glover [34],[35], is one of the most popular researched metaheuristics for the VRP. For a long time it has been the most effective metaheuristic for VRPs available. The basic idea of TS is as follows. At each iteration it moves from solution S to the best solution $S' \in N(S)$, as in Figure 2.12. Tabu search avoids local optima by continuing the local search even when it has reached a local optimum. To avoid cycling, solutions possessing some attributes of the previous solution S are declared *tabu* (forbidden) for a certain amount of iterations, the red vertices in Figure 2.12. A solution that contains some *aspiration criteria* (e.g. attractive / best known solution), will still be accepted, even though it is tabu. Important elements of tabu search are intensification, and diversification, which can be done using penalties. An overview of some of the most effective tabu search methods can be found in Cordeau et al. [20] and Cordeau and Laporte [18]. They both compare the TS methods and evaluate them on their quality.

Some of the best TS methods are Taburoute, by Gendreau et al. [30], Taillard TS, by Taillard [75], Adaptive Memory Procedure (AMP, can also be viewed as a form of population search), by Rochat and Taillard [67], Unified TS (UTS), by Cordeau et al. [19], and Granular Tabu Search (GTS), by Toth and Vigo [76]. These methods all score very good on flexibility and accuracy, but on speed and simplicity there are still some points to achieve.

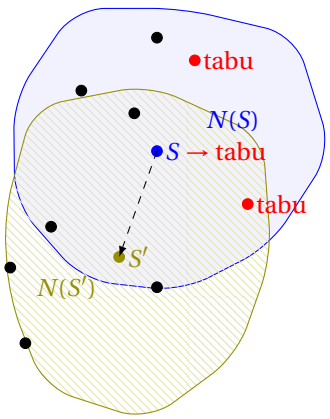


Figure 2.12: Visualisation of tabu search. The current solution S and its neighbourhood $N(S)$ are presented blue. Some of the solutions in the neighbourhood are tabu (red). It is possible to move from solution S to solution S' which results in a new solution area $N(S')$ in which solution S becomes tabu.

Variable neighbourhood search

Variable neighbourhood search (VNS) is introduced by Mladenović and Hansen [54]. The idea is to switch from one neighbourhood N_i to another N_{i+1} when a local optimum has been reached in the current N_i.

When a better solution is found in N_{i+1}, the algorithm restarts with its search in the first (usually smallest) neighbourhood N_1, as shown in Figure 2.13. For large instances for which local search becomes very expensive a stochastic manner can be used to select random solutions from a neighbourhood $N_i(S)$. Although other more complex metaheuristics are often better on the other points, the simplicity and speed of VNS makes the method very attractive. In Kytöjoki et al. [44] VNS is applied to the CVRP for very large instances.

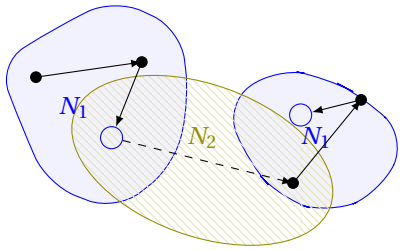


Figure 2.13: Visualisation of variable neighbourhood search. It starts moving from one solution to another in neighbourhood N_1, until a local optimum in that neighbourhood is reached. Then the algorithm moves to another neighbourhood N_2 that has found a better solution. The algorithm returns back to the first neighbourhood.

Iterated local search

One of the most efficient metaheuristics nowadays is *iterated local search* (ILS), introduced by Baum [9] under the name iterated descent. This method was put forward a few years before variable neighbourhood search and is in fact very similar. When local search ends up in a local optimum it changes the search space with a *perturbation*. This perturbation can be a random move in a different neighbourhood from the one used in local search. Choosing the right strength for the perturbation is crucial for this method. For a recent review see Lourenço et al. [52]. ILS is often used in hybrid methods. An example of a simple and efficient ILS method is provided by Prins [64].

(Adaptive) large neighbourhood search

Adaptive large neighbourhood search (ALNS) proposed by Pisinger and Ropke [60] is also similar to VNS. It is an extension of *large neighbourhood search* (LNS) Shaw [72]. This method exists of a set of (large) neighbourhoods N with destruction, $N^- \in N$, and repair operators, $N^+ \in N$. The size of the destructed elements impacts the size of the neighbourhood. When a neighbourhood has a desired influence on the solution the probability that this neighbourhood is chosen becomes larger. ALNS works especially good for very constraint problems that have trouble escaping a local minimum. Unfortunately the number of neighbourhoods for destruction and repair are at the expense of the simplicity. Furthermore, this method will probably lose a lot of speed with large problems. Another variant of LNS is *very large scale neighbourhood search* (VLSN) where the neighbourhoods, often defined by classical moves, have an exponential size. This is solved by building an auxilary graph, with which the best move can be determined by solving a network flow problem.

Guided local search

In *guided local search* (GLS), by Voudouris and Tsang [82], aspects from both ILS and tabu search are visible. It uses a pertubation principle, with a penalizing feature to escape local optima. An application to the VRPTW can be found in Kilby et al. [42]

Greedy randomized adaptive search procedure

A method that has successfully been used in hybrid heuristics together with ILS and ELS, in Prins [64], is *Greedy randomized adaptive search procedure*: GRASP. The GRASP is developed by Feo and Resende [25] in 1989, and consists of two stages.

1. Build a solution using a randomized greedy heuristic.
2. Apply local search on the obtained solution

An example of a greedy heuristic that can be used is the CWS heuristic discussed earlier. Instead of deterministically always choosing the best savings, the idea of GRASP is to make a candidate list and choose an item from this list. There are two policies to implement the randomness:

1. Make a random candidate list and select the best item from this list.
2. Make a candidate list of the best savings and randomly select an item form this list.

The difficulty is to decide the randomness of the GRASP. Providing a completely random solution to the local search leads to bad solutions, but some randomness is needed to diversify the search. In policy 1, a larger candidate list makes the method less random, while in policy 2 choosing a larger candidate list makes the method more random. Although GRASP can produce good quality solutions it is not very efficient and the method is very independent of the number of iterations.

2.4.2. Population search

The metaheuristics in this category use a set of initial solutions, the solution *population*, and combines items of the population to generate new solutions.

To create an initial solution population one can try to construct completely random solutions or use a heuristic such as Sweep or CWS with some randomness. In this section the metaheuristics adoptive memory procedure, genetic algorithms, evolutionairy local search, scatter search and path relinking are described.

Adoptive memory procedure

The *adoptive memory procedure* (AMP) by Rochat and Taillard [67], also named in the previous section in the category tabu search, does also belong in the category population search based heuristics. AMP uses a pool of good solutions to recombine new solutions. The worst solutions are replaced by better ones. Positive about this method is that it is very flexible and accurate. However the use of an adoptive memory requires extra computation time, and makes the coding a bit more complicated.

Genetic algorithms

Under *genetic algorithms* (GAs) three variants are subsumed; the basic variant by Holland [40] (GA), the *memetic algorithm* by Moscato et al. [55] (MA), and a *memetic algorithm with population management* by Sörensen and Sevaux [73] (MA|PM). The concept is

1. Select two parent solutions from the population, with a preference for "fitter" parents,
2. Let them reproduce by applying crossover,
3. And finally apply some mutations to ensure diversity in the population.

To get an idea of the reproduction an example of a much used crossover concept is given in Figure 2.14. Afterwards the children will replace some of the worse solutions in the population. In the reproduction step the parents are combined using a crossover operator. The basic version of GA has not been very successful in combinatorial optimization and therefore MA also includes an improvement step after the reproduction step. Here local search is applied to the children before undergoing mutation. On top of this improvement MA|PM also improves the replacement step. Instead of using all new solutions, only solutions that are significantly different from the ones already in the population are accepted. A way of measuring the distance between solutions is the Hamming distance, which counts the number of positions for which the customers are different. The first algorithm that was able to improve upon the results of tabu search methods was the MA by Prins [63]. Good examples of recent genetic algorithms, able to solve a large class of VRPs, are Vidal et al. [80] and Vidal et al. [81]. An overview of GAs upto 2009 can be found in Potvin [61].

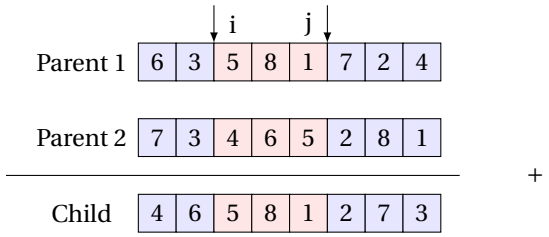


Figure 2.14: Example of OX crossover. Suppose a VRP solution is represented by strings of customer routes sequenced. Randomly select two positions i, j with $i < j$. All customers between position i and j are copied from Parent 1 to the Child. Check which customers from position j to $j-1$ in Parent 2 are not represented yet in the sequence of the Child and place them starting at $j+1$ and ending at $i-1$.

Evolutionary local search

Evolutionary local search (ELS), by Wolf and Merz [83], is a form of ILS where in each iteration pertubation and local search is applied to create a population of p child solutions. The current solution S is then replaced by the best child. ELS is applied by Prins [64], together with the “route-first, cluster-second" concept from Section 2.2.2.

Scatter search

Scatter search (SS), by Glover [33], is very similar to MA. It also uses crossover operators and local search, but applied to a much smaller set containing only good and divers solutions. SS starts with a pool of well-scattered solutions to ensure diversity. This can be measured with a distance measure as described for MA|PM. Local search is applied to the population and crossover is applied to all pairs of a subset of the population on which local search is again applied. The best children from this crossover process are saved and the process is repeated. Unfortunately SS is very time consuming, but there are a few successful implementations, e.g. Russell and Chiang [68].

Path relinking

Path relinking (PR) was introduced, by Glover et al. [32] Glover [36], to intensify a TS or SS. PR considers a small subset of good solutions and generates new solutions from each pair of solutions, U and V, where U is the initiating solution and V is the guiding solution. U is guided toward V by incorporating attributes from V. The intermediate solutions that arise are checked to see if they improve the best current solution. This method is, as it is intended, often used within other metaheuristics. In the hybrid methods where it has been used it provides good results, for example see Ho and Gendreau [39].

2.4.3. Learning mechanisms

Methods in the category learning mechanisms construct a solution using feedback from solutions already found. Examples are ant colony optimization and neural networks.

Ant colony optimization

Ant colony optimization (ACO), see Dorigo et al. [24], is another bioinspired algorithms inspired by the foraging behaviour of ants. Ants leave tracks of pheromone when searching for food, to mark a favourable path. ACO is an iterative algorithm where in each iteration artificial ants are used to construct solutions favouring partial solutions, in the VRP case edges, with more pheromones. At the end of the iteration the pheromones are updated based on the quality of the solution. This way edges appearing in good solutions have a bigger probability of getting chosen. A good example of ACO approaches applied to the CVRP is Reimann et al. [66].

Particle Swarm optimization

Other swarm inspired methods are *particle swarm optimization* (PSO) and the *artificial bee colony* (ABC) method. Those methods are less examined and satisfactory (unless used in hybrid methods) compared to other metaheuristics.

Neural networks

Other methods based on learning mechanisms are *neural networks*, inspired by neurons in the brain. Like in ACO, solutions are constructed through a feedback mechanism. None of the known neural networks could compete with other metaheuristics, when comparing the computation time and the accuracy of the methods.

2.4.4. Hybrid heuristics

The best metaheuristics often combine elements from different metaheuristic principles. This way the strength of different heuristics can be taken into advantage. This however can be a very difficult task. For the combined metaheuristic to succeed the method should be accurate, fast, simple and flexible. Thus, choosing the best elements of the methods is the key to success.

There are two types of hybrid heuristics; meta-meta hybridizations and matheuristics.

There is a very large class of hybrid methods proposed for all kinds of vehicle routing problems. This section will not go in to details about all possible methods, but refers the reader to Prodhon and Prins [65].

Meta-meta hybridizations

Hybrid heuristics that combine several concepts from different metaheurisics are sometimes called *meta-meta hybridizations*. These can be combined by replacing elements in a metaheuristic by elements from another metaheuristic, by implementing complete metaheuristics into another, by using the metaheuristics in a sequential way or by splitting the main problem into subproblems that are solved by different metaheuristics.

As mentioned before, local search heuristics are often combined with other methods and are therefore usually part of a hybrid metaheuristic. In the previous sections a few have already been named; (SA+Tabu) Osman [57], (GRASP +ILS) Prins [64] and (ILS+VND) Vidal et al. [81].

Matheuristics

Hybrid heuristics that combine exact algorithms and metaheuristics are named *matheuristics*. An example of a recent matheuristic is a combination of ILS and set partitioning by Subramanian et al. [74]. For a detailed survey see Archetti and Speranza [2].

2.5. Qualifying VRP solving methods

Since there are so many different solving methods it is easy to get lost. To decide when to use which solving method for a VRP, it is useful to know the qualities of the method.

Cordeau et al. [20] gives the following *guiding criteria* to qualify the heuristics for practical use:

1. Accuracy. This measures at the gap between the solution value and the optimal value. Usually this is measured with the best known solution, since optimal values are not always available.

2. Speed. The importance of speed depends on the use of the method. This of course is different for real-time planning compared to daily planning. Unfortunately computation times can be hard to compare, because of the use of different computers and experiments and because some heuristics are based on parallel computing.

3. Simplicity. Some of the solving methods can become so complicated to understand and to code, they are hardly used. Simpler codes, that are easy to adopt, have a better chance of working. This is of main importance when different programmers are working on the code.

4. Flexibility. In real-life the problems, the CVRP will often be extended with side constraints, such as time windows and heterogeneous vehicles. Therefore it is important that it is not to difficult to adopt the solving method to deal with these constraints.

Based on these criteria one can decide which type of method would be most useful for their case. For example, for a case with a lot of customers (n is large) that needs fast results, speed is important. For smaller cases where saving costs is most important accuracy is more important while speed is less important. And for cases with lots of constraints, flexibility is important. Furthermore the simplicity is mainly important for the programmers that have to build the method. Simplicity makes it easier when writing and maintaining the code. Thus this is important when the code can not be too complicated, because a lot of maintenance and extension is expected.

The above described criteria will be used to evaluate the different methods named in the previous sections of this chapter.

2.5.1. Accuracy

When accuracy is most important for choosing a solving method, an exact method such as the branch-cut-and-price (BCP) method (Pecin et al. [59]) is the most logical choice. When a problem is solved with an exact method one can be sure the answer is an optimal solution. Therefore these methods are also very helpful as a reference for heuristic methods, to see how accurate they are.

Unfortunately the size and the complexity of most real-life problems and the time available to solve them does not allow for exact methods. Therefore usually heuristics are used to solve these problems. In the category heuristics there are a few very accurate hybrid genetic search metaheuristics (HGS) which can solve problems up to 1000 of customers such as Vidal et al. [80].

2.5.2. Speed

An easy and fast way to get a feasible solution for a VRP is by using a simple heuristic such as the Clarke and Wright savings method (CWS) (Clarke and Wright [17]). Nevertheless, this method is not very accurate compared to metaheuristics or exact methods developed and this method is very inflexible, which means it can not easily take extra constraints into account without deteriorating the solution. Using an exact method to solve a problem is certainly not a good choice when time is an issue, as the results of Pecin et al. [59] and Uchoa et al. [79] show that even the current state of the art of exact methods can sometimes take more than a day to solve a problem. For this reason metaheuristics usually give a more satisfactory result when speed is important.

An example of a very fast metaheuristic is the evolutionary local search algorithm of Mester and Bräysy [53]. This method works especially well for large problems.

2.5.3. Simplicity

Simplicity is very hard to qualify, because it is not really possible to validate with some value. Moreover it can be seen more as an opinion rather than a fact. However, if a method can easily be understand and programmed by a reasonable skilled researcher the method can be seen as a relatively simple method, and if it takes a lot of time, effort and extra knowledge for most researches it can be seen as a complicated method.

Another great quality of the CWS is the simplicity. The concept is very easy to understand and the implementation is also not too difficult. Though if one also wishes for a more accurate and flexible method, metaheuristics again are a better option. To my opinion the concept of variable neighbourhood search (VNS) or iterated local search (ILS) are easy to understand. In addition, simplicity is also named as one of the main features of the ILS described in Prins [64].

Most exact methods become complicated very quickly. To my opinion the two-index formulation with a simple version of the branch-and-bound method (discussed in Section 2.1.3) is very intuitive and the implementation is not too complicated. However, this method is very limited. When one wants to solve larger, or more complicated problems, or the computation time needs to be improved, the exact method will rapidly get more difficult. For these reasons I would not classify the most exact methods high in the category of simplicity.

2.5.4. Flexibility

Flexibility has become more and more important for the solution methods for the VRP. This is because in practice most VRPs are *rich* VRPs (VRPs with many extra constraints such as time windows etc.). However, qualifying the flexibility of a method is just as hard as qualifying the simplicity. Since flexibility does not only concerns the alteration of the accuracy and the speed of the solution when extending the method to solve multiple VRPs, but also the difficulty of extending a method to solve multiple VRPs. However, will mention some methods that were specifically developed to be able to deal with multiple attributes of the VRP.

Some of the solving methods for the VRP are so problem specific that adopting them for another VRP can be very complex or even impossible. Specialized solving methods can be useful for a company dealing with one specific type of VRP, but even then there are sometimes still some extensions to be made to completely fit to the real-life problem. Furthermore, many companies, like EVO-it, deal with a lot of different type of multi-attribute VRPs. This was a motivation for the development of more general methods such as the Unified Hybrid Genetic Search metaheuristic (UHGS), by Vidal et al. [81], the ALNS metaheuristic, by Pisinger and Ropke [60] and the ILS-SP matheuristic, by Subramanian et al. [74]. Recently, even some unified exact methods have been developed such as the set partitioning based method of Baldacci and Mingozzi [5].

However, extending a metaheuristic to solve problems with attributes that were not originally considered will usually give less complications than extending an exact method to solve these problems.

3

Complications in real-life instances at EVO-it

EVO-it has around 140 customers that use Ritplan on a daily basis. These customers are usually dealing with multi-attribute VRPs, thus VRPs including multiple attributes, for example time-windows ánd heterogeneous vehicles ánd multiple depots. To get an idea of the type of problems of Ritplan users and how Ritplan users enter their data in Ritplan, an assessment has been made using the information of a set of 26 customers. This will be referred to as the selected data. The attributes that appeared during this research will be elaborated in this chapter. First, the general attributes are described in Section 3.1. Then, the attributes with respect to the orders and the vehicles are described in Section 3.2 and Section 3.3, respectively. Finally, the attributes that are not covered by the current model of RitOpt are elaborated in Section 3.4.

The resulting occurrence of each type of attribute is summarized in Section 3.5.

3.1. General attributes

This section will describe the general attributes, which concerns the type of planning: period or day, and if the solution should be based on a given route or can be completely optimized.

3.1.1. Period or day planning

The users of Ritplan can choose to make a new route schedule each day or a schedule for a certain period. The latter option can be used for example if a company has orders that need to be delivered somewhere in a week, but not on a specific day.

A schedule for one day is a more standard problem, which can be solved by many of the methods in the literature, while a period planning may need more complicated solving methods.

According to the data gathered from Ritplan only 8% of the evaluated instances use a period planning.

3.1.2. Complete or partial freedom of variables

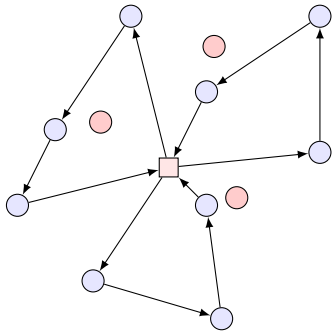


Figure 3.1: A base route is given for the blue customer vertices, but a few extra orders, given in red, have emerged. New routes needs to be constructed.

Some of the customers of EVO-it use a *base route*. A base route can be seen as a route that is already constructed for a (not necessarily strict) subset of the orders. This base route can be used as an initial solution, that can still be completely optimized. Alternatively the user can command the program to keep some of the features of the initial solution. For example, keep the orders on the current vehicle, or maybe even on the current position in the route. When the latter options are chosen there is only a freedom in placing the orders that have not been scheduled yet. A base route can be used for example when some of the orders were added after a schedule had already been made.

Around 31% of the evaluated instances make use of a base route.

3.2. Order attributes

As explained in Section 1.2, the data that a Ritplan user provides to the program consists of a lists of addresses and orders. However, in the formulation of the VRP it is common to refer to a set of customers, which are locations with a certain demand. As such, the input to the Ritplan program (addresses and orders), does not completely match with the definition of a customer as defined for the VRP. The addresses are the complete set of locations, thus also contain depots and customers that do not have an order. Orders are associated with an “from" and a “to" address and define a certain demand.

Comparing the definitions of Ritplan with the literature shows that the order demand combined with the “to" address is similar to a customer as defined for the VRP in the literature.

This section will elaborate on the attributes considering the orders, which are:

- the number of orders,
- the type of an order: delivery, pick-up, or pick-up and delivery,
- if an order is preloaded, or afterwards unloaded,
- time windows,
- allowance of unscheduled orders and priority,
- rayons
- and vehicle restrictions.

3.2.1. Number of orders

Counting the number of addresses does not necessary correspond to the number of customers, because some of the addresses that are provided might not be used. Moreover, some of the addresses are from the depots or from drivers of the vehicles, thus have nothing to do with the number of customers. Furthermore, the amount of orders also does not necessarily agree with the number of customers, because some customers might have several orders. Nevertheless, several orders to one address (thus for one customer) can be seen as multiple customers with distance 0. With this reasoning, the number of orders is used as an appropriate way to measure the size of the problems.

Figure 3.2 represents the frequency in which the problem sizes occurred in the selected data.

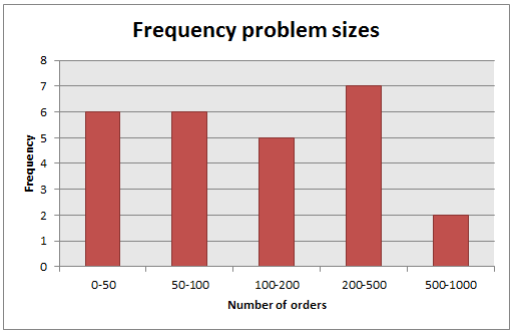


Figure 3.2: Histogram of the frequency of the problem sizes for the 26 instances of EVO-it that were considered in this thesis.

Remark: Some of the Ritplan users might have a total of N orders to schedule and p depots from where these orders have to be delivered. Suppose that each order is already linked to one specific depot. This means the order can only be served by that depot. That is why the problem can be split into solving p smaller VRPs, one for each depot. The problem sizes used for Figure 3.2 are based on the problem size after these splits.

From the evaluated data the following conclusions can be drawn. A bit more than 40% of the problems has less than 100 orders and therefore might be interesting to solve with an exact algorithm. Unfortunately 67% (8/12) of these problems contain extra attributes such as time windows and heterogeneous vehicles (vehicles with different constraints). These extensions can make the problem significantly more difficult, certainly since finding a feasible solution with time windows is already NP-hard. Next to time windows and heterogeneous vehicles there are many other attributes that can make the problem harder to solve, such as multiple depots, different start and end points and orders with priority.

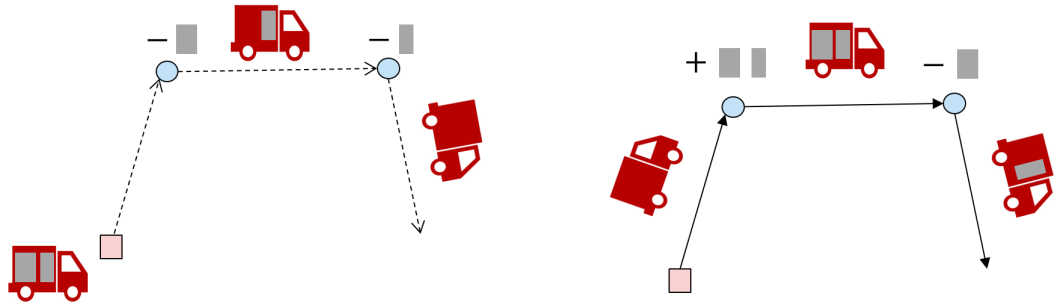
Furthermore, only 7% of the evaluated instances had more than 500 customers. For the most heuristic solvers, solving problems up to 1000 of customers is not a problem, but it will decrease the speed of the solving process and/or deteriorate the accuracy of the solution.

3.2.2. Delivery, pick-up or pick-up and delivery

The orders of Ritplan can be of three different types:

- Delivery, represented in Figure 3.3a, this type of order goes from a depot to the customer.
- Pick-up, this type of order goes from the customer to a depot and is thus the opposite of a delivery order.
- Pick-up and delivery, represented in Figure 3.3b, this type of order has to be picked-up at a customer and delivered at another customer.

Similar to the delivery orders, is the depot-to-depot order. Hence, it is not considered as a fourth type.



(a) Delivery orders: The orders are picked up at the depot and delivered to the customers.

(b) Pick-up and delivery orders: The orders are picked up at one customer and delivered to another customer.

Figure 3.3: Visualisation of different order types; delivery and pick-up and delivery. The orders are presented in grey, the depot is represented as a red square and the customers as round circles

The difficulty for a pick-up and delivery order is that the customers have to be visited in a specific order. One cannot deliver what is not picked-up yet , which makes a lot of solutions infeasible. Another difficulty is that for example the delivery locations of orders can be very close, but the pick-up locations can be in opposite directions, or the other way around. This could mean that some areas have to be visited multiple times, as represented in Figure 3.4, which seems very inefficient.

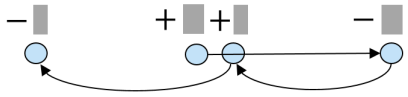


Figure 3.4: Problem with Pick-up and delivery. Some areas have to be visited multiple times, which results in inefficient routes.

These problems make the optimization process for pick-up and delivery more difficult. For this reason RitOpt has an extended algorithm for these kind of problems. In the literature these problems are called *pickup and delivery (vehicle routing) problems* (PDVRPs or PDPs), discussed by Savelsbergh and Sol [70] and Berbeglia et al. [11]. When the orders consider people instead of goods it is called the *dial-a-ride problem* (DARP). From the evaluated instances 19% deal with pick-up and delivery orders.

The problems characterized by having only delivery or only pick-up orders are the easiest to solve. In total 54% of the evaluated problems are of this type. The remainder of the problems are more complicated, because they have a combination of delivery, pick-up, or pick-up and delivery orders.

3.2.3. Preloaded or afterwards unloaded

The orders in Ritplan can get the status *preloaded* (Dutch: “voorladen”) or *afterwards unloaded* (Dutch “nalossen”). The first term is used for delivery orders and the latter for pick-up orders.

If an order is *preloaded* this implies that the vehicle does not have to go to a specific depot to pick-up the order, because the order is already on the vehicle. If all orders are preloaded this means that the vehicle delivering these orders can start anywhere. Usually this is from one of the depots, or from the home address of the driver.

Afterwards unloaded is very similar, orders that have been picked-up do not have to go to a specific depot at the end. This means a vehicle with only orders of this type can end anywhere.

Important to mention is that all orders have an address “from" and an address “to". Without preloading or afterwards unloading the orders have to be picked-up at the “from” address and delivered at the “to” address. This means that, in case of multiple depots, the orders are already linked to a certain depot and the problem can be split into multiple smaller VRPs, one for each depot. A side issue of preloading or afterwards unloading is that the orders are no longer attached to a specific the depot, thus can start/end at any depot, and therefore the problem can not be split into subproblems. In this case the problem can be considered as a *multiple-depot VRP* (MDVRP), Figure 3.5.

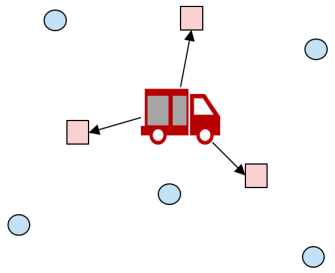


Figure 3.5: When all orders are preloaded the problem becomes equal to a multi-depot problem, which means the vehicle can start from multiple startpositions.

3.2.4. Time windows

For vehicle routing problems with time windows each order i must be delivered or picked-up within a given time window $[a_i, b_i]$, or within one of several time windows $[a_{i_1}, b_{i_1}], ..., [a_{i_k}, b_{i_k}]$.

In practice nearly all VRPs use time windows, because there is always a certain time period in which the orders need to be delivered, e.g. a day. However, often time windows are not an obstacle when scheduling the vehicle route. Other constraints, like capacity constraints are much more tight and in that way, ensure

that each order is also delivered within the period of one working day. Therefore not all VRPs that where researched at EVO-it will be categorized underneath *VRP with time windows* (VRPTW). Problems for which a subset of orders have to be delivered around a certain time of the day will be considered as VRPTW. Problems for which all orders can be delivered at any time of the day will not be considered as a VRPTW.

From the instances considered in the data, 73% contian time windows for a subset of orders. The VRPTW is not only NP-hard, but even finding a feasible solution to the VRPTW is NP-complete, Savelsbergh [69]. For this reason research has mainly focused on heuristics for VRPTWs, for an extensive survey see Bräysy and Gendreau [12], Bräysy and Gendreau [13].

The sizes of the time windows for the considered problems vary between 0 minutes and a whole day.

In the first case, the time to deliver is already exactly defined so the only decisions left are which vehicle will deliver this order and together with which other orders. Restricting the order to a specific time gives very bad results, so usually customers of EVO-it are told not to create such a time window. The average smallest time window used in an instance is around 1,5 hours.

When the time window is 24 hours, the time window is not really a bottleneck. An explanation for why these time windows are used is that some users probably use time windows for each order, thus also for orders that can be delivered the entire day. The average largest time window is around 12 hours.

Furthermore, the user can allow a delay, which is often set to an allowance of 30 minutes. This is implemented with a penalty construction, such that model tries to avoid exceeding the time window.

3.2.5. Unscheduled orders and priority

Some Ritplan users enter more orders into the program than it is possible to schedule for one day, whilst satisfying all constraints. To deal with this, RitOpt uses penalties in the objective function, and does not use constraints to make sure that each order is scheduled exactly once. Thus when it is not possible to schedule all orders, RitOpt can give a solution that has excluded some of the orders.

Sometimes users have a subset of orders for which they want certainty that these will be scheduled. These orders can get the label “priority order". RitOpt will try to schedule these before any others. When these orders are scheduled they become “fixed". This means that for these orders it is not allowed to get unscheduled, when a new solution is constructed while the local search process of RitOpt is looking for neighbourhood solutions.

From the evaluated data, 15% of the problems used priority on some of the orders.

3.2.6. Rayons

The orders in RitPlan can be linked to *rayons*. These rayons are defined by the user and can for example be a region, e.g. east-west-south-north, or they can be used because certain orders can not be in the same truck at the same time, e.g. certain chemicals. Orders can be linked to more than one rayon, which means that rayons can overlap. In 48% of the evaluated problems rayons were used on the orders, but often the rayons were not used for the vehicles or depots. This means that these settings would not have any influence on the solving process. So it was difficult to see how many of the problems were actually influenced by this constraint.

3.2.7. Vehicle restrictions

In 42% of the problems there are orders that are restricted to a specific subset of vehicles. In a mathematical model this is probably similar to the rayon constraint. In the literature these problems are called *site-dependent vehicle routing problem* (SDVRPs).

3.3. Vehicle attributes

This section will continue with the problem attributes with respect to the vehicles. This concerns:

- the number of vehicles,
- the capacity of the vehicles,
- time windows and route length,
- the costs of the vehicles,
- the possibility of multiple trips,

- heterogeneous vehicles,
- start- and endpoints,
- multidepots
- and breaks.

3.3.1. Number of vehicles

For the evaluated instances, the number of *used* vehicles lies between 1 and 50, and has an average of 14 used vehicles. Often there are (much) more vehicles available than there used.

What might be unexpected, is that a bigger number of available vehicles decreases the difficulty of the problem for RitOpt, which can be explained as follows. The number of used vehicles in Ritplan is always a *variable*. This means that if there are 10 vehicles available, it is possible to use $1 \leq m \leq 10$ vehicles. The optimization program RitOpt will always try to use an optimal number of vehicles (which is not necessarily minimal). RitOpt uses simulated annealing with different local search neigbourhoods. The reason that optimization becomes less difficult when the number of available vehicles is large is because there are more feasible moves, because the solution space is larger. This makes it easier to move from one solution towards a better one.

The average number of orders handled by one vehicle for the evaluated problems lies around 22. The highest average number of orders handled by one a vehicles is around 147.

3.3.2. Capacity

In the classical CVRP described in Section 1.3 all vehicles have a certain capacity, but for some of the problems of Ritplan there is no capacity defined. This can be because the orders are so small or light that the time limit or route length constraints will already make sure that the orders will always fit.

Nevertheless, a majority of 77% of the problems use capacity constraints.

3.3.3. Time windows - route length

In some of the problems the vehicles have time constraints. This can mean that the vehicle can only be used in one or more time windows. Next to this, all of the vehicles have restrictions on how long they can be used, since a driver has a certain maximum of working hours. For example it can be that a vehicle is available between 5 a.m. and 6 p.m., and a driver can only work for a maximum of 8 hours. This latter can be seen as a limited route length. This case is called a distance constrained vehicle routing problem in the literature. (DVRP, or DCVRP in case there are also capacity constraints.)

3.3.4. Costs

The goal of the optimization in Ritplan is to minimize the total costs of the vehicle routes. These costs depend on the costs of the vehicles. Vehicles can have three different types of costs:

- Fixed costs,
- Costs per hour,
- Costs per kilometre.

The fixed costs vary between 0 and 100 euros, but most of the time they are around 50. When the fixed costs are very low the model is stimulated to use more vehicles. When the fixed costs are high the model will less likely choose to use this vehicle. The costs per hour vary between 0 and 60 euros, and the average is around 15,50 euros. The costs per kilometre vary between 0,05 and 1,30 euros, and the average is around 0,35 euros.

If the costs per hour and kilometre are low but the fixed costs are high the model is more likely to build long distance routes rather than constructing a new route if the capacity and the distance constraint permits this.

3.3.5. Multiple trip

When a vehicle route has reached the limit of the capacity of the vehicle, but not the limit of the route duration, the vehicle can return to the depot at the end of the route to reload/unload. This is built in the model of RitOpt, so in all problems the vehicle tours have the ability to do this. In the literature this is called a *multi-trip* VRP (MTVRP), for a recent survey see Şen and Bülbül [71].

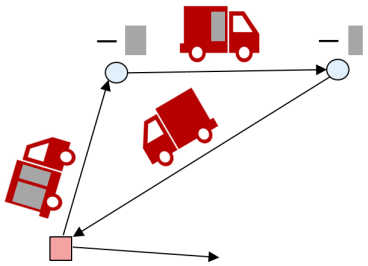


Figure 3.6: Multiple trip vehicle routing: When the capacity is very tight, but there is still time left for this route, a vehicle can do multiple trips.

3.3.6. Heterogeneous vehicles

In the literature, a VRP with a heterogeneous or mixed fleet of vehicles (HVRP) means that the vehicles have different capacity/costs/time constraints,. When all vehicles have common capacity/costs/time it is called a homogeneous fleet of vehicles.

In Ritplan the vehicles can also have different start and endpoints and next to this they can be of a different vehicle “type". This latter only results in extra constraints if some of the orders can only be served by specific vehicle types. This makes the problem a site dependent vehicle routing problem (SDVRP). In 88% of the cases different vehicle types were used, but often the orders can be served by all vehicle types.

Unfortunately there is no precise data on how often problems have heterogeneous capacity/cost/time vehicles, because it was not that easy to compare. It seemed that in all cases there where small differences between at least one of the factors. For example one vehicle that starts at 4 a.m. while the other one starts at 5 a.m. or a vehicle that has a fixed costs of 5.000 and another of 4.950. But there were also some big differences in the costs. There where for example cases in which some of the vehicles had no fixed costs at all while others had an average fixed cost around 5.000. This type of input will of course stimulate the model to use the ones with no fixed costs.

3.3.7. Start and endpoints

The Ritplan users define the start and endpoint of the vehicle tours. This can be one of the depots, but this can also be the home address of one of the drivers. In case the vehicle tour starts from the home address of one of the drivers the vehicle might first have to pass by a depot first to pick-up the orders to deliver. In the case all orders on the vehicle are preloaded, or in the case of only pick-up or pick-up and delivery orders this is not necessary.

The start point and the end point of the vehicle tours can differ. A vehicle can for example start from a home address and end at a depot, or can go from one depot to another. In 35% of the problems the endpoint was not the same as the start point.

3.3.8. Multiple depots

Around 31% of the problems have multiple depots and use preloading or afterwards unloading. This means these can be modelled as a multi-depot problem, MDVRP. It can also occur that there are not actual multiple depots but some of the vehicles do start or end at a different location, for example the home address of the driver, this can also be modelled as a MDVRP. A nice survey on genetic algorithms for MDVRP is described by Karakatič and Podgorelec [41].

3.3.9. Breaks

Around 38 % of the Ritplan users also want to schedule a break for the drivers, after a certain time of driving. At the moment these are often scheduled by hand, but these could be taken into account when optimizing, but this is mainly important when time windows have to be taken into account.

3.4. Attributes not covered by RitOpt

This section describes some of the problem attributes of customers of EVO-it that are not covered by the optimization model in RitOpt.

3.4.1. Dependency between orders

There are some customers of EVO-it that deliver orders that they have to pick-up again later. For example, this can be trays in which food is delivered that are reused every day. The receiver of these goods needs to get the time to unpack before the order is picked up again. Therefore there is time dependency between the delivery and the pick-up of these orders. At the moment this time dependency issue is solved with time windows. This is done by creating a time window to deliver all the food and creating a time window to pick up the food, such that there is enough time between this time window and the previous one.

This method will probably not provide the most efficient solution. Perhaps this could also be handled by using a smart dependency constraint, such that it could be solved in a more efficient way and get closer to an optimal solution.

3.4.2. Dockplanning

The issue of dockplanning of the customers of EVO-it is as follows. There is not enough room to load or unload all vehicles at the same time. Therefore there is dependency between vehicles for loading or unloading the trucks. This problem is now manually solved using fixed start and end times for the vehicles. Hence, the start/end times are not optimized by the algorithm of RitOpt.

3.4.3. “Hubplanning"

Large vehicles drive with many orders to an exchange point where the orders are put on small vehicles that will drive the delivery tours. At the moment this problem is also solved manually using time windows. In the literature these problems can be categorized as 2-echelon vehicle routing problems (VRP-2E), see Cuda et al. [22].

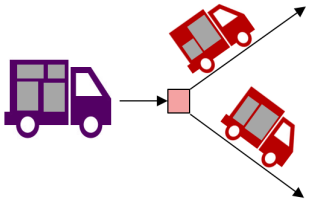


Figure 3.7: Hubplanning: A large vehicle drives to an exchange point (a “hub") where the orders divided over multiple small vehicles.

3.4.4. Deliver from any establishment

As said before each order has a “from” and a “to" address. Therefore orders need to be set “preloaded" to make the problem a multi-depot VRP. It would be better if there would be an option to set, in case of a delivery order type, the “from" addresses to any starting point. The same holds for the VRP for services. In this case there is nothing that needs to be loaded, so the vehicle can start anywhere. At this moment a service order is created with a pick-up and delivery at the same address.

3.4.5. Split orders

Sometimes the demand of an order does not fit in one vehicle and therefore has to be split into multiple orders. At the moment this is done by hand, but this can of course lead to suboptimal solutions.

A review on the VRP with split deliveries is given by Archetti and Speranza [3].

3.4.6. Traffic congestion

At some periods of the day some roads may take a lot more time because of traffic congestion. This means that there are time dependent travelling times. In the literature these problems can be described by time-dependent VRP (TDVRP).

3.4.7. Open VRP

In the open VRP (OVRP) the costs for the last return to the depot are not counted in the objective. Some of the customers of EVO-it want to use this version, because these costs are not passed on to the customer and are therefore not (or less) important for the optimisation. The OVRP is very similar to the CVRP, thus with a small adaptation the same solving methods can be used.

3.5. Overview

In the previous sections all problem types of the customers of EVO-it are discussed. Table 3.1 provides an overview of the problems and their occurrence.

This data is all gathered by taking a sample of 26 (from more or less 140) customers and looking through their cases in the Ritplan program. Therefore this only gives a very global idea of the occurrence of the problem types.

There was no information on how often problems occur that cannot be handled yet by Ritplan. Therefore, it was not possible to determine the current need for solving these problem types.

Problems with	**Percentage**
day planning	92%
period planning	8%
base route	31%
delivery or pick-up	54%
mixed: deliver, pick-up, pick-up & del.	46%
pick-up & delivery	19%
multi-depot (with preload/afterward unload)	31%
priority	15%
site dependency	42%
time windows	73%
capacity	77%
route length	100%
variable number of vehicles	100%
fixed costs	92%
heterogeneous vehicles	±100%[a]
multi-trip	n.a.[b]
start/end differ	35%
breaks	38%
fixed number of vehicles	n.a. [c]
dependency between orders	n.a. [c]
dockplanning	n.a. [c]
hubplanning	n.a. [c]
split orders	n.a. [c]
time dependent (traffic)	n.a. [c]
open VRP	n.a. [c]
#orders < 50	23%
#orders < 100	42%
#orders < 200	62%
#orders < 500	88%
#orders < 1000	100% [d]

Table 3.1: Overview problem types.
[a] It seemed that most of the problems had different types of vehicles, different costs, vehicles with different capacity or time constraints. Since there was no clear overview this is an estimate of how often this occurred.
[b] This has not been included in the data, when looking at instances of Ritplan. The reason for this was that it came to the attention a bit later and that it is more difficult to measure than some other problem attributes.
[c] Ritplan can not handle these type of problems yet. Therefore it was not possible to get information on how many customers of EVO-it are experiencing these problem types.
[d] EVO-it and CQM have agreed upon a maximum of 1000 orders that RitOpt should be able to handle. Therefore there are no (much) bigger problems than 1000 orders.

4

Extensions of exact algorithms

In Section 2.1 some concepts for the exact methods for the CVRP are discussed. It emerged that the current state-of-the-art method, by Pecin et al. [59], can solve problems with up to 200 customer locations. This is a huge improvement compared to what was possible when CQM started developing RitOpt 20 years ago. As seen in Table 3.1 more than half of the customers of EVO-it have problems with less than 200 orders. Furthermore the current performance of computers is much better than 20 years ago. For these reasons a research has been conducted in the area of the exact methods, and how those can be applied to solve real-life problems within reasonable time limits. This research will be elaborated in this chapter.

The goal of this research is find out if an exact method can be used to solve some (small cases) of the real life problems at EVO-it to eventually compare the results of RitOpt with the solutions of the exact method. This can provide insight in the possibilities for CQM to improve.

Moreover, the exact method can also be used to solve a simplified version of a problem. For example the exact method could provide a solution that does not take all the complicated constraints into account. This solution can be used as an initial solution to the heuristic that might be able to turn this into a good feasible solution. Another possibility is to solve a subproblem exactly when the heuristic has already divided the orders over the vehicles.

Even though the method of Pecin et al. [59] can solve problems with around 200 locations this does not mean it can solve the real life problems existing at EVO-it. The method of Pecin et al. [59] is developed for the standard CVRP and is, because of its complicated formulation, not easy to adapt to the extensions of the CVRP, if this is at all possible. Hence other exact formulations are considered in this chapter.

Taking into account *all* of the attributes that the customers of EVO-it are dealing with, has as a disadvantage that the exact method becomes overly extensive and complicated and may not finish within a reasonable amount of time. Better would be to look at the most important attributes, based on the data in Chapter 3. A selection of these attributes is made in Section 4.1.

In the subsequent sections different exact methods are elaborated. The computational results of the methods will be discussed in Chapter 5. The exact methods have been implemented using C++ and the solver of Gurobi Optimization [37].

4.1. Requirements

To solve real life problems from EVO-it with an exact method there are a few things the model should at least be able to handle. Based on the information in the previous chapter a list of requirements to be included in the model is constructed. The following problem attributes should be included in the model:

1. Capacitated vehicles, the standard CVRP.
2. A variable number of vehicles.
3. Distance constraints (DVRP).
4. Problems where not all orders can/need to be scheduled.

5. Different types of vehicle costs: fixed costs and variable costs. Where the variable costs can contain costs per km and costs per hour.

6. Heterogeneous vehicles, i.e. with different capacity, cost or maximum distance.

Even though Table 3.1 shows that many problems are dealing with time windows, these are left out of the scope of this project. Time windows are not easy to include in an exact method, thus this would make this research too comprehensive. Furthermore, for some of the problems, the time windows were not that strict and would unlikely be a bottleneck when solving the problem. Therefore the number of problems where time windows actually have a real influence on the solving process is expected to be lower. Nevertheless, it can be a good next step to include these, since many problems are dealing with time windows.

An interesting attribute named in the list above is number 4. Problems where not all orders can be scheduled, mentioned in Section 3.2.5. This requirement is remarkable, because it has not been mentioned in any literature about exact methods for VRPs before (that I know of).

The next sections describe the formulations used to solve different type of VRPs and trying to meet the requirements described above. For the implementation of the methods C++ is used and the solver of Gurobi Optimization [37] to solve the mixed-integer programming (MIP) problems.

4.2. Two-index flow formulation

The first step in the research about the application of exact methods to real life problems, was to start with the most intuitive formulation. This is the two-index flow formulation (F), described in Section 2.1, and repeated below.

$$(F) \qquad \text{minimize} \quad \sum_{(i,j)\in E} c_{ij}x_{ij}, \tag{4.1}$$

$$\text{subject to} \quad \sum_{(i,j)\in\delta(\{h\})} x_{ij} = 2, \qquad \forall h \in V_c \tag{4.2}$$

$$\sum_{j\in V_c} x_{0j} = 2m, \tag{4.3}$$

$$\sum_{i,j\in\delta(S)} x_{ij} \geq 2v(S), \qquad \forall S \subseteq V_c, |S| \geq 2 \tag{4.4}$$

$$x_{ij} \in \{0,1\}, \qquad \forall (i,j) \notin \delta(0) \tag{4.5}$$

$$x_{ij} \in \{0,1,2\}, \qquad \forall (i,j) \in \delta(0) \tag{4.6}$$

This formulation is for the standard CVRP, so this model only covers the first two attributes in the list of requirements. To test the implementation, cases from Uchoa et al. [78] have been used. See Chapter 5 for the test results of this formulation.

4.2.1. First implementation

In the first implementation of formulation (F) the complete set of subsets $\mathscr{S} = \{S \subseteq V_c, |S| \geq 2\}$ is generated to make the subtour elimination constraints (SECs). This is clearly not the best way to implement this formulation, since the number of subsets increases exponentially ($|\mathscr{S}| = 2^n$ where $n = |V_c|$), but it was very informative to see in practice how the time to create the subsets and the SECs as well as the time for Gurobi to solve the problem increased as the problem size got bigger. The results can be found in the first columns of Table 5.2.

4.2.2. Second implementation

In the first tests of the two-index flow formulation, Test 1 in Table 5.2, no initial solution was provided. The time Gurobi needed to find a feasible solution had a significant contribution to the total solution time. Therefore in the next test, Test 2 in Table 5.2, an initial solution is provided to Gurobi. This initial solution is constructed using the Clarke and Wright savings method (CWS), as described in Section 2.2. Providing an initial solution immediately provides an upper bound to the optimal solution.

4.2.3. Third and fourth implementation

For the instances of Uchoa et al. [79] a fixed amount of vehicles is assumed. This is how many of the CVRPs for exact methods are defined in literature. However in reality the amount of vehicles is usually not fixed. In

reality, there is an upper bound, since there are not infinitely many vehicles available, but using less vehicles is often not a problem. For this reason the number of vehicles is assumed to be a variable in the third and fourth implementation. Let x be the number of vehicles in the instances of Uchoa et al. [79]. This number x is very tight, i.e., this is the minimum number of vehicles needed to get a feasible solution. For this reason the number of available vehicles is increased. The upper bound to the number of available vehicles is set to $U = x + 3$ and, instead of m being a fixed parameter, m becomes an integer variable $0 \leq m \leq U$.

Increasing the number of available vehicles has the advantage that a CWS has a better chance at providing a feasible initial solution.

4.2.4. Final Branch-and-Bound implementation

In the first implementations of the two-index flow formulation all subsets and all subtour elimination constraints (SECs) (4.2) were constructed. However, building these sets and constraints takes an exponential amount of time. Furthermore, solving a problem with this many constraints takes too much time.

In the final implementation of the two-index flow formulation a branch-and-bound method as described in Section 2.1.3 is implemented. This highly improved the performance of the exact method. Using the branch-and-bound method means that the SECs are dropped and relaxation (LF) described in Section 2.1.3 is used.

Gurobi already uses a method based on branch-and-bound to solve MIP problems. The user can retrieve the solutions that Gurobi finds using a callback function. This way, when Gurobi finds an integer solution for relaxation (LF), this solution can be checked for subtours. When a subtour is found the corresponding SEC is added to the relaxation using the lazy constraint functions of Gurobi. The results of using the branch-and-bound method described in Section 2.1.3 are presented in Table 5.3.

Implementing the two-index flow formulation was very informative and a good way to learn how to create such a model in C++ and how to work with Gurobi, however this formulation is not suitable to the requirements made in Section 4.1. For this reason the next Section will continue with the three-index flow formulation.

4.3. Three-index flow formulation

The three-index flow formulation is very similar to the two-index flow formulation and also has the quality of being very intuitive. The three-index flow formulation extends the two-index flow formulation with, next to indices for the vertices $i, j \in V$, an index for the vehicles $k \in \{1, ..., U\}$ where U is the total number of vehicles available. This section will describe the three-index flow formulation and adjust the formulation such that all of the requirements of Section 4.1 are satisfied.

4.3.1. Basic three-index flow formulation

To be able to solve problems with a maximum tour length and with heterogeneous vehicles an extra index for the vehicles is needed. In this formulation the variables

$$y_i^k = \begin{cases} 1 \text{ if customer } i \text{ is served by vehicle } k \\ 0 \text{ otherwise} \end{cases}$$

and

$$x_{ij}^k = \begin{cases} 1 \text{ if vehicle } k \text{ traverses edge } (i, j) \in A \text{ from } i \text{ to } j \\ 0 \text{ else} \end{cases}$$

are used to define which customers are visited by which vehicles and which edges are traversed to do this. Note that in this formulation arcs A are used instead of edges E, which means traversing from i to j is different from traversing from j to i. In addition a capacity parameter Q_k and a maximum route length parameter L_k is defined for each vehicle $k \in \{1, ..., U\}$. The maximum route length in Ritplan is defined in time. To keep track of the length of a route, a travel time parameter t_{ij}^k is introduced for each arc $(i, j) \in A$ and for each vehicle $k \in \{1, ..., U\}$. Based on the three-index flow formulation by Fisher and Jaikumar [27] [28], this leads to the following formulation.

$$(F3) \qquad \text{minimize} \quad \sum_{k=1}^{U} \sum_{i,j \in V} c_{ij} x_{ij}^k, \tag{4.7}$$

$$\text{subject to} \quad \sum_{i=1}^{n} q_i y_i^k \leq Q_k \quad \forall k \in \{1,...,U\} \tag{4.8}$$

$$\sum_{i,j \in V} x_{ij}^k t_{ij}^k \leq L_k \quad \forall k \in \{1,...,U\} \tag{4.9}$$

$$\sum_{j=0}^{n} x_{ij}^k = y_i^k \quad \forall i \in V_c, \forall k \in \{1,...,U\} \tag{4.10}$$

$$\sum_{i=0}^{n} x_{ij}^k = y_j^k \quad \forall j \in V_c, \forall k \in \{1,...,U\} \tag{4.11}$$

$$\sum_{i,j \in S} x_{ij}^k \leq |S| - 1 \quad S \subseteq V_c,\ S \geq 2, \forall k \in \{1,...,U\} \tag{4.12}$$

$$\sum_{k=1}^{U} y_i^k = 1 \quad \forall i \in V_c \tag{4.13}$$

$$x_{ij}^k \in \{0,1\} \quad \forall i,j \in V,\ \forall k \in \{1,...,U\} \tag{4.14}$$

$$y_{jk} \in \{0,1\} \quad \forall j \in V,\ \forall k \in \{1,...,U\} \tag{4.15}$$

Constraints 4.8 and 4.9 make sure that, respectively, the capacity and the route length constraints are not violated. Constraints 4.10 and 4.11 guarantee that a customer i, visited by vehicle k, uses an arc going towards customer i and an arc coming from customer i used by vehicle k. Constraints 4.12 are the subtour elimination constraints of this formulation. This constraint ensures that for each vehicle k each subset of customers $S \in V_c$ has no more than $|S|-1$ arcs traversed by that vehicle. This way a vehicle can not have any tours where the depot is not visited. The last constraints 4.13 ensure that each customer is visited by exactly one vehicle.

With this formulation requirements 4 and 5 are not yet included in the model. Therefore in the next subsection an extension to this model will be made.

4.3.2. Three-index flow formulation with penalties

To solve a problem with more orders than it is possible to deliver (requirement 4) while satisfying the capacity and the distance constraints, constraint 4.13 should be dropped. Instead, a penalty parameter p_i is introduced for each customer i. When customer i is not visited, the objective value increases with p_i. When $p_i = p,\ \forall i \in V_c$ and p is large enough the model is forced to visit a maximum number of customers. In case the penalty is different for some customers and it is not possible to visit all customers, this might not be the case, since customers with a higher penalty will be prioritized over customers with a lower penalty.

Let F_k be the fixed cost for using vehicle k. Let t_{ij}^k and d_{ij}^k, respectively, be the travel time from customer i to customer j and the distance going from customer i to customer j, for vehicle k. If these are the same for each vehicle we use $t_{ij}^k = t_{ij}$ and $d_{ij}^k = d_{ij}, \forall (i,j) \in A$. Furthermore, c_t^k is defined to be the cost per time unit and c_d^k the cost per distance unit for vehicle k. Let

$$c_{ij}^k := c_t^k t_{ij}^k + c_d^k d_{ij}^k. \tag{4.16}$$

Including everything in the model results in formulation (FP), a three-index flow formulation with penalties.

To find out if a vehicle k is used, the sum of all edges from vehicle k leaving the depot is taken. If one of these edges is used by vehicle k then $x_{0j}^k = 1$ for some $j \in V_c$ and we know this vehicle is used and the fixed cost needs to be included.

To ensure that the program will always try to schedule a visit for each customer, the penalty p_i should always be of higher costs than the costs to visit customer i with any of the vehicles. For this implementation p_i is set to

$$p_i = p = \max_{k \in M}\{F_k\} + \max_{k \in M}\left\{\max_{j \in V_c}\{c_{0j}^k + c_{j0}^k\}\right\}, \quad \forall i \in V_c$$

However, this would make it more beneficial to visit a customer multiple times, hence constraint 4.23 is needed to make sure that a customer is not visited more than once.

$$(FP) \qquad \text{minimize} \quad \sum_{k=1}^{U}\left(\sum_{i,j\in V} c_{ij}^k x_{ij}^k + \sum_{j\in V_c} x_{0j}^k F_k\right) + \sum_{i\in V_c} p_i\left(1-\sum_{k=1}^{U} y_i^k\right), \tag{4.17}$$

$$\text{subject to} \quad \sum_{i=1}^{n} q_i y_i^k \le Q_k \quad \forall k\in\{1,...,U\} \tag{4.18}$$

$$\sum_{i,j\in V} x_{ij}^k t_{ij} \le L_k \quad \forall k\in\{1,...,U\} \tag{4.19}$$

$$\sum_{j=0}^{n} x_{ij}^k = y_i^k \quad \forall i\in V, \forall k\in\{1,...,U\} \tag{4.20}$$

$$\sum_{i=0}^{n} x_{ij}^k = y_j^k \quad \forall j\in V, \forall k\in\{1,...,U\} \tag{4.21}$$

$$\sum_{i,j\in S} x_{ij}^k \le |S|-1 \quad S\subseteq V_c,\ S\ge 2, \forall k\in\{1,...,U\} \tag{4.22}$$

$$\sum_{k=1}^{U} y_i^k \le 1 \quad \forall i\in V_c \tag{4.23}$$

$$x_{ij}^k \in \{0,1\} \quad \forall i,j\in V,\ \forall k\in\{1,...,U\} \tag{4.24}$$

$$y_{jk} \in \{0,1\} \quad \forall j\in V,\ \forall k\in\{1,...,U\} \tag{4.25}$$

4.3.3. Problems covered by the model

Using some of the tricks and definitions described in Baldacci and Mingozzi [5] formulation (FP) covers the following classes of vehicle routing problems:

- The capacitated VRP by setting $Q_k = Q,\ L_k = \infty$ (and/or $t_{ij} = 0$), $c_{ij}^k = c_{ij}$ and $F_k = 0\ \forall k\in\{1,...,m\}$.
- The heterogeneous VRP, where a mixed fleet of vehicles has different capacities Q_k, variable and fixed routing costs c_{ij}^k, F_k.
- The distance constrained VRP, where a maximum route length L_k can be provided per vehicle k.
- The site dependent VRP, where some customers i can only be visited by a specific subset of vehicles $k\in M_i$. This is achieved by setting the costs

$$c_{ij}^k = \begin{cases} c_{ij}, \text{ if } k\in M_i\cap M_j \\ \infty, \text{ otherwise.}\end{cases}$$

- The multi-depot VRP, where some vehicles depart and end at a different depot. This can be achieved by adjusting the distance (and travel time) from the depot to all other locations. Suppose there are p depots and let $\hat{d}_{i,j}$ be the distance matrix where $\hat{d}_{n+h,i}$ is the travel cost for going from depot $h\in\{1,...,p\}$ to location $i\in V_c$. Now, if vehicle k uses depot h, for the edges coming from the depot going towards a customer i, set

$$d_{0i}^k = \hat{d}_{hi}$$

  and do the same for the edges d_{i0}^k going from a customer i towards the depot 0:

$$d_{i0}^k = \hat{d}_{i_h}$$

  This way for each vehicle k a distance matrix d_{ij}^k is constructed where $di0^k$ and d_{0i}^k are adjusted to start position of the depot.

Furthermore, this formulation can solve problems where not all orders have to be scheduled. This means that this formulation is also able to handle problems with more orders than feasible for the available amount of vehicles, while satisfying the capacity and/or route length constraints, which is something that has not been included in any literature (to the best of my knowledge).

4.3.4. Solving (FP)

A problem in Formulation (FP) is solved in a similar way to a problem in Formulation (F). A relaxation of (FP) is constructed by dropping the subtour constraints 4.22. The branch-and-bound method is used and in each iteration where an integer solution is found, the solution is checked for subtours. If a subtour is found the corresponding subtour constraint is added for *each* vehicle k (otherwise the subtour could move from vehicle to vehicle). Nevertheless, a similar instance defined in Formulation (FP) takes a lot more time compared to solving this problem using Formulation (F). This can be explained by looking at the number of constraints the models use.

After dropping the subtour constraints, Formulation (F) uses only n constraints, where n is the number of locations. Whereas for Formulation (FP), there are still $U + U + nU + nU + U = (2n+2)U + n$ constraints left after dropping the subtour constraints, where U is the number of available vehicles. This large number of constraints will have a negative effect on the solving time. Besides, each time Gurobi finds an integer solution that contains a subtour, the corresponding violated subtour constrains will also be included. Where only one subtour constraint at a time is included in Formulation (F), Formulation (FP) includes U subtour constraints each time; the same subtour constraint for each vehicle k.

The results using the three-index flow formulation (FP) are reported in Section 5.2.

Since the number of constraints and variables grows so fast when the number of vehicles or the number of customers increases, solving larger problems with this formulation is very limited. For this reason another formulation developed for heterogeneous vehicle routing problems (and a few other VRP extensions) is evaluated, in the next section.

4.4. Set Partitioning formulation

As mentioned in Section 2.1, next to the two-index flow formulation, the set partitioning formulation is one of the most widely used mathematical formulations of the CVRP. Baldacci and Mingozzi [5] have developed an unified exact method that makes use of this formulation and they are eventually able to solve heterogeneous vehicle routing problems (HVRPs) with up to 75 locations.

The formulation of Baldacci and Mingozzi [5] can deal with almost all of the requirements mentioned in Section 4.1, except for requirement 3 (distance constrained VRPs) and requirement 4 (problems where not all orders need to be scheduled). Because this formulation was already so close to meeting all the requirements and the results reported in the article of Baldacci and Mingozzi [5] are promising, it is chosen to explore this formulation in further detail and investigate if it is possible to extend it to meeting all the requirements.

This section will describe the formulation of Baldacci and Mingozzi [5] and the procedure to solve a HVRP using this formulation. In the next section the adjustments to the formulation and the procedures to also meet requirements 3 and 4. are described.

4.4.1. Formulation (ESP)

In this formulation a set $M = \{1, ..., m\}$ is defined as the set of vehicle types. For each vehicle type $k \in M$ the upper bound to the number of available vehicles is defined by U_k. As described in the previous section there is a limited capacity Q_k, a fixed cost parameter F_k and a cost matrix c_{ij}^k, but now defined for each vehicle *type* k.

Let $\mathscr{R}^k$ be the set of all feasible routes of vehicle type $k \in M$, and the complete set of feasible routes $\mathscr{R} = \bigcup_{k \in M} \mathscr{R}^k$. The set $\mathscr{R}_i^k \subset \mathscr{R}^k$ is defined to be the set of routes of vehicle type k covering customer i. Each route $l \in \mathscr{R}^k$ is associated with a cost c_l^k and a binary variable x_l^k, where

$$x_l^k = \begin{cases} 1 \text{ if route } l \in \mathscr{R}^k \text{ is chosen} \\ 0 \text{ otherwise.} \end{cases}$$

Let $R_l^k \subset V_c$ be the subset of customers visited by route $l \in \mathscr{R}^k$. Now, the extended set partitioning formulation (ESP) for the heterogeneous vehicle routing problem (HVRP) is defined as follows:

$$(ESP) \quad \text{minimize} \quad \sum_{k\in M}\sum_{l\in\mathscr{R}^k}(F_k + c_l^k)x_l^k, \tag{4.26}$$

$$\text{subject to} \quad \sum_{k\in M}\sum_{l\in\mathscr{R}_i^k} x_l^k = 1, \quad \forall i \in V_c, \tag{4.27}$$

$$\sum_{l\in\mathscr{R}^k} x_l^k \le U_k, \quad \forall k \in M, \tag{4.28}$$

$$x_l^k \in \{0,1\}, \quad \forall l \in \mathscr{R}^k, \forall k \in M \tag{4.29}$$

Constraints (4.27) make sure that each customer is visited by exactly one of the vehicles. Constraints (4.28) ensure that the number of routes for a specific vehicle does not exceed the number of available vehicles.

To solve (*ESP*) a set of feasible routes $\mathscr{R}^k$ is needed for each vehicle type $k \in M$. Instead of finding the complete set of feasible routes $\mathscr{R}^k$ for each vehicle (which can become exponentially large), a reduced set $\hat{\mathscr{R}}^k \subseteq \mathscr{R}^k$, $\forall k \in M$ is generated using the solutions of lower bound methods to eliminate routes that cannot belong to the optimal solution of (*ESP*).

Baldacci and Mingozzi [5] give three relaxations which are used for five different lower bound procedures. One of these relaxations is described in Subsection 4.4.2. The bounding procedures H^1 and H^2 that make use of this relaxation are described in Subsections 4.4.3 and 4.4.4.

In Subsection 4.4.5 it is explained how the attained lower bound can eliminate the routes that cannot belong to the optimal solution of the HVRP. To generate the reduced set $\hat{\mathscr{R}}^k$, the procedure GENROUTE described by Baldacci et al. [6] is used. The essence of procedure GENROUTE will be explained in Subsection 4.4.6. When the reduced sets $\hat{\mathscr{R}}^k$ are generated, the reduced problem ($\widehat{ESP}$), obtained by replacing each set $\mathscr{R}^k$ by the set $\hat{\mathscr{R}}^k$, can be solved.

While there are five bounding procedures described by Baldacci and Mingozzi [5], this thesis focusses on the first two lower bound procedures, H^1 and H^2. To proceed and improve the results found by this thesis the next three lower bounds can be included.

4.4.2. Relaxation LSP of (*ESP*)

The first two lower bound procedures H^1 and H^2 mentioned in the article are based on the LP-relaxation, LSP, of problem (*ESP*), where 4.29 is relaxed to $x_l^k \ge 0$. The dual formulation (*DSP*) of LSP is as follows.

$$(DSP) \quad \text{maximize} \quad \sum_{i\in V_c} u_i + \sum_{k\in M} U_k v_k, \tag{4.30}$$

$$\text{subject to} \quad \sum_{i\in R_l^k} u_i + v_k \le c_l^k + F_k, \quad \forall l \in \mathscr{R}^k, \forall k \in M, \tag{4.31}$$

$$u_i \in \mathbb{R}, \quad i \in V_c, \tag{4.32}$$

$$v_k \le 0, \forall k \in M \tag{4.33}$$

Where $\mathbf{u} = (u_1, ..., u_n)$ and $\mathbf{v} = (v_1, ..., v_m)$ are the dual variables associated with constraints (4.27) and (4.28), respectively.

Solving the dual problem (*DSP*) is still impractical, because the sets of feasible routes $\mathscr{R}^k$, which are typically exponentially many, are still needed. However, an important remark to this problem is that any feasible solution for the dual problem gives a lower bound on the HVRP (weak duality theorem). Baldacci and Mingozzi [5] describe a clever way to find good solutions to (*DSP*), without generating all feasible sets, which is used for both procedures H^1 and H^2. They are both based on the following theorem.

Theorem 1. *Associate penalties $\lambda_i \in \mathbb{R}$, $\forall i \in V_c$, with constraints 4.27 and penalties $\mu_k \le 0$, $\forall k \in M$, with constraints 4.28. Define*

$$b_{ik} = q_i \min_{l\in\mathscr{R}_i^k}\left\{\frac{c_l^k + F_k - \lambda(R_l^k) - \mu_k}{q(R_l^k)}\right\}, \quad \forall i \in V_c, \forall k \in M \tag{4.34}$$

where $\lambda(R_l^k) = \sum_{i\in R_l^k}\lambda_i$ and $q(R_l^k) = \sum_{i\in R_l^k} q_i$.

A feasible (DSP) *solution* $(\mathbf{u},\mathbf{v})$ *is given by:*

$$u_i = \min_{k\in M}\{b_{ik}\} + \lambda_i, \quad \forall i \in V_c, \tag{4.35}$$

$$v_k = \mu_k, \quad \forall k \in M. \tag{4.36}$$

Proof. From Baldacci and Mingozzi [5]:
Note that for a route $l \in \mathscr{R}^k$ for a given vehicle type $k \in M$, it holds that $l \in \mathscr{R}_i^k,\ \forall i \in R_l^k$. From this follows that:

$$b_{ik} \le q_i \frac{c_l^k + F_k - \lambda(R_l^k) - \mu_k}{q(R_l^k)}, \quad \forall i \in R_l^k. \tag{4.37}$$

Defining $(\boldsymbol{u},\boldsymbol{v})$ as in Expression (4.35) and (4.36) gives:

$$\begin{aligned}\sum_{i\in R_l^k} u_i &\le \sum_{i\in R_l^k} q_i \frac{c_l^k + F_k - \lambda(R_L^k) - \mu_k}{q(R_l^k)} + \sum_{i\in R_l^k}\lambda_i \\ &= c_l^k + F_k - \mu_k \\ &= c_l^k + F_k - v_k\end{aligned}$$

Which corresponds to the dual constraint (4.31). □

Both procedures H^1 and H^2 use subgradient optimization to find a solution. In the following sections the procedures are elaborated.

4.4.3. Lower bound procedure H^1

Procedure H^1 uses Theorem 1, q-routes and subgradient optimization to compute a (DSP) solution $(\mathbf{u}^1,\mathbf{v}^1)$ of costs $LH1 = DSP(\boldsymbol{u}^1,\boldsymbol{v}^1)$, which is a lower bound to problem (ESP).

The requirement that a route should be a simple cycle is relaxed and instead *q-routes* are used. Recall that a q-route is a route starting and ending at the depot, traversing a set of customers with a total demand q, $q \in W^k$. Where W^k is the set of possible loads on vehicle type k which can be defined as

$$W_k = \{q : q = \sum_{i\in V_c} \xi_i q_i, \text{ such that } \xi \in \{0,1\}, q \le Q_k\}. \tag{4.38}$$

Note that a q-route is not necessarily simple, i.e., it can happen that vertices are visited more than once. For a given penalty vector λ the modified edge costs $\overline{c}_{ij}^k$ are defined as follows:

$$\overline{c}_{ij}^k = c_{ij}^k - \frac{1}{2}\lambda_i - \frac{1}{2}\lambda_j, \quad \forall (i,j) \in E,\ \forall k \in M. \tag{4.39}$$

Such that the modified cost of a route $l \in \mathscr{R}^k$ is $\overline{c}_l^k = \sum_{(i,j)\in E(R_l^k)} \overline{c}_{ij}^k = c_l^k - \lambda(R_l^k)$.

Let $C(k,q,i)$ be a q-route for vehicle k with a total demand of exactly q, starting from the depot passing through vertex $i \in V_c$ and returning back to the depot. The function $\phi(k,q,i)$ is defined to be the cost of the least cost q-route $C(k,q,i)$ using the modified edge costs $\overline{c}_{ij}^k$. This makes $\phi(k,q,i)$ a lower bound for the cost $\overline{c}_l^k$ for any feasible route $l \in \mathscr{R}_i^k$ with a total load $q(R_l^k) = q$. The values of ϕ are computed using the method described in Christofides et al. [16]. Important to mention is that for simplicity the cost matrices $(c_{ij})^k$ are assumed to be symmetric. A new value $\overline{b}_{ik}$ is defined as follows:

$$\overline{b}_{ik} = q_i \min_{q_i \le q \le Q_k}\left\{\frac{F_k + \phi(k,q,i) - \mu_k}{q}\right\}, \quad \forall i \in V_c,\ \forall k \in M \tag{4.40}$$

such that $\overline{b}_{ik} \le b_{ik}$. Replacing b_{ik} in Expression (4.35) with $\overline{b}_{ik}$, Theorem 1 remains valid.

Procedure H^1 starts with initializing $LH1 = 0$, $\lambda = 0$ and $\mu = 0$. Furthermore an upper bound z_{UB} is needed in the execution. This can be the objective value of a feasible solution. H^1 executes a predefined number of $Maxt1$ iterations, where in each iteration the following steps are executed:

Step 1. Compute the modified costs $\overline{c}_{ij}^{k}$ as in Expression (4.39) and use these to compute the q-route functions $\phi(k,q,i)$ for all $k \in M,\ \forall i \in V$ and for all possible loads $q \in W_k,\ q \geq q_i$.

Step 2. Compute values $\overline{b}_{ik}$ using Expression (4.40) and use these values to compute a feasible (DSP) solution $(\mathbf{u},\mathbf{v})$ using Expressions (4.35) and (4.36).
Let $\overline{q}_{ik}$ be the value of q giving the minimum in (4.40) and let k_i be the vehicle type k giving the minimum in (4.35) for $i \in V_c$. Hence,

$$u_i = \overline{b}_{ik_i} + \lambda_i = q_i \frac{F_{k_i} + \phi(k_i, \overline{q}_{ik_i}, i) - \mu_{k_i}}{\overline{q}_{ik_i}} + \lambda_i.$$

If the computed solution $DSP(\boldsymbol{u},\boldsymbol{v}) > LH1$, then update lower bound $LH1 = DSP(\boldsymbol{u},\boldsymbol{v})$ and $\boldsymbol{u}^1 = \boldsymbol{u}, \boldsymbol{v}^1 = \boldsymbol{v}, \boldsymbol{\lambda}^1 = \boldsymbol{\lambda}$ and $\boldsymbol{\mu}^1 = \boldsymbol{\mu}$

Step 3. Let θ_j be the total number of times that customer $j \in V$ is visited by the q-routes $C(k_i, \overline{q}_{ik_i}, i)\ \forall i \in V_c$. Let ρ_k be the number of vehicles of type k used by the q-routes $C(k_i, \overline{q}_{ik_i}, i)\ \forall i \in V_c$ and let $\delta_j(k_i, \overline{q}_{ik_i}, i)$ be the number of times that customer j is visited by the route $C(k_i, \overline{q}_{ik_i}, i)$ for $i \in V_c$.
Initialize $\theta_j = 0, \forall j \in V_c$ and $\rho_k = 0,\ \forall k \in M$. Update the values of θ and ρ as follows:
For each customer $i \in V_c$, update $\theta_j = \theta_j + \frac{q_i}{\overline{q}_{ik_i}} \delta_j(k_i, \overline{q}_{ik_i}, i),\ \forall j \in V_c$ and update $\rho_{k_i} = \rho_{k_i} + \frac{q_i}{\overline{q}_{ik_i}}$.
Remark: In the article of Baldacci and Mingozzi [5] ρ_k is updated as $\rho_k = \rho_k + \frac{q_i}{\overline{q}_{ik_i}}, \forall k \in M$. This is incorrect, since it would result in the same value for all ρ_k. Therefore this is adjusted to $\rho_k = \rho_k + \frac{q_i}{\overline{q}_{ik_i}}$, only for $k = k_i$ instead of $\forall k \in M$, which is equal to updating $\rho_{k_i} = \rho_{k_i} + \frac{q_i}{\overline{q}_{ik_i}}$, as in line 13. This adjustment has been communicated to the authors prof. Baldacci and prof. Mingozzi and has been confirmed.

Step 4. (Subgradient optimization step.) Update the values of $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ as follows:
Let $\gamma := \frac{Z_{UB} - DSP(\mathbf{u},\mathbf{v})}{\sum_{j \in V_c}(\theta_j - 1)^2 + \sum_{k \in M}(\rho_k - U_k)^2}$ and ϵ a positive constant value. For each customer $j \in V_c$ update $\lambda_j = \lambda_j - \epsilon\gamma(\theta_j - 1)$ and for each vehicle type $k \in M$ update $\mu_k = \min\{0, \mu_k - \epsilon\gamma(\rho_k - U_k)\}$. These updates will make the solution $DSP(\mathbf{u},\mathbf{v})$ computed with Expression (4.35) and (4.36) converge to an optimal value for (DSP).

A summary of the procedure of H^1 to compute the lower bound $LH1$ can be described as follows:

Procedure 1 Bounding procedure H^1

Input: (1) An upper bound z_{UB}, (2) a set W_k, $\forall k \in M$ of all possible loads, (3) a value $Maxt1$, which prescribes the number of iterations (4) and a positive constant ϵ.
Output: A (DSP) solution $(\mathbf{u^1}, \mathbf{v^1})$ and a lower bound $LH1 = DSP(\mathbf{u^1}, \mathbf{v^1})$ to problem (ESP).

Initialize: $LH1 = 0$, $\lambda = 0, \mu = 0$ and $it = 0$;
repeat
  Compute the modified costs $\overline{c}_{ij}^k$ as in Expression 4.39;
  Use $\overline{c}_{ij}^k$ to compute the functions $\phi(k,q,i)$ for all $q \in W_k$;
  Compute values $\overline{b}_{ik}$ using Expression 4.40;
  Use $\overline{b}_{ik}$ to compute a feasible (DSP) solution $(\mathbf{u},\mathbf{v})$ using Expressions 4.35 and 4.36;
  if the costs $DSP(\mathbf{u},\mathbf{v}) > LH1$ **then**
    Update $LH1 = DSP(\mathbf{u},\mathbf{v})$, $\mathbf{u^1} = \mathbf{u}$, $\mathbf{v}^1 = \mathbf{v}$, $\boldsymbol{\lambda}^1 = \boldsymbol{\lambda}$ and $\boldsymbol{\mu}^1 = \boldsymbol{\mu}$;
  end
  Compute the values of $\boldsymbol{\theta}$ and $\boldsymbol{\rho}$: Initialize $\theta_j = 0$, $\forall j \in V_c$, and $\rho_k = 0$, $\forall k \in M$;
  for each customer i in V_c **do**
    Update $\theta_j = \theta_j + \frac{q_i}{\overline{q}_{ik_i}} \delta_j(k_i, \overline{q}_{ik_i}, i)$, $\forall j \in V_c$;
    Update $\rho_{k_i} = \rho_{k_i} + \frac{q_i}{\overline{q}_{ik_i}}$ $^{(a)}$;
  end
  Let $\gamma = \frac{Z_{UB} - DSP(\mathbf{u},\mathbf{v})}{\sum_{j \in V_c}(\theta_j - 1)^2 + \sum_{k \in M}(\rho_k - U_k)^2}$
  Update $\lambda_j = \lambda_j - \epsilon\gamma(\theta_j - 1)$, $\forall j \in V_c$
  Update $\mu_k = \min\{0, \mu_k - \epsilon\gamma(\rho_k - U_k)\}$, $\forall k \in M$
  $it = it + 1$;
until $it = Maxt1$

There is also a method implemented that can reduce the upper bound U_k for the number of vehicles. This can reduce the problem if necessary. This method uses lower bound procedure H^1 and is executed after H^1 is executed.This method is performed as described in the article of Baldacci and Mingozzi [5].

4.4.4. Lower bound procedure H^2

Procedure H^2 uses the dual solution achieved by H^1 and applies a column generation method to compute an even better lower bound $LH2$. Different from other column generation methods, H^2 uses the expressions from Theorem 1 and subgradient optimization to solve the master problem heuristically.

Procedure H^2 starts with initializing $LH2 = 0$, $\boldsymbol{\lambda} = \boldsymbol{\lambda}^1$ and $\boldsymbol{\mu} = \boldsymbol{\mu}^1$.

Next, the initial master problem of the column generation method is defined. The master problem is obtained from relaxation LSP by replacing $\mathscr{R}^k$ with $\overline{\mathscr{R}}^k \subset \mathscr{R}^k$. To create subset $\overline{\mathscr{R}}^k \subset \mathscr{R}^k$ the procedure GENROUTE is used, which is described in Subsection 4.4.6. For the input of GENROUTE the (DSP) solution $(\boldsymbol{u}^1, \boldsymbol{v}^1)$ obtained by H^1 is used and the parameters γ and Δ are to $\gamma = \infty$ and $\Delta = \Delta^{\min}$. Where $\Delta^{\min}$ is a predefined parameter.

Subsequently, H^2 executes a predefined number of $Maxt2$ macro iterations, where in each iteration the following steps are executed:

Step 1. **Solve the master problem.** Initialize $z^* = 0$. Repeat for a predefined number of $Maxt3$ iterations:

(a) Compute the dual solution $DSP(\boldsymbol{u},\boldsymbol{v})$ to the master problem using expressions (4.34), (4.35) and (4.36), with the current values of λ and μ.
If the computed solution $DSP(\boldsymbol{u},\boldsymbol{v}) > z^*$, then update $z^* = DSP(\boldsymbol{u},\boldsymbol{v})$, $\boldsymbol{u}^* = \boldsymbol{u}$ and $\boldsymbol{v}^* = \boldsymbol{v}$

(b) Let k_i, θ_j and ρ_k be defined as in procedure H^1. Let l_{ik} be the route $l \in \overline{\mathscr{R}}_i^k$ giving the minimum expression in Equation 4.34. Hence,

$$u_i = b_{ik_i} + \lambda_i = q_i \frac{c_{l_{ik_i}}^{k_i} + F_{k_i} - \lambda(R_{l_{ik_i}}^{k_i}) - \mu_{k_i}}{q(R_{l_{ik_i}}^{k_i})} + \lambda_i$$

Furthermore define $\delta_j(l_{ik_i})$ to be the number of times customer j is visited by route l_{ik_i}:

$$\delta_j(l_{ik_i}) := \begin{cases} 1, \text{ if customer } j \text{ is visited by the route } l_{ik_i} \text{ for } i \in V_c \\ 0 \text{ otherwise} \end{cases}$$

(c) (Subgradient optimization step.) Update the penalties $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ as described for procedure H^1.

Step 2. **Check if the solution to the master problem is a feasible** (DSP) **solution.** For each $k \in M$, use the procedure GENROUTE (Subsection 4.4.6) to generate the largest subset $\mathcal{N}^k \subset \mathcal{R}^k \backslash \overline{\mathcal{R}}^k$, $|\mathcal{N}^k| \leq \Delta^a$ containing the routes with the largest negative reduced cost with respect to the dual solution $(\boldsymbol{u}^*, \boldsymbol{v}^*)$. Where Δ^a is a predefined parameter.

If $\mathcal{N}^k = \emptyset$, $\forall k \in M$, and $z^* > LH2$, then update $LH2 = z^*$, $\boldsymbol{u}^2 = \boldsymbol{u}^*$ and $\boldsymbol{v}^2 = \boldsymbol{v}^*$.

If $\mathcal{N}^k \neq \emptyset$, for some $k \in M$, then extend the master problem $\overline{\mathcal{R}}^k = \overline{\mathcal{R}}^k \bigcup \mathcal{N}^k$, $\forall k \in M$.

A summary of the procedure of H^2 is described in Procedure 2.

Procedure 2 Bounding procedure H^2

Input: (1) The (DSP) solution $(\mathbf{u^1},\mathbf{v^1})$ found by H^1 and $\boldsymbol{\lambda}^1$ and $\boldsymbol{\mu}^1$ from H^1. (2) Parameters $\Delta^{\min}$, Δ^a, $Maxt2$ and $Maxt3$.

Output: A DSP solution $(\boldsymbol{u}^2,\boldsymbol{v}^2)$ and a lower bound $LH2 = DSP(\boldsymbol{u}^2,\boldsymbol{v}^2)$ to problem (ESP).

Initialize: $LH2 = z^* = 0,\ \lambda = \boldsymbol{\lambda}^1, \mu = \boldsymbol{\mu}^1$ and $it2 = it3 = 0$;

Use GENROUTE to generate $\overline{\mathscr{R}}^k \subset \mathscr{R}^k$, for each $k \in M$, using the following input parameters:

$$\hat{\boldsymbol{u}} = \boldsymbol{u^1},\ \hat{v}^k = v_k^1,\ \gamma = \infty \text{ and } \Delta = \Delta^{\min}$$

The master problem of LSP is then obtained by replacing $\mathscr{R}^k$ with $\overline{\mathscr{R}}^k,\ \forall k \in M$;

repeat

  repeat

    Compute a dual solution $(\boldsymbol{u},\boldsymbol{v})$ to the master problem, using expressions 4.34, 4.35 and 4.36 for the current values of $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$;

    if $DSP(\boldsymbol{u},\boldsymbol{v}) > z^*$ **then**

      Update $z^* = DSP(\boldsymbol{u},\boldsymbol{v}), (\boldsymbol{u}^*,\boldsymbol{v}^*) = (\boldsymbol{u},\boldsymbol{v})$;

    end

    Compute the values of $\boldsymbol{\theta}$ and $\boldsymbol{\rho}$:

    Initialize $\theta_j = 0,\ \forall j \in V_c$, and $\rho_k = 0,\ \forall k \in M$;

    for each customer i in V_c **do**

      Update $\theta_j = \theta_j + \frac{q_i}{q(R^{k_i}_{l_{ik_i}})}\delta_j(l_{ik_i}),\ \forall j \in V_c$;

      Update $\rho_{k_i} = \rho_{k_i} + \frac{q_i}{q(R^{k_i}_{l_{ik_i}})}$;

    end

    Let $\gamma = \frac{Z_{UB} - DSP(\mathbf{u},\mathbf{v})}{\sum_{j\in V_c}(\theta_j - 1)^2 + \sum_{k\in M}(\rho_k - U_k)^2}$;

    Update $\lambda_j = \lambda_j - \epsilon\gamma(\theta_j - 1),\ \forall j \in V_c$;

    Update $\mu_k = \min\{0, \mu_k - \epsilon\gamma(\rho_k - U_k)\},\ \forall k \in M$;

    $it3 = it3 + 1$;

  until $it3 = Maxt3$

  Check if the master solution $(\boldsymbol{u}^*,\boldsymbol{v}^*)$ is feasible:

  For each $k \in M$, use GENROUTE to generate the set $\mathscr{N}^k \subset \mathscr{R}^k \backslash \overline{\mathscr{R}}^k$, using the following input:

$$\hat{\boldsymbol{u}} = \boldsymbol{u}^*,\ \hat{v}^k = v_k^*,\ \gamma = 0 \text{ and } \Delta = \Delta^a$$

  if $\mathscr{N}^k = \emptyset,\ \forall k \in M$ and $z^* > LH2$ **then**

    Update $LH2 = z^*,\ \boldsymbol{u}^2 = \boldsymbol{u}^*,\ \boldsymbol{v}^2 = \boldsymbol{v}^*$;

  else if $\mathscr{N}^k \neq \emptyset$ for some $k \in M$ **then**

    update $\overline{\mathscr{R}}^k = \overline{\mathscr{R}}^k \bigcup \mathscr{N}^k,\ \forall k \in M$;

  end

  $it2 = it2 + 1$;

until $it2 = Maxt2$

4.4.5. Solving (ESP)

As mentioned in Subsection 4.4.1, to solve the set partitioning formulation, (ESP), the set of feasible routes $\mathscr{R}^k$ is needed. Instead of generating the complete set $\mathscr{R}^k$, $\forall k \in M$, an upper bound and a lower bound to problem (ESP) are used to select a subset $\hat{\mathscr{R}}^k \subseteq \mathscr{R}^k$ of feasible routes, such that any route $r \in \mathscr{R}^k \backslash \hat{\mathscr{R}}^k$ can not belong to the optimal solution of problem (ESP).

In the method of Baldacci and Mingozzi [5] one of the final two lower bounds is used to create the subset of feasible routes $\hat{\mathscr{R}}^k \subseteq \mathscr{R}^k$.

Even though the final two lower bounds are not created for this project, it is possible to create the subsets $\hat{\mathscr{R}}^k \subseteq \mathscr{R}^k, \forall k \in M$, such that it contains the necessary feasible routes. This is done by using a dual (DSP) solution $(\hat{\boldsymbol{u}}, \hat{\boldsymbol{v}})$ as the input of the GENROUTE procedure of Baldacci et al. [6] and set the parameters $\gamma = z_{UB} - DSP(\hat{\boldsymbol{u}}, \hat{\boldsymbol{v}})$ and $\Delta = \infty$. This way the procedure GENROUTE, described in Subsection 4.4.6, creates the reduced sets $\hat{\mathscr{R}}^k$, for each vehicle k, containing all routes $l \in \mathscr{R}^k$ for which

$$c_l^k + F_k - \sum_{i \in R_l^k} u_i - v_k \leq z_{UB} - DSP(u, v)$$

Replacing the sets $\mathscr{R}^k$ in problem (ESP) with the reduced sets $\hat{\mathscr{R}}^k$ results in the reduced problem $\widehat{ESP}$. Problem $\widehat{ESP}$ can be solved using Gurobi Optimization [37] or another MIP solver.

In Theorem 2 it is proven that the subset $\hat{\mathscr{R}}^k$ has the property that any route $r \in \mathscr{R}^k \backslash \hat{\mathscr{R}}^k$ can not belong to the optimal solution of (ESP).

Theorem 2. *Let $DSP(\boldsymbol{u}, \boldsymbol{v})$ be the objective value of (DSP) for a feasible solution $(\boldsymbol{u}, \boldsymbol{v})$ and z_{UB} an upper bound to problem (ESP). Any route $l \in R^k$, $k \in M$ for which*

$$c_l^k + F_k - \sum_{i \in \mathscr{R}_l^k} u_i - v_k > z_{UB} - DSP(u, v)$$

can not belong to the optimal solution of the HVRP.

Proof. This will be proven in a similar way as proven for the lower bound $LD2$ in the article of Baldacci and Mingozzi [5]. Define the reduced costs of a route $l \in \mathscr{R}^k$ as

$$\hat{c}_l^k = c_l^k + F_k - \sum_{i \in R_l^k} u_i - v_k.$$

Let $\overline{x}$ be a solution to ESP, with costs $ESP(\overline{x})$. Using the definition of the reduced costs $\hat{c}_l^k$ gives:

$$\sum_{k \in M} \sum_{l \in \mathscr{R}^k} \overline{x}_l^k \hat{c}_l^k = \sum_{k \in M} \sum_{l \in \mathscr{R}^k} \overline{x}_l^k (c_l^k + F_k) - \sum_{k \in M} \sum_{l \in \mathscr{R}^k} \overline{x}_l^k \sum_{i \in R_l^k} u_i - \sum_{k \in M} \sum_{l \in \mathscr{R}^k} \overline{x}_l^k v_k$$

Note that each customer is visited exactly once, because of constraints (4.27) and therefore

$$\sum_{k \in M} \sum_{l \in \mathscr{R}^k} \overline{x}_l^k \sum_{i \in R_l^k} u_i = \sum_{i \in V_c} u_i$$

Furthermore, not more than U_k vehicles of type k can be used, because of constraints (4.28). Let $\overline{m}_k$ be the number of times vehicle k is used, hence $\lim m_k = \sum_{l \in \mathscr{R}^k} \overline{x}_l^k$. We obtain:

$$\begin{aligned}
\sum_{k \in M} \sum_{l \in \mathscr{R}^k} \overline{x}_l^k \hat{c}_l^k &= ESP(\overline{x}) - \sum_{i \in V_c} u_i - \sum_{k \in M} \overline{m}_k v_k \\
&\leq ESP(\overline{x}) - \Big(\sum_{i \in V_c} u_i + \sum_{k \in M} U_k v_k \Big) \\
&= ESP(\overline{x}) - DSP(\boldsymbol{u}, \boldsymbol{v})
\end{aligned}$$

If $\overline{x}$ is an optimal solution, or any solution better than or equal to the upper bound, then it must hold that $ESP(\overline{x}) - DSP(\boldsymbol{u}, \boldsymbol{v}) \leq z_{UB} - DSP(u, v)$. Note that $\hat{c}_l^k \geq 0, \forall l \in \mathscr{R}^k$, $k \in M$, because of constraint 4.31. Hence, a route $l \in \mathscr{R}^k$, $k \in M$ for which $\hat{c}_l^k > z_{UB} - DSP(u, v)$, can never belong to an optimal solution. □

4.4.6. Procedure GENROUTE

GENROUTE is a dynamic programing procedure, described by Baldacci et al. [6], that generates a set of feasible routes. Procedure GENROUTE is used in bounding procedure H^2 to create the inital sets $\overline{R}^k \subset \mathscr{R}^k$ and the sets $\mathscr{N} \subset \mathscr{R}^k$, for each $k \in M$, and finally to create the sets $\hat{\mathscr{R}}^k \subseteq \mathscr{R}^k,\ \forall k \in M$, required to solve the reduced problem $\widehat{ESP}$ of ESP. This subsection will give an idea of what GENROUTE does. For a further explanation the reader is referred to the article of Baldacci et al. [6].

To create the set of feasible routes $\mathscr{B}^k \subseteq \mathscr{R}^k$ for a vehicle type k GENROUTE uses a solution to the dual of a linear relaxation, LSP, of ESP. When LSP is extended with capacity constraints and clique inequalities, the dual solution contains four variables, i.e., $(\hat{\boldsymbol{u}}, \hat{\boldsymbol{v}}, \hat{\boldsymbol{w}}, \hat{\boldsymbol{g}})$. These are used for the input of GENROUTE in Baldacci and Mingozzi [5]. Since the relaxation used in this report does not have capacity constraints and clique inequalities, a solution $(\hat{\boldsymbol{u}}, \hat{\boldsymbol{v}})$ to the dual (DSP) is provided instead.

For each vehicle type $k \in M$, given the vector $\hat{\boldsymbol{u}}$ and $\hat{v}_k$, and two pre-defined parameters γ and Δ, GENROUTE will produce the largest subset $\mathscr{B}^k \subseteq \mathscr{R}^k$ satisfying the following conditions:

(a) $\max_{l \in \mathscr{B}^k}\{c_l^k\} \leq \min_{l \in \mathscr{R}^k \setminus \mathscr{B}^k}\{\hat{c}_l^k\}$

(b) $|\mathscr{B}^k| \leq \Delta$

(c) $\max_{l \in \mathscr{B}^k}\{\hat{c}_l^k\} \leq \gamma$

Where $\hat{c}_l^k,\ l \in \mathscr{R}^k$, is the reduced cost which are defined as

$$\hat{c}_l^k = c_l^k + F_k - \sum_{i \in R_l^k} \hat{u}_i - \hat{v}_k \tag{4.41}$$

The procedure GENROUTE consists of two phases.

Phase 1. Generating a set $\mathscr{P}$ of paths, using the GENPATH procedure.

Phase 2. Generating the route set $\mathscr{B}^k$ by combining the paths generated by GENPATH.

In the procedure GENPATH, a modified cost $\overline{c}_{ij}^k$ is used. These modified costs are defined as

$$c_{ij}^k = c_{ij}^k - \frac{1}{2}\hat{u}_i - \frac{1}{2}\hat{u}_j \tag{4.42}$$

For a route $l \in \mathscr{R}^k$ this gives the following modified costs

$$c_l^k = \sum_{(i,j) \in E(R_l^k)} \overline{c}_{ij}^k = c_l^k - \sum_{i \in \mathscr{R}_l^k} \hat{u}_i \tag{4.43}$$

Which results to the first observation (O1) of Baldacci et al. [6]:

(O1) Since $\hat{v}_k \leq 0$ and $F_k \geq 0$ it holds that $\hat{c}_l^k \geq \overline{c}_l^k,\ \forall l \in R^k, \forall k \in M$.

The GENPATH procedure defines a function $LB(P)$ to compute a lower bound to the modified cost $\overline{c}(R)$ of any route R containing path P. This leads to the observation, (O5) of Baldacci et al. [6]:

(O5) Any path P for which $LB(P) > \gamma$ cannot belong to a route of the final set $\mathscr{B}^k$, since $\overline{c}(R) \leq \hat{c}(R)$ for any route R.

A few other observations are made by Baldacci et al. [6], for these the reader is referred to the article.

Using all of the observations in Baldacci et al. [6], Baldacci et al., conclude that route set $\mathscr{B}^k$ can be constructed using only paths P from the largest simple path set $\mathscr{P}$ satisfying the following conditions:

(a) $\max_{P \in \mathscr{P}}\{LB(P)\} \leq \min_{P \notin \mathscr{P}}\{LB(P), \gamma\}$

(b) $|\mathscr{P}| \leq \Delta$

(c) $q(P) \leq \frac{Q}{2} + q_{e(P)},\ \forall P \in \mathscr{P}$, where $e(P)$ is the last vertex in the path P.

(d) a path $P \in \mathscr{P}$ cannot be *dominated* by another path $P' \in \mathscr{P}$. Where a path P is dominated by P' if $e(P) = e(P')$, $V_c(P) = V_c(P')$ and $c(P) \geq c(P')$.

For the exact steps in the procedure GENPATH and GENROUTE the reader is referred to Baldacci et al. [6].

4.5. Set partitioning formulation with penalties

The formulation and method of Baldacci and Mingozzi [5] does not include distance constrained VRPs or VRPs where not all orders can and need to be scheduled. Therefore, this section describes the adjustments made to be able to deal with these problem types. Note that the exact method described in this section also covers all of the problem types that were described in Subsection 4.3.3 and that were also covered by the three-index flow formulation with penalties.

Subsection 4.5.1 will describe the adjustments to formulation (ESP) and introduce a set partitioning formulation with penalties (SPP) which is able to deal with VRPs where not all orders need to be scheduled. The dual formulation of the LP-relaxation $LSPP$ of (SPP) is described in sSubsection 4.5.2. Furthermore, subsections 4.5.3 and 4.5.4 describe the bounding procedures H^{1*} and H^{2*}, which are the adjusted versions of lower bound procedures H^1 and H^2 respectively. Subsequently subsections 4.5.5 and 4.5.6 show how to generate the set of feasible routes $\hat{\mathscr{R}}^k \subseteq \mathscr{R}^k$, for each $k \in M$, that are needed to solve (SPP).

All of the procedures, H^{1*}, H^{2*} and GENROUTE, need an upper bound. To generate this upper bound an extended version of the Clark and Wright Savings method, CWS* is used. This is described in Subsection 4.5.7.

4.5.1. Formulation (SPP)

Assume that the set of feasible routes $\mathscr{R}^k$ satisfies the distance constraints. Adjusting formulation (ESP) to one that can also deal with problems where not all orders have to be scheduled, gives the set partitioning formulation with penalties (SPP). Where y_i is defined to be a binary variable for which

$$y_i = \begin{cases} 1 \text{ if customer } i \text{ is visited} \\ 0 \text{ otherwise} \end{cases}$$

and a penalty p_i as defined in Section 4.3.

(SPP)

$$\text{minimize} \quad \sum_{k\in M} \sum_{l\in\mathscr{R}^k} (F_k + c_l^k) x_l^k - \sum_{i\in V_c} p_i y_i, \tag{4.44}$$

$$\text{subject to} \quad -y_i + \sum_{k\in M} \sum_{l\in\mathscr{R}_i^k} x_l^k = 0, \quad \forall i \in V_c, \tag{4.45}$$

$$\sum_{l\in\mathscr{R}^k} x_l^k \leq U_k, \quad \forall k \in M, \tag{4.46}$$

$$y_i \leq 1, \quad \forall i \in V_c, \tag{4.47}$$

$$x_l^k \in \{0,1\}, \quad \forall l \in \mathscr{R}^k, \forall k \in M \tag{4.48}$$

$$y_i \in \{0,1\}, \quad \forall i \in V_c \tag{4.49}$$

Note that instead of adding the penalty when a customer is not visited (as in Section 4.3), the penalty is subtracted whenever a customer is visited. Thus, in the objective function, "$-\sum_{i\in V_c} p_i y_i$" is used instead of "$+\sum_{i\in V_c} p_i(1-y_i)$" as was done in the three-index flow formulation with penalties. This way it can be seen as a reward that is achieved when the customer is visited, instead of a penalty when the customer is not visited.

This will result in the same set of optimal solutions $(\mathbf{x}, \mathbf{y})$, since $\sum_{i\in V_c} p_i$ is a constant. Note that the value of the objective of (SPP) is not equal to the costs. When the actual value of the costs needs to be calculated $\sum_{i\in V_c} p_i$ should be added to the objective value.

4.5.2. Relaxation LSPP of (SPP)

Let $LSPP$ be the LP-relaxation of (SPP). The dual of $LSPP$ is denoted by DSNP. Let $\mathbf{w} = \{w_1, ..., w_n\}$ be the dual variable vector associated to Constraints (4.47).

$$\text{(DSPP)} \qquad \text{maximize} \quad \sum_{k\in M} U_k v_k + \sum_{i\in V_c} w_i, \tag{4.50}$$

$$\text{subject to} \quad \sum_{i\in R_l^k} u_i + v_k \le c_l^k + F_k, \quad \forall l \in \mathscr{R}^k,\ \forall k \in M, \tag{4.51}$$

$$-u_i + w_i \le -p_i, \quad i \in V_c, \tag{4.52}$$

$$u_i \in \mathbb{R}, \quad i \in V_c, \tag{4.53}$$

$$v_k \le 0, \forall k \in M \tag{4.54}$$

$$w_i \le 0, \forall i \in V_c \tag{4.55}$$

$$\tag{4.56}$$

To get a feasible $(DSPP)$ solution $(\mathbf{u},\mathbf{v},\mathbf{w})$ equations (4.35) and (4.36) from Theorem 1 can still be used and setting

$$w_i = \min\{0, u_i - p_i\},\ \forall i \in V_c \tag{4.57}$$

will make sure that Constraints (4.52) are satisfied and $w_i \le 0$.

4.5.3. Lower bound procedure H^{1*}

This section will describe the adjustments made to procedure H^1 of [5] to solve problems with distance constraints and orders that do not need to be scheduled. The new bounding procedure is defined as H^{1*}.

Adjustments to the q-route procedure

For lower bound procedure H^1 a relaxation of q-routes is used. The dynamic programming method used to compute the cost $\phi(k,q,i)$ for the q-routes for H^1 does not take any distance constraints into account. Therefore the dynamic programming method in H^1 needs to be adjusted to make sure that the q-routes also satisfy the distance constraints, otherwise $\overline{b}_{ik}$ would not necessarily provide a lower bound to b_{ik}.

To derive the costs $\phi(k,q,i)$ for the q-routes the following functions are introduced. Let $f(k,q,i)$ be the cost of the least cost *q-path*, for vehicle type k, starting from the depot ending at vertex i and with a total demand $q \in W_k, q_i \le q$. Let $\pi(k,q,i)$ be the vertex prior to node i in the path corresponding to $f(k,q,i)$. Let $\psi(k,q,i)$ be the cost of the least cost path starting from the depot ending at vertex i, with a total demand q and with node $\overline{\pi}(k,q,i)$ prior to i, such that $\overline{\pi}(k,q,i) \neq \pi(k,q,i)$. How to compute these functions is described by Christofides et al. [16].

To make sure that the q-routes satisfy the distance constraints, it is necessary to keep track of the travelled distance for the paths corresponding to the functions $f(k,q,i)$ and $\psi(k,q,i)$. Let $t_f(k,q,i)$ and $t_\psi(k,q,i)$ be the travelled distance for the path corresponding to $f(k,q,i)$ and $\psi(k,q,i)$ respectively. The initialization of the distance functions is as follows:

$$t_f(k,q_i,i) = t_{0i}^k,\ t_\psi(k,q_i,i) = \infty \quad \forall i \in V_c,$$
$$t_f(k,q,i) = \infty,\ t_\psi(k,q,i) = \infty \quad \forall q \neq q_i,\ \forall i \in V_c.$$

For a given vertex i and load q, let $l = q - q_i$, $\tau = \pi(k,q,i)$ and $\overline{\tau} = \overline{\pi}(k,q,i)$. The values for t_f and t_ψ are updated as follows:

$$t_f(k,q,i) = \begin{cases} t_f(k,l,\tau) + t_{\tau i}^k, & \text{if } \pi(k,l,\tau) \neq i \\ t_\psi(k,l,\tau) + t_{\tau i}^k & \text{otherwise} \end{cases} \tag{4.58}$$

$$t_\psi(k,q,i) = \begin{cases} t_f(k,l,\overline{\tau}) + t_{\overline{\tau} i}^k, & \text{if } \pi(k,l,\overline{\tau}) \neq i \\ t_\psi(k,l,\overline{\tau}) + t_{\overline{\tau} i}^k & \text{otherwise} \end{cases} \tag{4.59}$$

The q-route $C(k,q,i)$ is composed from two q-paths ending at vertex i, whose total loads add up to $q' = q + q_i$. In procedure H^1, the cost of the q-route $\phi(k,q,i)$ was computed as follows:

$$\phi(k,q,i) = \min_{q_i \le q^* \le \frac{1}{2}q'} \left[\begin{cases} f(k,q^*,i) + f(k,q'-q^*,i), & \text{if } \pi(k,q^*,i), \neq \pi(k,q'-q^*,i) \\ \min\{f(k,q^*,i) + \psi(k,q'-q^*,i), \psi(k,q^*,i) + f(k,q'-q^*,i)\}, & \text{if } \pi(k,q^*,i) = \pi(k,q'-q^*,i) \end{cases} \right]$$

In procedure H^{1*} this is replaced with:

$$\phi(k,q,i) = \min_{q_i \le q^* \le \frac{1}{2}q'} \left[\begin{cases} f(k,q^*,i) + f(k,q'-q^*,i), & \text{if } \pi(k,q^*,i) \neq \pi(k,q'-q^*,i) \\ & \text{and } t_f(k,q^*,i) + t_f(k,q'-q^*,i) \le L_k \\ \min\{f(k,q^*,i) + \psi(k,q'-q^*,i), \psi(k,q^*,i) + f(k,q'-q^*,i)\}, & \text{if } \pi(k,q^*,i) = \pi(k,q'-q^*,i) \\ & \text{and } t_f(k,q^*,i) + t_f(k,q'-q^*,i) \le L_k \\ \infty, \quad \text{otherwise} \end{cases} \right]$$

Computing lower bound $LH1$

Procedure H^{1*} uses the same steps as procedure H^1 to compute a $(DSPP)$ solution of costs $LH1 = DSPP(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$, such that $LH1$ is a lower bound to problem (SPP).

Instead of computing a (DSP) solution $(\mathbf{u}, \mathbf{v})$ in Step 2. procedure H^1, procedure H^{1*} computes a $(DSPP)$ solution $(\mathbf{u}, \mathbf{v}, \mathbf{w})$ is computed using expressions (4.40), (4.35), (4.36) and (4.57).

The summary of procedure H^{1*} is described in Procedure 3.

Procedure 3 Bounding procedure H^{1*}

Input: (1) An upper bound z_{UB}, (2) a set W_k, $\forall k \in M$ of all possible loads, (3) a value $Maxt1$, which prescribes the number of iterations (4) and a positive constant ϵ.

Output: A $(DSPP)$ solution $(\mathbf{u}^1, \boldsymbol{v}^1, \boldsymbol{w}^1)$ and a lower bound $LH1 = DSPP(\boldsymbol{u}^1, \boldsymbol{v}^1, \boldsymbol{w}^1)$ to problem (SPP).

1: Initialize: $LH1 = 0$, $\lambda = 0, \mu = 0$ and $it = 0$;
2: **repeat**
3: Compute the modified costs $\overline{c}^k_{ij}$ as in Expression 4.39;
4: Use $\overline{c}^k_{ij}$ to compute the costs $\phi(k,q,i)$ using the adjusted q-route procedure for all $q \in W_k$;
5: Compute values $\overline{b}_{ik}$ using Expression 4.40;
6: Use $\overline{b}_{ik}$ to compute a feasible $(DSPP)$ solution $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$ using Expressions 4.35, 4.36 and 4.57;
7: **if** the costs $DSPP(\mathbf{u}, \mathbf{v}, \boldsymbol{w}) > LH1$ **then**
8: Update $LH1 = DSPP(\mathbf{u}, \mathbf{v}, \boldsymbol{w})$, $\mathbf{u^1} = \mathbf{u}$, $\mathbf{v}^1 = \mathbf{v}$, $\boldsymbol{w}^1 = \boldsymbol{w}$, $\boldsymbol{\lambda}^1 = \boldsymbol{\lambda}$ and $\boldsymbol{\mu}^1 = \boldsymbol{\mu}$;
9: **end**
10: Compute the values of $\boldsymbol{\theta}$ and $\boldsymbol{\rho}$: Initialize $\theta_j = 0$, $\forall j \in V_c$, and $\rho_k = 0$, $\forall k \in M$;
11: **for each** customer i in V_c **do**
12: Update $\theta_j = \theta_j + \frac{q_i}{\overline{q}_{ik_i}} \delta_j(k_i, \overline{q}_{ik_i}, i)$, $\forall j \in V_c$;
13: Update $\rho_{k_i} = \rho_{k_i} + \frac{q_i}{\overline{q}_{ik_i}}$ $^{(a)}$;
14: **end**
15: Let $\gamma = \frac{Z_{UB} - DSP(\mathbf{u},\mathbf{v})}{\sum_{j \in V_c}(\theta_j - 1)^2 + \sum_{k \in M}(\rho_k - U_k)^2}$
16: Update $\lambda_j = \lambda_j - \epsilon\gamma(\theta_j - 1)$, $\forall j \in V_c$
17: Update $\mu_k = \min\{0, \mu_k - \epsilon\gamma(\rho_k - U_k)\}$, $\forall k \in M$
18: $it = it + 1$;
19: **until** $it = Maxt1$

4.5.4. Lower bound procedure H^{2*}

For procedure H^2 similar adjustments are made as for procedure H^1. The new bounding procedure is defined as H^{2*}.

H^{2*} is executed as described in Subsection 4.4.4, where instead of the (DSP) solution $(\boldsymbol{u}, \boldsymbol{v})$, a $(DSPP)$ solution $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$ is computed using Expression 4.57 next to expressions 4.34, 4.35 and 4.36. Furthermore, when generating the subsets $\overline{R}^k \subset \mathscr{R}^k$ for each $k \in M$ and $\mathscr{N}^k \subset \mathscr{R}^k \backslash \overline{\mathscr{R}}^k$ the adjusted version of GENROUTE, GENROUTE*, that takes distance constraints into account, described in Section 4.5.6 is used.

The pseudo code of procedure H^{2*} is described in Procedure 4.

Procedure 4 Bounding procedure H^{2*}

Input: (1) The $(DSPP)$ solution $(\mathbf{u^1}, \mathbf{v^1}, \boldsymbol{w}^1)$ found by H^{1*} and $\boldsymbol{\lambda}^1$ and $\boldsymbol{\mu}^1$ from H^{1*}. (2) Parameters $\Delta^{\min}$, Δ^a, ϵ, $Maxt2$ and $Maxt3$.

Output: A DSPP solution $(\boldsymbol{u}^2, \boldsymbol{v}^2, \boldsymbol{w}^2)$ and a lower bound $LH2 = DSPP(\boldsymbol{u}^2, \boldsymbol{v}^2, \boldsymbol{w}^2)$ to problem (SPP).

Initialize: $LH2 = z^* = 0,\ \lambda = \boldsymbol{\lambda}^1, \mu = \boldsymbol{\mu}^1$ and $it2 = it3 = 0$;

Use GENROUTE* to generate $\overline{\mathscr{R}}^k \subset \mathscr{R}^k$, for each $k \in M$, using the following input parameters:

$$\hat{\boldsymbol{u}} = \boldsymbol{u}^1,\ \hat{v}^k = v_k^1,\ \hat{\boldsymbol{w}} = \boldsymbol{w}^1 \gamma = \infty \text{ and } \Delta = \Delta^{\min}$$

The master problem of $LSPP$ is then obtained by replacing $\mathscr{R}^k$ with $\overline{\mathscr{R}}^k,\ \forall k \in M$;

repeat

  repeat

    Compute a dual solution $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$ to the master problem, using expressions 4.34, 4.35, 4.36 and 4.57 for the current values of $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$;

    if $DSP(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}) > z^*$ **then**

      Update $z^* = DSP(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}), (\boldsymbol{u}^*, \boldsymbol{v}^*, bmw^*) = (\boldsymbol{u}, \boldsymbol{v}, bmw)$;

    end

    Compute the values of $\boldsymbol{\theta}$ and $\boldsymbol{\rho}$:

    Initialize $\theta_j = 0,\ \forall j \in V_c$, and $\rho_k = 0,\ \forall k \in M$;

    for each customer i in V_c **do**

      Update $\theta_j = \theta_j + \frac{q_i}{q(R^{k_i}_{l_{ik_i}})} \delta_j(l_{ik_i}),\ \forall j \in V_c$;

      Update $\rho_{k_i} = \rho_{k_i} + \frac{q_i}{q(R^{k_i}_{l_{ik_i}})}$;

    end

    Let $\gamma = \frac{Z_{UB} - DSP(\mathbf{u},\mathbf{v})}{\sum_{j \in V_c}(\theta_j - 1)^2 + \sum_{k \in M}(\rho_k - U_k)^2}$;

    Update $\lambda_j = \lambda_j - \epsilon\gamma(\theta_j - 1),\ \forall j \in V_c$;

    Update $\mu_k = \min\{0, \mu_k - \epsilon\gamma(\rho_k - U_k)\},\ \forall k \in M$;

    $it3 = it3 + 1$;

  until $it3 = Maxt3$

  Check if the master solution $(\boldsymbol{u}^*, \boldsymbol{v}^*, \boldsymbol{w}^*)$ is feasible:

  For each $k \in M$, use GENROUTE* to generate the set $\mathscr{N}^k \subset \mathscr{R}^k \backslash \overline{\mathscr{R}}^k$, using the following input:

$$\hat{\boldsymbol{u}} = \boldsymbol{u}^*,\ \hat{v}^k = v_k^*,\ \hat{\boldsymbol{w}}^k = \boldsymbol{w}^*,\ \gamma = 0 \text{ and } \Delta = \Delta^a$$

  if $\mathscr{N}^k = \emptyset,\ \forall k \in M$ and $z^* > LH2$ **then**

    Update $LH2 = z^*,\ \boldsymbol{u}^2 = \boldsymbol{u}^*,\ \boldsymbol{v}^2 = \boldsymbol{v}^*,\ \boldsymbol{w}^2 = \boldsymbol{w}^*$;

  else if $\mathscr{N}^k \neq \emptyset$ for some $k \in M$ **then**

    update $\overline{\mathscr{R}}^k = \overline{\mathscr{R}}^k \bigcup \mathscr{N}^k,\ \forall k \in M$;

  end

  $it2 = it2 + 1$;

until $it2 = Maxt2$

4.5.5. Solving (SPP)

To solve (SPP) the reduced problem $\widehat{SPP}$ is created by replacing each set $\mathscr{R}^k$ with the reduced set $\hat{\mathscr{R}}^k$.

Since the problem (ESP) is extended to problem (SPP), a different reduced cost is used to generate the reduced sets $\hat{\mathscr{R}}^k \subset \mathscr{R}^k$, for each $k \in M$, such that any route $r \in \mathscr{R}^k \backslash \hat{\mathscr{R}}^k$ can not belong to the optimal solution of (SPP).

For problem (SPP) the reduced cost $\hat{c}_l^k$ of a route $l \in \mathscr{R}^k$ is defined as

$$\hat{c}_l^k = c_l^k + F_k - \sum_{i \in R_l^k} (w_i + p_i) - v_k.$$

Let z_{DSPP} be the objective value of a feasible solution $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$ to $(DSPP)$. For each $k \in M$, the reduced subset $\hat{\mathscr{R}}^k$ is generated as the set of all routes $l \in \mathscr{R}^k$ for which

$$\hat{c}_l^k \leq z_{UB} - z_{DSPP}. \tag{4.60}$$

Theorem 2 shows that any route $l \in \mathscr{R}^k \backslash \hat{\mathscr{R}}^k$ can not belong to the optimal solution.

To create a set of feasible routes $\hat{\mathscr{R}}^k$, for each $k \in M$, that satisfies the distance constraints, the procedure GENROUTE*, described in Subsection 4.5.6, is used. The parameters required for GENROUTE* are defined as follows:

$$\hat{\boldsymbol{u}} = \boldsymbol{w} + \boldsymbol{p},\ \hat{v}^k = v_k^1,\ \gamma = z_{UB} - DSP(u, v, w) \text{ and } \Delta = \infty$$

After the sets $\hat{\mathscr{R}}^k$ are generated the reduced problem $\widehat{SPP}$ can be solved using an MIP solver such as Gurobi Optimization [37].

Theorem 3. *Let z_{DSPP} be the objective value of a feasible solution $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$ to $(DSPP)$ and z_{UB} be an upper bound to Problem (SPP). Any route $l \in \mathscr{R}^k,\ k \in M$ for which*

$$\hat{c}_l^k > z_{UB} - z_{DSPP} \tag{4.61}$$

can not belong to an optimal solution of (SPP).

Proof. Suppose that $(\overline{x}, \overline{y})$ is a feasible solution to (SPP) with costs $\overline{z}_{SPP}$. Using the definition of $\hat{c}_l^k$, gives

$$\sum_{k \in M} \sum_{l \in \mathscr{R}^k} \overline{x}_l^k \hat{c}_l^k = \sum_{k \in M} \sum_{l \in \mathscr{R}^k} \overline{x}_l^k (c_l^k + F_k) - \sum_{k \in M} \sum_{l \in \mathscr{R}^k} \overline{x}_l^k \sum_{i \in R_l^k} (w_i + p_i) - \sum_{k \in M} \sum_{l \in \mathscr{R}^k} \overline{x}_l^k v_k.$$

From constraints (4.45) we know $\sum_{k \in M} \sum_{i \in R_l^k} \overline{x}_l^k = \overline{y}_i,\ \forall i \in V_c$. Let $\overline{m}_k = \sum_{l \in \mathscr{R}^k} \overline{x}_l^k$ be the number of times vehicle k is used. Using this we derive

$$\sum_{k \in M} \sum_{l \in \mathscr{R}^k} \overline{x}_l^k \hat{c}_l^k = \sum_{k \in M} \sum_{l \in \mathscr{R}^k} \overline{x}_l^k (c_l^k + F_k) - \sum_{i \in V_c} \overline{y}_i (w_i + p_i) - \sum_{k \in M} \overline{m}_k v_k \tag{4.62}$$

$$\leq \sum_{k \in M} \sum_{l \in \mathscr{R}^k} \overline{x}_l^k (c_l^k + F_k) - \sum_{i \in V_c} \overline{y}_i p_i - \sum_{i \in V_c} \overline{y}_i w_i - \sum_{k \in M} U_k v_k \tag{4.63}$$

$$\tag{4.64}$$

Note that $\overline{z}_{SPP} = \sum_{k \in M} \sum_{l \in \mathscr{R}^k} \overline{x}_l^k (c_l^k + F_k) - \sum_{i \in V_c} \overline{y}_i p_i$. Furthermore $w_i \leq 0,\ \forall i \in V_c$. Hence,

$$\sum_{k \in M} \sum_{l \in \mathscr{R}^k} \overline{x}_l^k \hat{c}_l^k \leq \overline{z}_{SPP} - (\sum_{i \in V_c} w_i + \sum_{k \in M} U_k v_k) \tag{4.65}$$

$$= \overline{z}_{SPP} - z_{DSPP} \tag{4.66}$$

Note that $w_i + p_i \leq u_i,\ \forall i \in V_c$ due to constraints (4.52). Furthermore, due to constraints (4.51)

$$c_l^k + F_k - \sum_{i \in R_l^k} u_i - v_k \geq 0$$

From which we obtain that $c_l^k \geq 0$. Therefore, with the same reasoning as in Theorem 2, we can conclude that a route $l \in \mathscr{R}^k,\ k \in M$, for which $\hat{c}_l^k > z_{UB} - z_{DSPP}$ can not belong to an optimal solution to (SPP). □

4.5.6. Procedure GENROUTE*

To be able to solve VRPs with distance constraints the set of feasible routes generated for problem (SPP) needs to satisfy the distance constraints. To attain this, some adjustments to procedure GENROUTE of Baldacci et al. [6] have been made. The adjusted procedure is called GENROUTE*.

Recall that the routes in procedure GENROUTE are constructed by combining paths $P \in \mathscr{P}$ generated during GENPATH. The procedure GENPATH uses a modified cost $\overline{c}^k_{ij}$. The modified costs for a path P in GENPATH are defined as

$$\overline{c}^k_l = \sum_{(i,j)\in E(P)} \overline{c}^k_{ij}$$

To obtain a good selection of paths in procedure GENPATH* the modified costs for a path are adjusted. In GENPATH* the modified costs for a path P are defined as follows

$$\overline{c}^k_l = \begin{cases} \sum_{(i,j)\in E(P)} \overline{c}^k_{ij}, & \text{if } \sum_{(i,j)\in E(P)} t^k_{ij} \leq L_k \\ \infty, & \text{otherwise} \end{cases}$$

In the procedure GENROUTE two paths $P, P' \in \mathscr{P}$ are combined to produce a route R. This route R will only be added to the final route set $\mathscr{B}^k$ if some conditions, described in Baldacci et al. [6] are satisfied. In GENROUTE*, the condition that route R should satisfy the distance constraint

$$\sum_{(i,j)\in E(R)} t^k_{ij} \leq L_k$$

is added to the list of conditions that are checked before adding the route R to the final route set $\mathscr{B}^k$.

4.5.7. Extended Clarke and Wright savings method

Both lower bound procedures, H^{1*} and H^{2*}, and procedure GENROUTE* need an upper bound z_{UB}. To compute this upper bound an extended version of the Clarke and Wright savings method, CWS* is used. A simple description of the Clarke and Wright savings method (CWS) was given in Subsection 2.2.1. The standard version of the CWS considers the capacitated vehicle routing problem. This section will give a description of the extended method CWS*.

CWS* is formulated such that it is able to deal with all problem types described in the Section 4.1 about the requirements. The method CWS* contains the following steps:

Step 1. **Initialize routes.** Just as for the normal savings method, CWS* starts with a route set R that contains an initial route $r_i = (0, i, 0)$ for each customer $i \in V_c$.

Step 2. **Create set of savings.** For each pair of customers $i, j \in V_c$ the savings $s_{ij} = d_{i0} + d_{0j} - d_{ij}$, for the distance is computed. Savings s_{ij} represent the value of the distance that is saved, when the route r_i, ending with customer i, and the route r_j, starting with customer j, are merged. The total set of savings $S = \{s_{ij} : i, j \in V_c\}$ is ordered from highest to lowest savings.

Step 3. **Merge routes.** Different from the standard capacitated VRPs, the problems considered for this exact method can have a different capacity Q_k for each vehicle type k. Therefore, this step is different from the standard CWS.

Let $\bar{k} \in M$ be the vehicle type with the largest capacity $Q_{\max}$ and let Q' be the largest capacity smaller than $Q_{\max}$. Furthermore, a variable A_k is defined to keep track of the number of vehicles of type k. To start with, set $A_k = U_k$ for each $k \in M$. While the size of route set $|R|$ is not equal to the total number of available vehicles and the set of savings S is not empty repeat the following:

(a) Let s_{ij} be the highest savings in set S. Let $q(r_i)$ and $q(r_j)$ be the demand of customer i and j respectively. If $q(r_i) + q(r_j) \leq Q_{\max}$, and i and j are connected to the depot, then:

- Merge the routes r_i and r_j.
- If $q(r_i)+q(r_j) \geq Q'$, then set $A_{\bar{k}} = A_{\bar{k}} - 1$, because this means for this route the largest vehicle must be used for this route. If $A_{\bar{k}} = 0$ then set $Q_{\max} = Q'$ and let $\bar{k} \in M$ be the vehicle type corresponding to the capacity $Q_{\max}$, also update Q' to be the largest capacity smaller than $Q_{\max}$.

(b) Set $S = S \backslash s_{ij}$.

Step 4. **Compute route lengths and assign routes to vehicles.** Next to a different capacity for each vehicle type k, the problems considered for this exact method can have a maximum route length L_k. The maximum route length could be considered in the previous step “Merge routes", however, it is chosen to do this in a separate step, otherwise the previous step would get very extensive. Furthermore, since different vehicle types are considered in the exact method the routes need to be assigned to specific vehicles. This is also different from the standard CWS algorithm, were the algorithm just produces a set of routes without considering which route belongs to which vehicle.

For each route $r \in R$ that is created, test if this route can be placed on a vehicle type k without exceeding the route length L_k and without exceeding the total number of available vehicles U_k. For each route $r \in R$ do the following:
Let $\bar{Q}$ be the smallest capacity of an available vehicle, for which $\bar{Q} \geq q(r)$ and let $\hat{k}$ be the corresponding vehicle type.

If the length of this route does not exceed the maximum allowed route length, i.e., $\sum_{(i,j)\in E(r)} t_{ij} \leq L_{\bar{k}}$, then this route will be placed on this vehicle, and this vehicle will no longer be available.

Otherwise, the next vehicle with the smallest capacity will be tested for their route length. If there are no vehicles left on which this route would fit, the orders on this route will be left unscheduled. This means these orders will add a penalty to the objective.

5

Computational results

This chapter presents the computational results of the exact methods described in the previous chapter. All methods have been implemented using C++ and are solved using the MIP solver of Gurobi Optimization [37]. Section 5.1 reports the results of the two-index flow formulation, Section 5.2 reports the results of the three-index flow formulation with penalties and Section 5.3 reports the results of the set partitioning formulation with penalties.

5.1. Results two-index flow formulation

To test the two-index flow implementations the CVRP instances of Uchoa et al. [78] are used. The name of a CVRP instance (for example “E-n13-k4") consist of: a letter (“E" in this example), which refers to the article in which this instance is published, a number behind the letter n (13 in this example), which refers to the number of vertices, and a number behind the letter k (4 in this example), which refers to the number of vehicles used for the problem.

In tables 5.2 and 5.3 the results of the implementations of the two-index flow formulation are shown. Table 5.2 shows the results of the implementation where all subtour constraints are constructed and Table 5.3 shows the results of the implementation where the subtour constraints are added according to the branch-and-bound method described in Section 2.1.3.

Both tables report the results of tests with and without an initial solution provided by the Clarke and Wright savings method (CWS), and with and without a variable number of vehicles. A variable number of vehicles implies the following. Let x be the value of the number behind k. The upper bound U to the number of vehicles used is now set to $U = x + 3$. The number of vehicles used m is set to $m \leq U$. When this is not the case the number of vehicles used m is set to $m = x$. Table 5.1 shows what each test shown in tables 5.2 and 5.3 indicates.

	Initial solution provided	Variable number of vehicles
Test 1	No	No
Test 2	Yes	No
Test 3	No	Yes
Test 4	Yes	Yes

Table 5.1: Definition of the tests shown in Table 5.2 and Table 5.3.

The columns in the tables represent the following:

T_{BS} : time in seconds spent to build the subsets.
T_{SC} : time in seconds spent to build the subtour constraints.
T_{GU} : time in seconds spent by Gurobi to solve the two-index flow problem.
z^* : cost of the best solution found by the exact method.
BKS : Best known solution in literature.

Note: In literature the distances between the vertices is often rounded. This explains the difference between the solutions provided by the exact methods and the best known solution in literature. The solutions provided by the exact methods are proven optimal, unless noted differently.

			Test 1		Test 2		Test 3		Test 4		
Instance	T_{BS}	T_{SC}	T_{GU}	z^*	T_{GU}	z^*	T_{GU}	z^*	T_{GU}	z^*	BKS
E-n13-k4	0,008	0,09	4,635	290	2,514	290	3,462	290	2,498	290	290
P-n16-k8	0,066	0,849	54,265	450	$\pm 54,265^a$	450^a	68,795	450	24,722	450	450
P-n19-k2	0,61	8,755	315,858	213	286,571	213	201,95	213	194,825	213	212

Table 5.2: Test results two-index flow formulation, of the implementation where all subtour constraints are constructed. Using the instances from Uchoa et al. [78].
[a] CWS could not find a feasible solution with the given amount of vehicles. This means no feasible initial solution is provided to Gurobi and therefore Test 2 is no different from Test 1.

Comparing the results of Test 1 with Test 2 in Table 5.2 one can see that providing an initial solution with CWS decreased the solution time for instances E-n13-k4 and P-n19-k2. However, for the branch-and-bound formulation the difference between the solution time or the final solution is not that significant and goes both ways. In both tables, 5.2 and 5.3, there are a few instances for which Test 2 provides the same result as Test 1. The reason for this is that CWS could not provide an initial solution that was feasible for the constraints of these instances. This can happen when CWS needs an extra route to be able to schedule all customers and satisfy the capacity constraints. This is the case for instance P-n16-k8. In this instance $m = 8$ (strict), but the solution provided by CWS needs 9 routes.

Increasing the number of available vehicles and making the number of used vehicles variable in Test 3 and 4 has a very positive effect. With the exception of Test 3 for P-n16-k8. This could be the effect of the solution space that is larger. Since the number of vehicles is increased the solution of CWS for instance P-n16-k8 is now feasible which resulted in a much faster solution time for Test 4 in Table 5.2.

In Table 5.3 one can see that once the SECs are dropped and added to the model according to the branch-and-bound method, Gurobi was able to solve problems with up to 40 customers. With all the SECs, this was not doable within a reasonable amount of time.

It is interesting to see that providing an initial solution using CWS, does not always result in a faster solution time for Gurobi in this implementation. An explanation for this could be that the initial solution starts at an inconvenient position in the branch-and-bound search tree, which results in more nodes to be visited in the tree.

	Test 1		Test 2		Test 3		Test 4		
Instance	T_{GU}	z^*	T_{GU}	z^*	T_{GU}	z^*	T_{GU}	z^*	BKS
E-n13-k4	0,4	290	0,2	290	0,1	290	0,1	290	290
P-n16-k8	0,2	450	$\pm 0,2^a$	450^a	0,3	450	0,8	450	450
P-n19-k2	0,1	213	0,2	213	0,1	213	0,1	213	212
E-n23-k3	0,04	569	0,1	569	0,2	569	0,6	569	569
E-n30-k3	1083	536	$\pm 1083^a$	536^a	0,3	505^b	0,8	505^b	534
B-n31-k5	> 900	679^c	> 900	679^c	337,2	676	360,2	676	672
E-n33-k4	> 900	877^c	> 900	841^c	> 900	916^c	> 900	843^c	835
A-n38-k5	> 900	741^c	$> 900^a$	$\pm 741^a$	651,4	734	> 900	735^c	730
P-n40-k5	> 900	467^c	> 900	479^c	847,5	462	> 900	479^c	458

Table 5.3: Test results two-index flow formulation: Branch-and-bound implementation. Using instances from Uchoa et al. [78].
[a] CWS could not find a feasible solution with the given amount of vehicles. This means no feasible solution is provided to Gurobi and therefore Test 2 is no different from Test 1.
[b] An extra vehicle is used for this solution.
[c] The time limit of Gurobi is reached and therefore this solution is not (proven to be) optimal. Gurobi was given a time limit of 900 seconds.

5.2. Results three-index flow formulation (FP)

A few different HVRP instances are created to demonstrate that the three-index flow formulation with penalties is able to correctly solve them. The created instances are reported in Appendix A. The results are shown in Table 5.4. The columns of Table 5.4 report the following:

T_{GU} : time in seconds spent by Gurobi to solve the three-index flow problem.
z^* : cost of the best solution found by the exact method.
NS : number of orders that are not scheduled in the solution giving costs z_{UB}
BKS : Best known solution, allowing for unscheduled orders. These solutions are reported in this thesis since these instances are specially created for this thesis.

Each instance is solved with the three-index flow formulation with penalties (FP), and also with the three-index flow formulation where all customers need to be visited ($F3$), i.e. with Constraint (4.13):

$$\sum_{k=1}^{U} y_i^k = 1 \quad \forall i \in V_c$$

instead of Constraint (4.23). The formulation ($F3$) is considered to compare the computational results of the formulation with and without visiting all customers and to see how easily Gurobi can determine that a problem is infeasible if all customers have to be visited.

	Three-index flow visit all ($F3$)		**Three-index flow with penalties** (FP)			
Instance	T_{GU}	z^*	T_{GU}	z^*	NS	BKS
HVRP-E-n13-k4-t1	158	290	184	290		290
HVRP-E-n13-k6-t1	x	x	9510	290		290
HVRP-E-n13-k4-t1-LD	0	infeasible	44,5	340	1	340
HVRP-E-n13-k4-t1-TL	450,8	302	265,6	302		302
HVRP-E-n13-k4-t1-LC	49,1	170	37,9	170		170
HVRP-E-n13-k4-t4	14,54	319	8,75	319		319
HVRP-E-n13-k4-t4-SC	0,1	infeasible	8,6	456	1	456
HVRP-E-n13-k4-t2-SL	475	infeasible	79,1	418	2	418
HVRP-E-n13-k2-t2-C	0	infeasible	1,1	554	4	554
HVRP-E-n13-k5-t2-FC	> 900	310[a]	> 900	310[a]		310
HVRP-P-n16-k8-t8	> 900	498[a]	> 900	498[a]		498
HVRP-P-n16-k8-t8-TF	0	infeasible	> 900	763[a]		763
HVRP-P-n19-k2-t1	122,1	213	67,7	213		213
HVRP-P-n19-k3-t1	> 900	220[a]	> 900	213[a]		213
HVRP-P-n23-k18-t4	> 900	656[a]	> 900	652[a]		570

Table 5.4: Test results three-index flow formulation. The instances are reported in Appendix A.
[a] The time limit of Gurobi is reached and therefore this solution is not (proven to be) optimal.

As you might notice, the name of the HVRP instances is derived from the CVRP instances on which they are based. The letters HVRP are added to the front and a letter "t", with a value that indicates the number of vehicle types that are available is added. Furthermore, some of the names also include two extra letters to indicate a specific instance, see Appendix A.

The instances that are reported in Table 5.4 are selected, because each of these instances consider a different aspect of the problem types that can be solved with formulation (FP). In the subsections below the aspects that these instances consider are shortly discussed.

Simple instance

The first instance, HVRP-E-n13-k4-t1, is actually the same problem as the CVRP instance E-n13-k4, but now formulated as an HVRP with extra properties for the vehicles, such as a maximum route length and different type of costs. This is a nice way to validate that the outcome is the same for both instances. The distance matrix in the HVRP instance is copied from the distance matrix from the CVRP instance and the time matrix is set equal to the distance matrix. To get the same edge cost as for the CVRP instance let the cost $c_d^k = c_t^k = 0,5$. The maximum route length $L_k := 300$ and the fixed cost $F_k := 0$. Since the total costs of E-n13-k4 is 290, and is equal to the total traversed distance, such a high value for L_k will not have any influence to the solution. Furthermore the penalty p for not visiting a customer is set equal to

$$p = \max_{k \in M}\{F_k\} + \max_{k \in M}\left\{\max_{j \in V_c}\{c_{0j}^k + c_{j0}^k\}\right\} = 104.$$

In Table 5.4 it can be seen that solving HVRP-E-n13-k4-t1 using formulation ($F3$) or (FP) takes much more time than solving the CVRP instance E-n13-k4 with the two-index flow formulation. When two more vehicles are added in instance HVRP-E-n13-k6-t1 the solution time explodes. For this reason this instance is not also tested for formulation ($F3$).

Instance with too large demand

The instance HVRP-E-n13-k4-t1-LD is a copy of instance HVRP-E-n13-k4-t1, where one of the customers has a demand higher than the capacity, i.e., $q_i > Q_k$. The results show that formulation (FP) can indeed solve this type of problem. It also shows that, when using formulation ($F3$) for this problem type, Gurobi can tell that this is infeasible within a fraction of second.

Instance with vehicles with a tight route length

For instance HVRP-E-n13-k4-t1-TL the maximum route length of the vehicles is set to a more tight value, i.e. $L_k = 110,\ \forall k \in M$. This makes it more difficult to find a feasible solution and the solution found for instance HVRP-E-n13-k4-t1 is not feasible for this instance. The results show that both formulations can solve this type of problem, but that with formulation ($F3$) Gurobi needs more time to solve the problem.

Instance with vehicles with large capacity

In instance HVRP-E-n13-k4-t1-LC the capacity of the vehicles is set to a very large value, i.e. $Q_k = 15000,\ \forall k \in M$. This makes the objective value decrease to 170 and the solution time of Gurobi decrease to 37,9 seconds.

Instance with heterogeneous vehicles

Instance HVRP-E-n13-k4-t4 is created to test the formulation with different types of vehicles. The vehicles in this instance all have a different capacity, maximum route length, and fixed cost. The values for the capacity and route length are quite tight, which explains why Gurobi was able to solve this problem so fast.

Instance with not enough capacity

In instance HVRP-E-n13-k4-t4-SC the total capacity is too small to fit the total demand. As can be seen in Table 5.4 Gurobi was able to solve this problem and find out which customer should not be visited, using formulation (FP). Furthermore, using formulation ($F3$) Gurobi could find out this problem is infeasible within 0,1 seconds.

Instance with not enough route length

Instance HVRP-E-n13-k4-t2-SL contains two vehicle types, and in total four vehicles. In this instance it is not possible to visit all customers while satisfying the maximum route length. In the results in Table 5.4 it can be seen that this problem was solved within 79,1 seconds by Gurobi using formulation (FP), and that for the optimal solution two customers (9 and 11) are not visited. It is interesting to see that when formulation ($F3$) is used, Gurobi needs a lot more time to find out that this problem is infeasible, compared to the previous infeasible problems, but also compared to finding the solution with formulation (FP).

Instance with not enough capacity

Instance HVRP-E-n13-k2-t2-C is similar to instance HVRP-E-n13-k4-t1, but with only two vehicles. This means there is far too little capacity for the total demand and that multiple customer orders will have to be excluded in the final solution. As can be seen in Table 5.4, this resulted in excluding a total of four customers, which where customer 8,9,11 and 12.

Instance with different fixed costs

Instance HVRP-E-n13-k5-t2-FC is also similar to instance HVRP-E-n13-k4-t1, but with four vehicles with relative low fixed cost, i.e., $F_k = 5$, and one vehicle with relative high fixed costs, i.e. $F_k = 50$. This was to verify if the solution would contain the right vehicles. As you can see by the value of the objective Gurobi found the right solution ($290 + 4 * 5 = 310$), but Gurobi was not able to prove that this solution was optimal within the time limit of 900 seconds.

Some larger instances

To test the performance of formulation (FP) Gurobi on other larger instances, instances HVRP-P-n16-k8-t8, HVRP-P-n19-k2-t1 and HVRP-P-n23-k18-t4 are created and tested. As expected solving instance HVRP-P-n16-k8-t8 and HVRP-P-n23-k18-t4 to optimality takes a very long time. Gurobi is not able to find an optimal solution or to prove that the solution found is optimal within the given time limit of 900 seconds. It is interesting to see that instance HVRP-P-n19-k2-t1 can be solved relatively fast, since this instance contains 18 customers. Nevertheless, this instance only contains 2 vehicles and from Section 4.3.4 it is known that the number of constraints highly depends on the number of vehicles. When the number of vehicles is increased to three, it takes a lot more time to solve the instance and Gurobi is not able to prove the solutions optimality.

Instance with one customer too far

In instance HVRP-P-n16-k8-t8-TF the coordinates of one of the customers are changed to $(x_i, y_i) = (200, 200)$, such that it is not possible to visit that customer given the maximum route length for each vehicle. In Table 5.4 it can be seen that with formulation (FP) a (not necessarily optimal) solution is found, and that with formulation ($F3$) the infeasibility of the problem is established within 0 seconds.

5.3. Results set partitioning formulation with penalties

This section presents the computational results of the exact method described in Section 4.5.1, using the set partitioning formulation with penalties, (SPP). The exact method to solve (SPP) consists of four main steps:

Step 1. Compute lower bound $LH1$ using bounding procedure H^{1*} (Section 4.5.3).

Step 2. Compute lower bound $LH2$ using bounding procedure H^{2*} (Section 4.5.4).

Step 3. Generating the reduced sets $\hat{\mathscr{R}}^k$ using the GENROUTE* procedure (Section 4.5.6).

Step 4. Solving the reduced problem $\widehat{SPP}$ using Gurobi Optimization [37].

The results of each step are reported in Subsection 5.3.2. Furthermore, Section 5.3.1 describes the parameter settings for each procedure used in the exact method and will give some remarks about the implementation.

5.3.1. Implementation

The parameter values for procedures H^{1*} and H^{2*} are mostly defined as in Baldacci and Mingozzi [5], who have done several experiments to identify good parameter settings. In Subsection 6.1.4 the values of the parameters are discussed. Procedure H^{1*} uses the following parameter settings

- $\epsilon = 2$,
- $Maxt1 = 100$.

Procedure H^{2*} uses the following parameter settings:

- $\epsilon = 1$,
- $Maxt2 = 25$,
- $Maxt3 = 400$,
- $\Delta^{\min} = 500$,
- $\Delta^a = \min\{10n \cdot it, 100n\}$, where n is the number of vertices and it, $0 < it \leq 25$, is the number of the iteration.

Recall, for the GENROUTE* procedure a parameter γ and a parameter Δ are provided to create the largest subset $\mathscr{B}^k \subseteq \mathscr{R}^k$ for which:

(a) $\max_{l \in \mathscr{B}^k}\{c_l^k\} \leq \min_{l \in \mathscr{R}^k \setminus \mathscr{B}^k}\{\hat{c}_l^k\}$

(b) $|\mathscr{B}^k| \leq \Delta$

(c) $\max_{l \in \mathscr{B}^k}\{\hat{c}_l^k\} \leq \gamma$

Procedure GENROUTE* is used in H^{2*} to generate the sets $\overline{\mathscr{R}}^k$ and $\mathscr{N}^k$, and at the end of the exact method to generate the reduced sets $\hat{\mathscr{R}}^k$, for each vehicle type $k \in M$. For the procedure GENROUTE* the following parameter settings for γ and Δ are used:

- To generate $\overline{\mathscr{R}}^k$:
  $\gamma = HUGE\_VAL, \quad \Delta = \Delta^{\min}$.
  Where $HUGE\_VAL$ is the maximum value possible for a double in C++, hence this is used for the value of ∞.

- To generate $\mathscr{N}^k$:
  $\gamma = -1 \cdot 10^{-6}, \quad \Delta = \Delta^a$.
  Instead of $\gamma = 0$ the parameter γ is set equal to just below zero. This is done because otherwise the set $\mathscr{N}^k$ was often generating routes that already belonged to $\overline{\mathscr{R}}^k$.

- To generate $\hat{\mathscr{R}}^k$:
  $\gamma = z_{UB} - \max\{LH1, LH2\} + 1 \cdot 10^{-8}, \quad \Delta = 100.000$.
  A small value $1 \cdot 10^{-8}$ is added to the value of γ, to avoid rejecting routes because of numerical errors. Without adding this value some routes where rejected, which should actually be accepted. Furthermore, the value of Δ is set to 100.000 instead of ∞, to be able to run the tests within a manageable amount of time. If this limit is reached the optimality of the solution can not be proven.

5.3.2. Results

To validate the capabilities for the set partitioning formulation with penalties (SPP), most of the instances that where also used for the three-index flow formulation with penalties (FP) are used. The results are presented in Table 5.5. The columns of Table 5.5 report the following:

z_{UB} :	value of the upper bound computed to use for the execution of the exact method.
$LH1$:	lower bound $LH1$ found by procedure H^{1*}.
T_{H1} :	time in seconds spent by bounding procedure H^{1*}.
$LH2$:	lower bound $LH2$ found by procedure H^{2*}.
T_{H2} :	time in seconds spent by bounding procedure H^{2*}.
$\lvert\hat{\mathscr{R}}\rvert$:	size of the total set of generated routes $\hat{\mathscr{R}} = \bigcup_{k \in M} \hat{\mathscr{R}}^k$.
T_{GU} :	time in seconds spent by Gurobi to solve the reduced problem $\widehat{SPP}$ of (SPP).
T_{ex} :	total computing time in seconds for the complete exact method. This includes the bounding procedures, the procedure GENROUTE and solving the reduced problem with Gurobi.
z^* :	cost of the best solution found by the exact method.
NS :	number of orders that are not scheduled in the solution giving costs z^*.
BKS :	Best known solution, allowing for unscheduled orders. These solutions are reported in this thesis since these instances are specially created for this thesis.

As can be seen in Table 5.5 the results of the heuristic method CWS*, described in Subection 4.5.7, to compute the upper bound z_{UB} were not always very accurate. Since RitOpt can already provide good feasible solutions and thus good upper bounds for these type of problems, this research has not focussed on creating a very good upper bound. Nevertheless, the bounding procedures would work better with a tighter upper bound and in addition GENROUTE* can provide a better selection of routes when the gap between the upper bound and the lower bound is smaller.

Instance	z_{UB}	$LH1$	T_{H1}	$LH2$	T_{H2}	$\|\mathscr{R}\|$	T_{GU}	T_{ex}	z^*	NS	BKS
HVRP-E-n13-k4-t1	290	215	0,417	260	1,64	266	0,288	2,39	290		290
HVRP-E-n13-k4-t1-LD	484	300	0,281	316	1,183	342	0,009	1,525	340	1	340
HVRP-E-n13-k4-t1-TL	706	215	0,409	226	1,299	369	0,09	1,871	302		302
HVRP-E-n13-k4-t1-LC	218	129	3,761	155	7,469	3865	0,293	106,882	170		170
HVRP-E-n13-k4-t4	319	318	1,168	319	2,741	4	0,001	3,957	319		319
HVRP-E-n13-k4-t4-SC	940	252	0,837	252	2,174	699	0,02	3,112	456	1	456
HVRP-E-n13-k20-t4	319	288	1,217	290	2,263	368	0,021	3,561	298		298
HVRP-E-n13-k20-t4-FC	339	292	1,138	296	2,278	560	0,042	4,668	316		316
HVRP-E-n13-k4-t2-SL	1018	215	0,825	296	1,532	404	0,015	2,431	418	2	418
HVRP-E-n13-k2-t2-C	1018	215	0,917	306	3,793	1072	0,025	4,933	554	4	554
HVRP-E-n13-k5-t2-FC	378	231	0,888	262	3,117	1072	0,31	5,346	310		310
HVRP-P-n16-k8-t8	781	406	2,222	406	3,492	1248	0,07	5,93	498		498
HVRP-P-n16-k8-t8-TF	1375	669	1,415	669	3,043	954	0,15	4,699	763	1	763
HVRP-P-n19-k2-t1	238	148	7,532	200	190,558	x	x	>11hours	x		213
HVRP-P-n23-k18-t4	1121,94	439	2,549	456	13,296	10812	1,62	41,225	570		570

Table 5.5: Computational results for different type of instances with the set partitioning formulation with penalties (SPP). The instances are all reported in Appendix A.

Compare results with results formulation (FP)

The results of formulation (SPP) represented in the last three columns of Table 5.5 are compared with the results of formulation (FP) in Table 5.4. It can be seen that formulation (SPP) gives the same objective value, z^*, for almost all instances that where also tested using formulation (FP). For instance HVRP-P-n23-k4, the (SPP) formulation provides an even better one. Furthermore, formulation (SPP) is able to solve nearly all problems reported in Table 5.5 to optimality.

For most of the instances the total computation time for the exact method using formulation (SPP), T_{ex}, is smaller then the time used to solve the problem using formulation (FP). However, for instances where the capacity of the vehicles is relatively large, i.e., many orders fit on a vehicle, such as instance HVRP-E-n13-k4-t1-LC and HVRP-P-n19-k2-t1, the total computation time of formulation (SPP) is a lot higher. Instance HVRP-P-n19-k2-t1 is not even solved by the exact method using formulation (SPP), because it took to much time to generate the routes (even when the number of generated routes was limited to 100.000).

The fact that the total computation time for the exact method using formulation (SPP) is longer for problems with vehicles with a large capacity can be explained as follows. When the capacity is relatively large, compared to the size of the orders, the number of possible sets of orders that can be placed on the vehicle becomes very large (the set W_k in Expression (4.38)). As result the q-route procedure of H^{1*} and the GENROUTE procedure become very time consuming. One can see that the size of the reduce route set $\hat{\mathscr{R}}^k$ for instance HVRP-E-n13-k4-t1-LC is very large.

Instance with many vehicles

One may notice that the instance HVRP-E-n13-k6-t1, reported in Table 5.4, is not reported in Table 5.5. This is because using six vehicles instead of four barely influences the computation time when using formulation (SPP). For this reason, a new instance is created, HVRP-E-n13-k20-t4, that has a total of twenty vehicles. In Table 5.5 it can be seen that increasing the number of vehicles does not have a negative influence on the computation time.

Furthermore, an instance with twenty vehicles and where all vehicle types have fixed costs is created to validate that the reduction step that is executed at the end of procedure H^1 works. In this step the upper bound, U_k, for the number of vehicles of a vehicle type k can be reduced. This can make it easier to solve the problem. This step changed the upper bound to the number of vehicles of type $k2$ to $U_{k2} = 3$ instead of 5. Which shows that this step works. Nevertheless, in for this instance it did not positively influence the solution time, probably because the instance was too small.

Testing larger instances

The instances of EVO-it start with sizes of in the range of 30 orders. To demonstrate the effectiveness of the exact method on these problem sizes, a few larger HVRP instances are created. The instances are all reported in the Appendix A. The results of the instances are reported in Table 5.6. The best known solution for the instances is the one reported in Table 5.6, since these instances are created for this thesis. Therefore, Table 5.6 does not contain an extra column to report the best known solution. Instead a column with the gap between the upper bound and the lower bound (in Gurobi) is added, for the problems that were not solved to optimality (with respect to the routes in $|\hat{\mathscr{R}}|$) because the time limit for Gurobi was reached.

Instance	z_{UB}	$LH1$	T_{H1}	$LH2$	T_{H2}	$\lvert\hat{\mathscr{R}}\rvert$	T_{GU}	T_{ex}	z^*	NS	gap
HVRP-E-n30-k18-t4	8722	1871	2,9	-6e26	63,1	9804	3	101	4624	5	
HVRP-B-n31-k7-t2	908	658	7,5	816	275,4	> 122000	> 18000	> 26583	908^{ab}		4,3%
HVRP-E-n33-k10-t2	2190	1481	20,4	1663	1968	> 200000	> 18000	> 88241	1842^{ab}		2,5%
HVRP-A-n33-k15-t3	1286	839	11,1	998	294,4	> 202139	10,3	5495	1039^{a}		
HVRP-A-n38-k5-t2	10841	952	11,5	-2e46	5472	> 110455	0,2	19531	5701^{a}	6	
HVRP-P-n40-k18-t4	4265	2318	19,2	2402	86,8	32844	2,4	372,9	2943	2	

Table 5.6: Computational results for instances with 30 or more orders. The instances are all reported in A.
[a] The generated route set $\hat{\mathscr{R}}^k$ reached the limit of 100.000 for one of the vehicle types k. Therefore the optimality of the solution is not proven.
[b] The time limit of Gurobi is reached and therefore the optimality of the solution is not proven.

What strikes the eye is that the lower bound $LH2$ of instances HVRP-E-n30-k2 and HVRP-A-n38-k2 is very low. More important, the lower bound $LH2$ is lower than $LH1$, which is not possible in the exact method of Baldacci and Mingozzi [5]. An explanation for this can be the subgradient optimization step that has not been adjusted to the new formulation (SPP). The penalties $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ are now modified exactly as done in the article of Baldacci and Mingozzi [5], with the assumption that each customer should be visited once in the final solution. Since this is not possible for the instances HVRP-E-n30-k2 and HVRP-A-n38-k2 this can result in modifying the penalties $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ in the wrong direction.

For most instances reported in Table 5.6 the size of the generated route set $\hat{\mathscr{R}}^k$ reached its limit of 100.000 for some $k \in M$. Therefore, none of the solutions to these instances is proven to be optimal. The size of the generated route set $\hat{\mathscr{R}}^k$ is so large, because the capacity of the vehicles of type k is relatively large compared to the size of the orders. The size of $\hat{\mathscr{R}}^k$ can be reduced by reducing the gap between the upper and the lower bound.

Two of the instances, i.e. HVRP-E-n30-k2 and HVRP-P-40-k18-t4, reported in Table 5.6 are solved to optimality. Where in the solution of instance HVRP-E-n30-k2 there are no more than five orders per vehicle route and in the solution of HVRP-P-40-k18-t4 a route contains no more than four orders.

What the results point out is that the number of orders or the number of vehicles is both not necessarily an issue with this formulation. However, if the number of orders to fit on a vehicle is large, then this method gets difficulties solving the instance.

6

Discussion and recommendations

Chapter 4 described two exact methods to solve heterogeneous vehicle routing problems dealing with a variable number of capacitated vehicles, distance constraints, fixed and variable costs, and problems where not all orders need to be scheduled. Subsequently, Chapter 5 presented the results of these methods. This chapter will re-evaluate the methods that have been used in Chapter 4 and discuss the results reported in Chapter 5.

Furthermore, this chapter will describe multiple recommendations for future work, which will be summarized at the end of this chapter.

6.1. Discussion

This section will discuss the effects of the decisions made in this thesis, such as simplifications and considered instances. Furthermore, some remarks on formulation (SPP) will be discussed. Finally the results of the three-index flow formulation with penalties (FP) and the set partitioning formulation with penalties (SPP) will be compared.

6.1.1. Simplifications

The simplifications made in this thesis are the selected requirements, the symmetry of the matrices and not having the possibility of multiple trips. These are all discussed in the paragraphs below.

Selected requirements

To simplify the instances at EVO-it a selection of requirements has been made in Section 4.1. This makes the problem more manageable for an exact method, which is, as known from Section 2.5, usually not a very flexible method.

Nevertheless, restricting the method to this selection makes the method less applicable for other problems in real-life. If one wants to solve real-life vehicle routing problems with attributes that are not covered by the exact method in this thesis, such as time windows or multiple trips, there are two options. 1. One could try to extend one of the exact methods provided in this thesis, such that it is also able to handle problems with other attributes, such as time windows or multiple trips. 2. One can search for other exact methods, for example Hernandez et al. [38], that would be a better fit for solving problems with time windows and multiple trips. It could be that extending the exact method described in this thesis becomes more difficult than using a complete different exact method for these problems. Thus if CQM would be interested in solving other type of instances of EVO-it to optimality it is a good idea to also consider other exact formulations.

Symmetry of matrices

The exact method that uses formulation (SPP) is assuming a symmetry in the cost, distance and time matrices. Symmetry is used to build the q-routes from q-paths and to build the routes in GENROUTE using the paths from GENPATH. However, in real instances symmetry is certainly not guaranteed. This simplification of the instance can result in a solution that might not be optimal for the actual problem. To solve this issue one should consider the paths in both directions, thus from the depot towards a vertex i and from vertex i towards the depot i. This will however include much more computations and therefore more time.

A positive point about the three-index flow formulation considered in this thesis is that this formulation makes use of arcs instead of edges and can therefore take into asymmetric matrices.

Multiple trips are not considered

None of the exact methods that have been used in this thesis project have the possibility of multiple trips with one vehicle. This simplification can also result in a solution that might not be optimal in a real-life instance, where multiple trips are allowed.

6.1.2. Implementation of the exact methods

The exact methods have been fully implemented, tested and verified with different test scenarios and results from literature. The focus has been to demonstrate that the method works and point out criticalities and anomalies, but not so much in making the software as neat or efficiently coded as potentially possible. The time required to find solutions can as such be decreased even more. Therefore, the computation times reported in the results are not comparable with results in literature.

Using the knowledge of someone with a programming background to improve the procedures could make a difference in the solution time and is recommended if one wants to compare the results to other well performing methods.

6.1.3. Selected test instances

Even though the tests in this thesis consider many different type of instances, there are a few instances mentioned below that were not included in the tests of this thesis.

Infinite capacity

In all of the tested instances the vehicles have a maximum capacity of Q_k. However, in the data of the real-life instances of EVO-it it appeared that some instances (23%) do not have a maximum capacity. This is something that has not been tested by the exact methods described in Section 2.1. An instance where the vehicles do not have a maximum capacity can be created by setting the maximum capacity Q_k equal to infinity for each vehicle type k. Solving such a problem is expected to be significantly more difficult for the set partitioning formulation (SPP), since this formulation has trouble with problems with a large capacity. It is expected though, that the three-index flow formulation will not necessarily have an issue solving problems with infinite capacity, since the number of constraints does not increase.

Note that defining a problem with infinite capacity for the two-index flow formulation will result in a TSP.

Other problem types

Section 4.3.3 describes the different types of problems that are covered by the exact methods using the three-index flow and the set partitioning formulation with penalties. In this section the site dependent VRP and the multi-depot VRP are also listed as vehicle routing problems that can be solved with these methods. However, these were not tested. The reason for this is that for both cases this is a matter of rewriting the instance, which can be done before the exact method starts.

Defining site dependent and multi-depot problems, and subsequently rewriting the problems such that it these can be solved by the exact methods, contains additional programming work. Because of limited time it has been chosen not to include this in this project.

Testing real-life instances from EVO-it

Even though there might exist real-life instances at EVO-it for which (a simplified version of) the vehicle routing problem can be solved with the exact methods described in Chapter 4, this has not been tested. The reason for this is that the instance of EVO-it would first have to be formulated in the structure such that the program that was build to test the exact methods can read the instance of EVO-it. This would also contain additional programming work, which is because of limited time not included in the project, but this would be a very interesting step for CQM.

6.1.4. Remarks on formulation (SPP)

This section will point out some remarks on the parameters that have been used, on the procedure GENROUTE and on the subgradient optimization step used in the lower bound procedures.

Parameters

In the lower bound procedure H^{1*} the parameters $Maxt1$ and ϵ are used and lower bound procedure H^{2*} uses the parameters $Maxt2$, $Maxt3$, ϵ, $\Delta^{\min}$ and Δ^a. These parameters are mostly defined as in Baldacci and Mingozzi [5], who have done several experiments to identify good parameter settings. However, the values of $\Delta^{\min}$ and Δ^a are adjusted to different values, the reason for this is explained in the next paragraph.

This research did not do a well funded sensitivity analysis on the influences of the parameters for different instances. Therefore, it is recommended to do more experimenting with different parameters to find out which parameter settings work best.

Incomplete set of routes generated by GENROUTE

The reason that $\Delta^{\min}$ and Δ^a have been adjusted is because it was noted that in some cases the procedure H^{2*} was accepting a lower bound, which is incorrect, i.e. higher than the upper bound.

Recall from Section 4.4.4, that in Step 2. of procedure H^2 (and also for procedure H^{2*}) the solution constructed in Step 1 is checked for its feasibility for problem (DSP) by generating the set $\mathscr{N}^k \subset \mathscr{R}^k \backslash \overline{\mathscr{R}}^k$. If the set $\mathscr{N}^k = \emptyset$ for all $k \in M$, it is suggested that there are no routes with negative reduced cost with respect to the solution computed in Step 1 and the solution is accepted.

During the tests it appeared that when using the parameters:

- $\Delta^{\min} = 500$
- $\Delta^a = 200$

There were instances for which the set $\mathscr{N}^k$ was empty, for each $k \in M$, even though the solution could not be a feasible (DSP) solution. Recall that the set $\mathscr{N}^k$ is generated by GENROUTE. The reason that GENROUTE generated an empty set of routes is that the paths generated by GENPATH could not be combined into a feasible route, even though there actually did exist routes with negative reduced costs. This is explained in detail below.

The procedure GENROUTE is said to create the largest subset $\mathscr{B}^k \subseteq \mathscr{R}^k$ for which:

(a) $\max_{l \in \mathscr{B}^k}\{c_l^k\} \leq \min_{l \in \mathscr{R}^k \backslash \mathscr{B}^k}\{\hat{c}_l^k\}$

(b) $|\mathscr{B}^k| \leq \Delta$

(c) $\max_{l \in \mathscr{B}^k}\{\hat{c}_l^k\} \leq \gamma$

In the article of Baldacci et al. [6] it is stated that the set $\mathscr{B}^k$ in procedure GENROUTE can be generated using only the paths P from the largest simple path set $\mathscr{P}$ satisfying the following conditions:

(a) $\max_{P \in \mathscr{P}}\{LB(P)\} \leq \min_{P \notin \mathscr{P}}\{LB(P), \gamma\}$

(b) $|\mathscr{P}| \leq \Delta$

(c) $q(P) \leq \frac{Q}{2} + q_{e(P)}$, $\forall P \in \mathscr{P}$, where $e(P)$ is the last vertex in the path P.

(d) a path $P \in \mathscr{P}$ cannot be dominated by another path $P' \in \mathscr{P}$.

During the computational tests of this thesis report it seemed that this statement is not correct.

When Δ is not large enough it can occur that there is a path $P \notin \mathscr{P}$ for which $LB(P) \leq \gamma$. For this reason it could be that a route R that should belong to the final set of routes $\mathscr{B}^k$ contains this path P. However, as a result of condition (b) this path P and this route R will not be generated. Which can make the final set incomplete.

The procedure GENPATH generates a set of Δ paths with the lowest value $LB(P)$. But this does not necessarily mean that the generated paths can also be combined into a route. Suppose path P and path P' are the paths with the lowest $LB(P)$, but $e(P) \neq e(P')$. This means that these paths can not be combined to construct a route. In addition, there may be other reasons that path P and path P' can not be combined to construct a route, such as exceeding the limit of the capacity, or the paths P and P' are not internally disjoint.

For these reasons, it is not guaranteed that GENPATH will generate paths that can construct any route satisfying the conditions for route set $\mathscr{B}^k$, even though there do exists paths $P \notin \mathscr{P}$ for which $LB(P) \leq \gamma$ that

could construct a route R that would satisfy all of the constraints for route set $\mathscr{B}^k$. This leads to an empty set $\mathscr{B}^k$, while there actually still exists routes that would satisfy the conditions (a), (b) and (c) for route set $\mathscr{B}^k$.

Nevertheless restricting the size of the path set $\mathscr{P}$ to be smaller than a certain value Δ is useful, because otherwise this set can get very large. A suggestion to solve this problem is to use a larger parameter for the path set, for example $\Delta_{\mathscr{P}} = x\Delta$, where $x > 1$ is a user defined parameter.

After correspondence with the authors prof Baldacci and prof. Mingozzi, they acknowledge the above, and remark the importance that the parameter Δ is large enough.

Subgradient optimization

Both lower bound methods H^{1*} and H^{2*} make use of subgradient optimization. The subgradient optimization step that is used for H^{1*} and H^{2*} is directly copied from the subgradient optimization step from Baldacci and Mingozzi [5] used for H^1 and H^2.

The solutions that are constructed using this method are feasible. However, the subgradient step that is used is formulated for formulation (ESP) where all customers have to be visited as opposed to formulation (SPP) where this is not the case. Therefore, this subgradient optimization step might not lead to a convergence towards better solutions for H^{1*} or H^{2*}. This could be an explanation for the strange values for the lower bound $LH2$ for instances HVRP-E-n30-k2 and HVRP-A-n38-k2 in Table 5.6, which both have a large number of unscheduled orders.

Subgradient optimization has not been investigated in this thesis project, but it is highly recommended as a next step in the research for solving formulation (SPP).

6.1.5. Three-index flow versus set partitioning formulation

Before the computational tests of the three-index flow and the set partitioning formulation with penalties, (FP) and (SPP) respectively, were performed, it was expected that formulation (SPP) would perform much better, i.e. have a faster computation time. However, when comparing the results from Table 5.4 and Table 5.5 it can be seen that for instances with a relatively large capacity (many orders per vehicle) and a small amount of vehicles this is not the case. The performance of the formulation (SPP) is mainly decelerated by the large amount of routes that it generates when creating the set $\hat{\mathscr{R}}$. Improving the upper bound and the lower bound will result in a smaller set $\hat{\mathscr{R}}$ and will therefore improve the computation time for the exact method using formulation (SPP).

When the number of vehicles is increased, the computation time to find a proven optimal solution for formulation (FP) became, as expected, much higher. This is caused by the number of constraints, which increases as the number of vehicles (or the number of customers) increases. It might be possible to add additional cuts to the LP-relaxation similar to the ones described for the two-index flow formulation mentioned in Subsection 2.1.4 to improve the computation time of formulation (FP). However, when looking at the history of the three-index flow formulations (e.g. [27]) this formulation has not been very successful.

What can be concluded is that both methods have their advantages and disadvantages. The three-index flow formulation (FP) works better for cases with large capacity, with the restriction that the number of vehicles is not too large, and the set partitioning formulation (SPP) works better when the number of vehicles is high, but the capacity is not too large. Though it seems that improving the exact method using formulation (SPP) has a bigger chance of success when looking at previous successes of other researchers.

6.2. Future work exact method

A few points for future work are already mentioned in the discussion section above. This section will provide some more future work steps for the exact method using formulation (SPP).

6.2.1. Use ng-routes instead of q-routes

Procedure H^{1*} of the exact method using formulation (SPP) uses a relaxation based on q-routes. In later article of Baldacci et al. [7] this relaxation has been improved with a ng-routes relaxation.

This is something that would be interesting to implement in future research.

6.2.2. Include the other bounding methods

The method of Baldacci and Mingozzi [5] uses five bounding procedures to attain a final best lower bound. The exact method using formulation (SPP) uses only the first two lower bound procedures. Including the

other bounding methods can result in a better lower bound and therefore a smaller set $\hat{\mathscr{R}}$, which will make it easier to solve larger problems or problems where the capacity is relatively large.

6.2.3. Improve the upper bound

The exact method described in Section 4.5.1 uses an upper bound constructed by an extended version of the Clarke and Wright savings method, CWS*. In the results of Section 5.3 it can be seen that this method does not always give a very accurate upper bound. The consequence is that the bounding procedures H^{1*} and H^{2*} and procedure GENROUTE are performing not as well as they could. Improving this upper bound will therefore probably have a positive effect on the performance of the exact method.

To attain a better upper bound one can use local search methods to improve the solution provided by CWS*.

If CQM wants to provide a better upper bound it is possible to use RitOpt, since it provides a feasible solution and therefore an upper bound to the problem.

6.3. Recommendations for CQM

6.3.1. Connection between RitOpt and the exact method

The goal of this research was to provide an exact method such that the results of RitOpt can be compared to the results of an exact solution. This thesis does provide an exact method, however the connection between the exact method and the problems of RitOpt, thus practical instances of EVO-it, is still missing. This was already mentioned in Subsection 6.1.3, but the importance of this step is emphasized again in this section.

When a connection is built between the problems of EVO-it and the exact method, the results of RitOpt can be compared to an optimal solution. With this comparison it is possible to tell how good RitOpt performs for these cases, which can be useful for making problem specific improvements.

6.3.2. Improving the current methods used by RitOpt

The main question of CQM is how to improve RitOpt. Even though this thesis research might not give a direct answer to this question, it can give a push in the right direction. During the literature research for this thesis, many articles have been read. Using this knowledge the following recommendations emerged.

The implementation is most important

There are many different type of metaheuristics one can choose from to solve different types of vehicle routing problems. The method in RitOpt uses is a form of simulated annealing and large neighbourhood search. There are many methods described in the literature, see section 2.4, that CQM could investigate for solving the vehicle routing problems of EVO-it, but this would mean that a complete new method needs to be build that is at least as flexible and as accurate as the method that is currently used by CQM.

It will take a lot of time and work to attain a careful implementation that can compete with the implementation made by CQM. This is something to consider when CQM wants to investigate different methods. Other options are to improve by using as discussed below.

Distinguish between different problem types

One of the suggestions of CQM to improve RitOpt was to categorize problem types and solve each category using a problem specific method. However, most real-life problems do not fit in one category, but have multiple attributes. This makes categorizing the different problem types of EVO-it very difficult. Furthermore, for each attribute of a problem one can find multiple heuristics to solve problems with this type of attribute, which makes it very difficult to choose a method. If a method seems to be the best for a specific attribute of a problem it might not be the best method for the other attributes that a problem has.

Still, one important distinction between the problems that CQM can consider is between problems with (tight) time windows and problems without (tight) time windows. This distinction has also been made in most articles and books about the vehicle routing problems in literature.

Other distinctions that are made quite often in literature are VRPs with and without pick up and delivery and VRPs with and without stochastic attributes, such as stochastic travel times. The first is already distinguished by the solving method of CQM and the latter case is not considered in the problems of EVO-it.

Another reason why it is difficult to say which distinctions CQM should make is that it is unclear for which problems RitOpt performs well and for which problems RitOpt performs poorly. If there is more clarity on which problems cause difficulties for RitOpt to solve accurate, CQM can act to improve this.

The method provided in this thesis can give more clarity on the performance of RitOpt.

Improve by using different local search neighbourhoods

One way to improve solving specific problem types is to create a completely different method from scratch. However, using specified neighbourhoods for specific problem types and using them in a smart way can already give a significant improvement. Therefore it is recommended to investigate the effect of different local search methods. The article of Funke et al. [29] might help in this research.

6.4. Summary of recommendations

Throughout this chapter multiple recommendations for future work have been given. The list below will enumerate the most important ones with respect to this research.

- Create a link between the exact method and the instances of EVO-it.
- Improve the subgradient optimization step in H^{1*} and H^{2*}.
- Include the other bounding procedures of Baldacci and Mingozzi [5].
- Improve the upper bound.
- Do additional experimenting on the parameters.

7

Conclusion

This final chapter summarizes the content of this thesis, and will give an answer to the research question posed in the introduction of this thesis, being:

Which real-life instances of vehicle routing problems at EVO-it can be solved using exact methods?

General conclusion

The exact method using formulation (SPP) has been able to solve an instance of 40 orders, 18 vehicles and 4 orders per vehicle. The smallest instances of EVO-it start with about 30 orders and the average number of used vehicles is approximately 14. With some improvements to upper and lower bounds used by formulation (SPP) this method will be able to solve small real-life instances of EVO-it, that contain capacitated vehicles, a variable number of vehicles, distance constraints, problems where not all orders can/need to be scheduled, fixed and variable costs and heterogeneous vehicles, to optimality.

This conclusion is drawn using the answers to the following sub questions:

1. *What methods are used to solve the VRP in literature?*
   The different type of methods that can be used to solve VRPs are exact methods, construction heuristics, local search or metaheuristics. The main conclusion learned from looking at the different methods has been that metaheuristics are generally the most flexible, which means that these methods can more easily be adopted to solve different type of vehicle routing problems and still give a quite accurate solution. This makes metaheuristics generally most applicable to solve real-life instances.

   Whereas exact methods are the most inflexible methods, usually defined for a specific type of VRP and adopting the method to solve other type of vehicle routing problems is often very difficult. From this it is concluded that to solve real-life instances from EVO-it with an exact method it is best to focus on problems with only a small set of different attributes.

2. *What type of VRPs appear at EVO-it?*
   To discover which types of VRPs appear at EVO-it, data has been gathered from 26 different instances. In Chapter 3 it can be seen that RitOpt has many different type of VRPs to deal with. One of the conclusions attained from the data was that the instances usually have a combination of multiple attributes, which makes it difficult to categorize the different problem types. An overview of the occurrence of each attribute is given in 3.5.

3. *What situations should the exact method be able to tackle?*
   Section 4.1 provides a selection of the attributes that are considered in this thesis. This selection consist of capacitated vehicles, a variable number of vehicles, distance constraints, problems where not all orders can/need to be scheduled, fixed and variable costs and heterogeneous vehicles.

4. *Which exact method is most appropriate to be able to include these situations?*
   In Chapter 4 three different type of formulations are discussed: the two-index flow formulation, the similar three-index flow formulation and a set partitioning formulation. The latter two are extended to

the three-index flow formulation with penalties (FP) and the set partitioning formulation with penalties (SPP), such that they are able to deal with all of the problem situations described above.

5. *How do the exact methods perform on instances that deal with these situations?*
   Both formulations (FP) and (SPP) can successfully solve instances with capacitated vehicles, a variable number of vehicles, distance constraints, problems where not all orders can/need to be scheduled, fixed and variable costs and heterogeneous vehicles. The exact method using formulation (SPP) performs very well on instances with many vehicles (20 vehicles have been tested). However, the computation time of the exact method using formulation (FP) increases significantly when the number of vehicles increases.

   When the capacity of the vehicles is relatively large compared to the size of the orders, the exact method using formulation (SPP) fails to solve the problem (within a limited amount of time), while the exact method (FP) is able to solve these type of problems, with the condition that the number of vehicles is not too large. This issue is related to the very large number of routes that is generated by the method. As discussed in Chapter 6 this can be improved by using better upper and lower bounds.

   The results in Chapter 5 show that formulation (FP) was able to solve problems with 19 customers and 2 vehicles, with a maximum number of 9 orders per vehicle route, to optimality, and that formulation (SPP) was able to solve problems with 40 customer and 4 vehicle types with a total of 18 vehicles, with a maximum number of 4 orders per vehicle route, to optimality.

A

Test instances

This appendix presents the instances created to test the three-index formulation with penalties and the set partitioning formulation with penalties. The instances are based on some of the instances from Uchoa et al. [78].

For simplicity the time matrices and distance matrices are the same for each vehicle k and the time matrix is a copy of the distance matrix, thus

$$t_{ij}^k = d_{ij}^k = d_{ij}, \quad \forall k \in M$$

For most instances coordinates are provided for each vertex $i \in V$ instead of a distance matrix. The distances are then calculated as Euclidean distances, i.e.,

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

The cost matrices c_{ij}^k are computed as in 4.16:

$$c_{ij}^k := c_t^k t_{ij}^k + c_d^k d_{ij}^k$$

Instance: HVRP-E-n13-k4-t1

Number of customers:	12
Number of vehicles:	4
Number of vehicle types:	1

Vertex	Demand	Distances d_{ij}											
i	q_i	d_{i0}	d_{i1}	$d_{i,2}$	d_{i3}	d_{i4}	d_{i5}	d_{i6}	d_{i7}	d_{i8}	d_{i9}	d_{i10}	d_{i11}	d_{i12}
0			9	14	21	23	22	25	32	36	38	42	50	52
1	1200	9		5	12	22	21	24	31	35	37	41	49	51
2	1700	14	5		7	17	16	23	26	30	36	36	44	46
3	1500	21	12	7		10	21	30	27	37	43	31	37	39
4	1400	23	22	17	10		19	28	25	35	41	29	31	29
5	1700	22	21	16	21	19		9	10	16	22	20	28	30
6	1400	25	24	23	30	28	9		7	11	13	17	25	27
7	1200	32	31	26	27	25	10	7		10	16	10	18	20
8	1900	36	35	30	37	35	16	11	10		6	6	14	16
9	1800	38	37	36	43	41	22	13	16	6		12	12	20
10	1600	42	41	36	31	29	20	17	10	6	12		8	10
11	1700	50	49	44	37	31	28	25	18	14	12	8		10
12	1100	52	51	46	39	29	30	27	20	16	20	10	10	

Table A.1: Properties of all locations $i \in V$ for instance HVRP-E-n13-k4-t1. The travel time t_{ij} between two locations i and j is set to $t_{ij} = d_{ij}$

Vehicle type	Q_k	L_k	F_k	c_t^k	c_d^k	U_k
k1	6000	300	0	0,5	0,5	4

Table A.2: Properties of each vehicle type $k \in M$ for instance HVRP-E-n13-k4-t1.

Instance: HVRP-E-n13-k6-t1

This instance is a copy of HVRP-E-n13-k4-t1 with 6 vehicles instead of 4.

Instance: HVRP-E-n13-k4-t1-LD

This instance is a copy of HVRP-E-n13-k4-t1, where the demand of vertex 12 is replaced with $q_{12} = 6001$. This results in a demand higher than the vehicle capacity, i.e., $q_i > Q_k$.

Instance: HVRP-E-n13-k4-t1-TL

This instance is a copy of HVRP-E-n13-k4-t1, where the maximum route length of the vehicles is set to a more tight value, $L_k = 110,\ \forall k \in M$.

Instance: HVRP-E-n13-k4-t1-LC

This instance is a copy of HVRP-E-n13-k4-t1, where the capacity of the vehicles is very large $Q_k = 15000,\ \forall k \in M$.

Instance: HVRP-E-n13-k4-t4

Number of customers:	12
Number of vehicles:	4
Number of vehicle types:	4

This instance is a copy of HVRP-E-n13-k4-t1, with four vehicle types. Table A.2 is replaced with:

Vehicle type	Q_k	L_k	F_k	c_t^k	c_d^k	U_k
k1	1700	50	13	0.5	0.5	1
k2	5200	100	0	0.5	0.5	1
k3	5800	100	10	0.5	0.5	1
k4	5600	120	6	0.5	0.5	1

Table A.3: Properties of each vehicle type $k \in M$ for instance HVRP-E-n13-k4-t4.

Instance: HVRP-E-n13-k4-t4-SC

This instance is a copy of HVRP-E-n13-k4-t1, where Table A.2 is replaced with:

Vehicle type	Q_k	L_k	F_k	c_t^k	c_d^k	U_k
k1	5200	200	0	0.5	0.5	1
k2	5000	200	0	0.5	0.5	1
k3	2300	200	0	0.5	0.5	1
k4	4000	200	0	0.5	0.5	1

Table A.4: Properties of each vehicle type $k \in M$ for instance HVRP-E-n13-k4-t4-SC.

In this instance the capacity is too small to fit all demand.

Instance: HVRP-E-n13-k20-t4

Number of customers: 12
Number of vehicles: 20
Number of vehicle types: 4

This instance is a copy of instance HVRP-E-n13-k4-t4, with 5 available vehicles for each vehicle type, i.e., $U_k = 5,\ \forall k \in M$

Instance: HVRP-E-n13-k20-t4-FC

This instances is a copy of HVRP-E-n13-k20-t4 where all vehicles have fixed costs:

Vehicle type	Q_k	L_k	F_k	c_t^k	c_d^k	U_k
k1	1700	50	13	0.5	0.5	5
k2	5200	100	20	0.5	0.5	5
k3	5800	100	10	0.5	0.5	5
k4	5600	120	6	0.5	0.5	5

Table A.5: Properties of each vehicle type $k \in M$ for instance HVRP-E-n13-k20-t4-FC.

Instance: HVRP-E-n13-k4-t2-SL

Number of customers: 12
Number of vehicles: 4
Number of vehicle types: 2

This instance is a copy of HVRP-E-n13-k4-t1, where Table A.2 is replaced with:

Vehicle type	Q_k	L_k	F_k	c_t^k	c_d^k	U_k
k1	6000	110	0	0.5	0.5	1
k2	6000	70	0	0.5	0.5	3

Table A.6: Properties of each vehicle type $k \in M$ for instance HVRP-E-n13-k4-t2-SL.

Instance: HVRP-E-n13-k2-t2-C

Number of customers: 12
Number of vehicles: 2
Number of vehicle types: 2

This instance is a copy of HVRP-E-n13-k4-t1, where Table A.2 is replaced with:

Vehicle type	Q_k	L_k	F_k	c_t^k	c_d^k	U_k
k1	6000	200	0	0.5	0.5	1
k2	6000	250	0	0.5	0.5	1

Table A.7: Properties of each vehicle type $k \in M$ for instance HVRP-E-n13-k2-t2-C.

Instance: HVRP-E-n13-k5-t2-FC

Number of customers: 12
Number of vehicles: 5
Number of vehicle types: 2

This instance is a copy of HVRP-E-n13-k4-t1, where Table A.2 is replaced with:

Vehicle type	Q_k	L_k	F_k	c_t^k	c_d^k	U_k
k1	6000	300	5	0.5	0.5	4
k2	6000	300	50	0.5	0.5	1

Table A.8: Properties of each vehicle type $k \in M$ for instance HVRP-E-n13-k5-t2-FC.

Instance: HVRP-P-n16-k8-t8

Number of Customers:	15
Number of vehicles:	8
Number of vehicle types:	8

Vertex	Demand	Coordinates	
i	q_i	x_i	y_i
0		30	40
1	19	37	52
2	30	49	49
3	16	52	64
4	23	31	62
5	11	52	33
6	31	42	41
7	15	52	41
8	28	57	57
9	8	62	42
10	8	42	57
11	7	27	68
12	14	43	67
13	6	58	48
14	19	58	27
15	11	37	69

Table A.9: Properties of all locations $i \in V$ for instance HVRP-P-n16-k8-t8.

Vehicle type	Q_k	L_k	F_k	c_t^k	c_d^k	U_k
k1	35	200	10	0.5	0.5	1
k2	35	100	13	0.5	0.5	1
k3	35	400	5	0.5	0.5	1
k4	35	400	20	0.5	0.5	1
k5	35	100	0	0.5	0.5	1
k6	35	400	0	0.5	0.5	1
k7	35	400	0	0.5	0.5	1
k8	35	400	0	0.5	0.5	1

Table A.10: Properties of each vehicle type $k \in M$ for instance HVRP-P-n16-k8-t8.

Instance: HVRP-P-n16-k8-t8-TF

This instance is a copy of instance HVRP-P-n16-k8, where the coordinates of vertex 15 is replaced with $(x_{15}, y_{15}) = (200, 200)$. This makes this vertex too far to visit for any of the vehicles while satisfying the maximum length constraints.

Instance: HVRP-P-n19-k2-t1

Number of customers:	18
Number of vehicles:	2
Number of vehicle types:	1

Vertex	Demand	Coordinates	
i	q_i	x_i	y_i
1		30	40
2	19	37	52
3	30	49	43
4	16	52	64
5	23	31	62
6	11	52	33
7	31	42	41
8	15	52	41
9	28	57	58
10	14	62	42
11	8	42	57
12	7	27	68
13	14	43	67
14	19	58	27
15	11	37	69
16	26	61	33
17	17	62	63
18	6	63	69
19	15	45	35

Table A.11: Properties of all locations $i \in V$ for instance HVRP-P-n19-k2-t1.

Vehicle type	Q_k	L_k	F_k	c_t^k	c_d^k	U_k
k1	160	200	0	0.5	0.5	2

Table A.12: Properties of each vehicle type $k \in M$ for instance HVRP-P-n19-k2-t1.

Instance: HVRP-P-n23-k18-t4

Number of customers: 22
Number of vehicle types: 18
Number of vehicle types: 4

Vertex i	Demand q_i	Coordinates x_i	y_i
1		30	40
2	7	37	52
3	30	49	49
4	16	52	64
5	23	31	62
6	11	52	33
7	19	42	41
8	15	52	41
9	28	57	58
10	8	62	42
11	8	42	57
12	7	27	68
13	14	43	67
14	6	58	48
15	19	58	27
16	11	37	69
17	12	38	46
18	26	61	33
19	17	62	63
20	6	63	69
21	15	45	35
22	5	32	39
23	10	56	37

Table A.13: Properties of all locations $i \in V$ for instance HVRP-P-n23-k18-t4.

Vehicle type	Q_k	L_k	F_k	c_t^k	c_d^k	U_k
k1	40	300	30	0.5	0.5	5
k2	30	200	0	1.5	0.5	4
k3	50	350	25	0.5	0.5	5
k4	40	200	0	0.5	0.5	4

Table A.14: Properties of each vehicle type $k \in M$ for instance HVRP-P-n23-k18-t4.

Instance: HVRP-E-n30-k8-t2

Number of customers: 29
Number of vehicle types: 8
Number of vehicle types: 2

Vehicle type	Q_k	L_k	F_k	c_t^k	c_d^k	U_k
k1	1500	800	100	1	0	4
k2	1000	500	300	1	0	4

Table A.15: Properties of each vehicle type $k \in M$ for instance HVRP-E-n30-k8-t2.

Vertex	Demand	Coordinates	
i	q_i	x_i	y_i
1		162	354
2	300	218	382
3	1000	218	358
4	425	201	370
5	300	214	371
6	200	224	370
7	650	210	382
8	850	104	354
9	450	126	338
10	300	119	340
11	500	129	349
12	950	126	347
13	425	125	346
14	250	116	355
15	350	126	335
16	550	125	355
17	750	119	357
18	500	115	341
19	350	153	351
20	400	175	363
21	300	180	360
22	500	159	331
23	100	188	357
24	300	152	349
25	500	215	389
26	800	212	394
27	300	188	393
28	400	207	406
29	450	184	410
30	730	207	392

Table A.16: Properties of all locations $i \in V$ for instance HVRP-E-n30-k8-t2.

Instance: HVRP-B-n31-k7-t2

Number of customers:	31
Number of vehicles:	7
Number of vehicle types:	2

Vehicle type	Q_k	L_k	F_k	c_t^k	c_d^k	U_k
k1	100	650	10	1	0	2
k2	70	400	0	0.5	0.5	5

Table A.17: Properties of each vehicle type $k \in M$ for instance HVRP-B-n31-k7-t2.

Vertex	Demand	Coordinates	
i	q_i	x_i	y_i
1		17	76
2	25	24	6
3	13	96	29
4	13	14	19
5	17	14	32
6	16	0	34
7	29	16	22
8	22	20	26
9	10	22	28
10	16	17	23
11	18	98	30
12	13	30	8
13	16	23	27
14	16	19	23
15	10	34	7
16	24	31	7
17	16	0	37
18	15	19	23
19	14	0	36
20	15	26	7
21	12	98	32
22	12	5	40
23	18	17	26
24	20	21	26
25	15	28	8
26	18	1	35
27	22	27	28
28	15	99	30
29	10	26	28
30	13	17	29
31	19	20	26

Table A.18: Properties of all locations $i \in V$ for instance HVRP-B-n31-k7-t2.

Instance: HVRP-E-n33-k10-t2

Number of customers: 33
Number of vehicles: 10
Number of vehicle types: 2

Vehicle type	Q_k	L_k	F_k	c_t^k	c_d^k	U_k
k1	4000	500	10	1	0	5
k2	2000	500	10	0.5	0.5	5

Table A.19: Properties of each vehicle type $k \in M$ for instance HVRP-E-n33-k10-t2.

Vertex i	Demand q_i	Coordinates x_i	y_i
1		292	495
2	700	298	427
3	400	309	445
4	400	307	464
5	1200	336	475
6	100	320	439
7	80	321	437
8	2000	322	437
9	900	323	433
10	600	324	433
11	750	323	429
12	1500	314	435
13	150	311	442
14	250	304	427
15	1600	293	421
16	450	296	418
17	700	261	384
18	550	297	410
19	650	315	407
20	200	314	406
21	400	321	391
22	300	321	398
23	1300	314	394
24	700	313	378
25	750	304	382
26	1400	295	402
27	4000	283	406
28	600	279	399
29	1000	271	401
30	500	264	414
31	2500	277	439
32	1700	290	434
33	1100	319	433

Table A.20: Properties of all locations $i \in V$ for instance HVRP-E-n33-k2.

Instance: HVRP-A-n33-k15-t3

Number of customers: 33
Number of vehicles: 15
Number of vehicle types: 3

Vehicle type	Q_k	L_k	F_k	c_t^k	c_d^k	U_k
k1	100	500	10	1	0	5
k2	50	300	10	0.5	0.5	5
k3	100	400	0	0.5	1.5	5

Table A.21: Properties of each vehicle type $k \in M$ for instance HVRP-A-n33-k15-t3.

Vertex	Demand	Coordinates	
i	q_i	x_i	y_i
1		34	31
2	26	45	55
3	17	70	80
4	56	81	70
5	15	85	61
6	7	59	55
7	15	45	60
8	15	50	64
9	16	80	64
10	17	75	90
11	21	25	40
12	21	9	66
13	66	1	44
14	25	50	54
15	16	35	45
16	11	71	84
17	37	1	9
18	17	25	54
19	17	45	59
20	22	45	71
21	10	66	84
22	25	11	35
23	16	81	46
24	27	85	10
25	21	75	20
26	11	15	21
27	21	90	45
28	11	15	0
29	21	31	26
30	22	10	95
31	25	6	6
32	12	51	5
33	22	26	36

Table A.22: Properties of all locations $i \in V$ for instance HVRP-A-n33-k15-t3.

Instance: HVRP-A-n38-k5-t2

Number of customers:	37
Number of vehicles:	5
Number of vehicle types:	2

Vehicle type	Q_k	L_k	F_k	c_t^k	c_d^k	U_k
k1	70	150	10	1	0	2
k2	100	300	5	2	2	3

Table A.23: Properties of each vehicle type $k \in M$ for instance HVRP-A-n38-k5-t2.

Vertex	Demand	Coordinates	
i	q_i	x_i	y_i
1		69	63
2	12	3	35
3	51	71	79
4	8	1	47
5	12	11	15
6	18	87	23
7	12	37	33
8	11	87	29
9	19	35	81
10	23	55	71
11	8	41	51
12	25	93	9
13	1	11	49
14	5	75	89
15	17	75	69
16	13	97	95
17	29	15	13
18	13	63	95
19	19	47	41
20	15	45	41
21	26	89	43
22	9	45	59
23	20	95	23
24	21	19	83
25	8	71	69
26	12	27	19
27	13	17	57
28	12	93	15
29	14	59	29
30	19	35	39
31	25	33	51
32	17	61	21
33	23	89	53
34	22	33	85
35	24	37	37
36	13	21	91
37	14	67	95
38	14	61	15

Table A.24: Properties of all locations $i \in V$ for instance HVRP-A-n38-k5-t2.

Instance: HVRP-P-n40-k18-t4

Number of customers: 39
Number of vehicles: 18
Number of vehicle types: 4

Vehicle type	Q_k	L_k	F_k	c_t^k	c_d^k	U_k
k1	150	160	80	1	1	4
k2	140	100	100	0.5	0.5	5
k3	80	100	50	1	0	5
k4	50	140	100	0	0	4

Table A.25: Properties of each vehicle type $k \in M$ for instance HVRP-P-n40-k18-t4.

Vertex	Demand	Coordinates	
i	q_i	x_i	y_i
1		30	40
2	47	37	52
3	30	49	49
4	16	52	64
5	59	20	26
6	21	40	30
7	15	21	47
8	19	17	63
9	23	31	62
10	31	52	33
11	65	51	21
12	49	42	41
13	29	31	32
14	23	5	25
15	21	12	42
16	70	36	16
17	45	52	41
18	35	27	23
19	41	17	33
20	90	13	13
21	28	57	58
22	85	62	42
23	58	42	57
24	46	16	57
25	70	8	52
26	38	7	38
27	75	27	68
28	50	30	48
29	140	43	67
30	60	58	48
31	90	58	27
32	29	37	69
33	120	38	46
34	30	46	10
35	62	61	33
36	57	62	63
37	60	63	69
38	90	32	22
39	58	45	35
40	46	59	15

Table A.26: Properties of all locations $i \in V$ for instance HVRP-P-n40-k18-t4.

Bibliography

[1] Yogesh Agarwal, Kamlesh Mathur, and Harvey M Salkin. A set-partitioning-based exact algorithm for the vehicle routing problem. *Networks*, 19(7):731–749, 1989.

[2] Claudia Archetti and M Grazia Speranza. A survey on matheuristics for routing problems. *EURO Journal on Computational Optimization*, 2(4):223, 2014.

[3] Claudia Archetti and Maria Grazia Speranza. Vehicle routing problems with split deliveries. *International transactions in operational research*, 19(1-2):3–22, 2012.

[4] Philippe Augerat. *Approche polyèdrale du problème de tournées de véhicules*. PhD thesis, Institut National Polytechnique de Grenoble-INPG, 1995.

[5] Roberto Baldacci and Aristide Mingozzi. A unified exact method for solving different classes of vehicle routing problems. *Mathematical Programming*, 120(2):347–380, 2008.

[6] Roberto Baldacci, Nicos Christofides, and Aristide Mingozzi. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming*, 115(2):351–385, 2008.

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

[8] Michel L Balinski and Richard E Quandt. On an integer program for a delivery problem. *Operations Research*, 12(2):300–304, 1964.

[9] EB Baum. Iterated descent: A better algorithm for local search in combinatorial optimization problems. *Manuscript*, 1986.

[10] John E Beasley. Route first—cluster second methods for vehicle routing. *Omega*, 11(4):403–408, 1983.

[11] Gerardo Berbeglia, Jean-François Cordeau, Irina Gribkovskaia, and Gilbert Laporte. Static pickup and delivery problems: a classification scheme and survey. *Top*, 15(1):1–31, 2007.

[12] Olli Bräysy and Michel Gendreau. Vehicle routing problem with time windows, part i: Route construction and local search algorithms. *Transportation science*, 39(1):104–118, 2005.

[13] Olli Bräysy and Michel Gendreau. Vehicle routing problem with time windows, part ii: Metaheuristics. *Transportation science*, 39(1):119–139, 2005.

[14] N Christofides. The vehicle routing problem. combinatorial optimization. christofides n., mingozzi a., toth p., sandi c.(eds) j, 1979.

[15] Nicos Christofides and Samuel Eilon. An algorithm for the vehicle-dispatching problem. *Journal of the Operational Research Society*, 20(3):309–318, 1969.

[16] Nicos Christofides, Aristide Mingozzi, and Paolo Toth. Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Mathematical programming*, 20(1):255–282, 1981.

[17] Geoff Clarke and John W Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581, 1964.

[18] Jean-Francois Cordeau and Gilbert Laporte. Tabu search heuristics for the vehicle routing problem. *Metaheuristic Optimization via Memory and Evolution*, pages 145–163, 2005.

[19] Jean-François Cordeau, Gilbert Laporte, and Anne Mercier. A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational research society*, 52(8):928–936, 2001.

[20] Jean-Francois Cordeau, Michel Gendreau, Gilbert Laporte, Jean-Yves Potvin, and Frédéric Semet. A guide to vehicle routing heuristics. *Journal of the Operational Research society*, pages 512–522, 2002.

[21] Jean-François Cordeau, Gilbert Laporte, Martin WP Savelsbergh, and Daniele Vigo. Vehicle routing. *Handbooks in operations research and management science*, 14:367–428, 2007.

[22] Rosario Cuda, Gianfranco Guastaroba, and Maria Grazia Speranza. A survey on two-echelon routing problems. *Computers & Operations Research*, 55:185–199, 2015.

[23] George B Dantzig and John H Ramser. The truck dispatching problem. *Management science*, 6(1):80–91, 1959.

[24] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. Ant colony optimization. *IEEE computational intelligence magazine*, 1(4):28–39, 2006.

[25] Thomas A Feo and Mauricio GC Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations research letters*, 8(2):67–71, 1989.

[26] Marshall L Fisher. Optimal solution of vehicle routing problems using minimum k-trees. *Operations research*, 42(4):626–642, 1994.

[27] Marshall L Fisher and Ramchandran Jaikumar. *A decomposition algorithm for large-scale vehicle routing*. Department of Decision Sciences, Wharton School, University of Pennsylvania, 1978.

[28] Marshall L Fisher and Ramchandran Jaikumar. A generalized assignment heuristic for vehicle routing. *Networks*, 11(2):109–124, 1981.

[29] Birger Funke, Tore Grünert, and Stefan Irnich. Local search for vehicle routing and scheduling problems: Review and conceptual integration. *Journal of heuristics*, 11(4):267–306, 2005.

[30] Michel Gendreau, Alain Hertz, and Gilbert Laporte. A tabu search heuristic for the vehicle routing problem. *Management science*, 40(10):1276–1290, 1994.

[31] Billy E Gillett and Leland R Miller. A heuristic algorithm for the vehicle-dispatch problem. *Operations research*, 22(2):340–349, 1974.

[32] F Glover, M Laguna, et al. Tabu search (vol. 22), 1997.

[33] Fred Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8(1): 156–166, 1977.

[34] Fred Glover. Tabu search—part i. *ORSA Journal on computing*, 1(3):190–206, 1989.

[35] Fred Glover. Tabu search—part ii. *ORSA Journal on computing*, 2(1):4–32, 1990.

[36] Fred Glover. Scatter search and path relinking. *New ideas in optimization*, 297316, 1999.

[37] Inc. Gurobi Optimization. Gurobi optimizer reference manual. 2016. URL `http://www.gurobi.com`.

[38] Florent Hernandez, Dominique Feillet, Rodolphe Giroudeau, and Olivier Naud. 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, 2016.

[39] Sin C Ho and Michel Gendreau. Path relinking for the vehicle routing problem. *Journal of Heuristics*, 12 (1):55–72, 2006.

[40] John H Holland. Adaptation in natural and artificial systems. an introductory analysis with application to biology, control, and artificial intelligence. *Ann Arbor, MI: University of Michigan Press*, 1975.

[41] Sašo Karakatič and Vili Podgorelec. A survey of genetic algorithms for solving multi depot vehicle routing problem. *Applied Soft Computing*, 27:519–532, 2015.

[42] Philip Kilby, Patrick Prosser, and Paul Shaw. Guided local search for the vehicle routing problem. In *Proceedings of the 2nd International Conference on Metaheuristics*, pages 21–24, 1997.

[43] Scott Kirkpatrick, C Daniel Gelatt, Mario P Vecchi, et al. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[44] Jari Kytöjoki, Teemu Nuortio, Olli Bräysy, and Michel Gendreau. An efficient variable neighborhood search heuristic for very large scale vehicle routing problems. *Computers & operations research*, 34(9): 2743–2757, 2007.

[45] Gilbert Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European journal of operational research*, 59(3):345–358, 1992.

[46] Gilbert Laporte. Fifty years of vehicle routing. *Transportation Science*, 43(4):408–416, 2009.

[47] Gilbert Laporte, Yves Nobert, and Martin Desrochers. Optimal routing under capacity and distance restrictions. *Operations research*, 33(5):1050–1073, 1985.

[48] Gilbert Laporte, Yves Nobert, and Martin Desrochers. Optimal routing under capacity and distance restrictions. *Operations research*, 33(5):1050–1073, 1985.

[49] Gilbert Laporte, Hélène Mercure, and Yves Nobert. An exact algorithm for the asymmetrical capacitated vehicle routing problem. *Networks*, 16(1):33–46, 1986.

[50] Feiyue Li, Bruce Golden, and Edward Wasil. Very large-scale vehicle routing: new test problems, algorithms, and results. *Computers & Operations Research*, 32(5):1165–1179, 2005.

[51] Shen Lin and Brian W Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.

[52] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. Iterated local search: Framework and applications. In *Handbook of metaheuristics*, pages 363–397. Springer, 2010.

[53] David Mester and Olli Bräysy. Active guided evolution strategies for large-scale vehicle routing problems with time windows. *Computers & Operations Research*, 32(6):1593–1614, 2005.

[54] Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Computers & operations research*, 24(11):1097–1100, 1997.

[55] Pablo Moscato et al. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech concurrent computation program, C3P Report*, 826:1989, 1989.

[56] Ilhan Or. *Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking*. PhD thesis, Northwestern University Evanston, IL, 1976.

[57] Ibrahim Hassan Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of operations research*, 41(4):421–451, 1993.

[58] Christos H Papadimitriou. The euclidean travelling salesman problem is np-complete. *Theoretical computer science*, 4(3):237–244, 1977.

[59] Diego Pecin, Artur Pessoa, Marcus Poggi, and Eduardo Uchoa. Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation*, 9(1):61–100, 2017.

[60] David Pisinger and Stefan Ropke. A general heuristic for vehicle routing problems. *Computers & operations research*, 34(8):2403–2435, 2007.

[61] Jean-Yves Potvin. State-of-the art review—evolutionary algorithms for vehicle routing. *INFORMS Journal on Computing*, 21(4):518–548, 2009.

[62] Jean-Yves Potvin and Jean-Marc Rousseau. An exchange heuristic for routeing problems with time windows. *Journal of the Operational Research Society*, 46(12):1433–1446, 1995.

[63] Christian Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, 31(12):1985–2002, 2004.

[64] Christian Prins. A grasp× evolutionary local search hybrid for the vehicle routing problem. *Bio-inspired algorithms for the vehicle routing problem*, pages 35–53, 2009.

[65] Caroline Prodhon and Christian Prins. Metaheuristics for vehicle routing problems. In *Metaheuristics*, pages 407–437. Springer, 2016.

[66] Marc Reimann, Karl Doerner, and Richard F Hartl. D-ants: Savings based ants divide and conquer the vehicle routing problem. *Computers & Operations Research*, 31(4):563–591, 2004.

[67] Yves Rochat and Éric D Taillard. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of heuristics*, 1(1):147–167, 1995.

[68] Robert A Russell and Wen-Chyuan Chiang. Scatter search for the vehicle routing problem with time windows. *European Journal of Operational Research*, 169(2):606–622, 2006.

[69] Martin WP Savelsbergh. Local search in routing problems with time windows. *Annals of Operations research*, 4(1):285–305, 1985.

[70] Martin WP Savelsbergh and Marc Sol. The general pickup and delivery problem. *Transportation science*, 29(1):17–29, 1995.

[71] Ahmet Şen and Kerem Bülbül. A survey on multi trip vehicle routing problem. 2008.

[72] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *International Conference on Principles and Practice of Constraint Programming*, pages 417–431. Springer, 1998.

[73] Kenneth Sörensen and Marc Sevaux. Ma| pm: memetic algorithms with population management. *Computers & Operations Research*, 33(5):1214–1225, 2006.

[74] Anand Subramanian, Eduardo Uchoa, and Luiz Satoru Ochi. A hybrid algorithm for a class of vehicle routing problems. *Computers & Operations Research*, 40(10):2519–2531, 2013.

[75] Éric Taillard. Parallel iterative search methods for vehicle routing problems. *Networks*, 23(8):661–673, 1993.

[76] Paolo Toth and Daniele Vigo. The granular tabu search and its application to the vehicle-routing problem. *Informs Journal on computing*, 15(4):333–346, 2003.

[77] Paolo Toth and Daniele Vigo. *Vehicle routing: problems, methods, and applications*. SIAM, 2014.

[78] Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, Anand Subramanian, and Ivan Xavier. Cvrplib, capacitated vehicle routing problem library, 2014. URL `http://vrp.atd-lab.inf.puc-rio.br/index.php/en/`.

[79] Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, and Anand Subramanian. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3):845–858, 2017.

[80] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, and Christian Prins. A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers & operations research*, 40(1):475–489, 2013.

[81] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, and Christian Prins. A unified solution framework for multi-attribute vehicle routing problems. *European Journal of Operational Research*, 234(3): 658–673, 2014.

[82] Christos Voudouris and Edward Tsang. Guided local search and its application to the traveling salesman problem. *European journal of operational research*, 113(2):469–499, 1999.

[83] Steffen Wolf and Peter Merz. Evolutionary local search for the super-peer selection problem and the p-hub median problem. *Hybrid Metaheuristics*, pages 1–15, 2007.