A practical framework for the electric vehicle routing problem

Johan Hellmark

LUND UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6155
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Tryckeriet i E-huset
Lund 2022

Abstract

The routing of a delivery fleet is a classical optimization problem, known as the vehicle routing problem (VRP), which can heavily impact the quality of a logistic distribution process. Historically, the VRP formulation has mainly included internal combustion engine vehicles (ICEVs). However, due to their reduced environmental impact, the inclusion of electric vehicles (EVs) has become more popular. The inclusion requires accounting for a shorter driving range and limited infrastructure support. This thesis presents a framework for solving several practical extensions of the electric vehicle routing problem (E-VRP). Previously presented solvers usually target specific problem variants, optimize based on predetermined objectives, and display a general lack of discussion on their practical applicability. To counteract these shortcomings, the implemented framework allows for customizable objective functions, is capable of solving a wide range of practically relevant extensions, and provides an interface for investigating the properties of the proposed solutions. Examples of subjects treated are partial recharging, time-dependent variables, and dynamic settings. Solutions to real-world settings, modeled using the API of Iternio Planning AB, are demonstrated and the implemented solver shows promising results on a wide range of tested problem instances.

Acknowledgements

First and foremost, I would like to express my appreciation to Iternio Planning AB for allowing me to perform my thesis in collaboration with them. I am particularly grateful for the engagement shown by Bo Lincoln who has guided me through the process as my supervisor at the company. From the Department of Automatic Control, I would like to acknowledge the help provided by my supervisor Giacomo Como and his valuable feedback.

Lastly, I would like to extend a general thanks to everyone how has shown interest and participated in discussions regarding my thesis.

Contents

1. Introduction **9**
- 1.1 Background . . . 9
- 1.2 Outlined report structure . . . 10
- 1.3 Purpose & research questions . . . 11
- 1.4 Why is the problem hard? . . . 11
- 1.5 Work processes . . . 12

2. Theory **14**
- 2.1 Vehicle routing problems (VRP) . . . 14
- 2.2 Electric vehicle routing problems (E-VRP) . . . 15
- 2.3 VRP extensions . . . 17
- 2.4 Fixed size homogeneous capacitated E-VRP with time windows . 20
- 2.5 NP-hardness . . . 23
- 2.6 General-purpose solution approaches . . . 23
- 2.7 Penalty functions . . . 24
- 2.8 Introduction to algorithms . . . 24
- 2.9 Algorithms in depth . . . 26

3. Outlined approach **31**
- 3.1 The objective function, constraints and costs . . . 31
- 3.2 Dealing with uncertainties . . . 32
- 3.3 Charging stations and partial recharging . . . 34
- 3.4 Dealing with time-dependence . . . 35
- 3.5 Problem instances . . . 35
- 3.6 Dealing with large datasets . . . 36
- 3.7 Constructing an initial solution for the E-VRP . . . 37

4. The framework **38**
- 4.1 Overview . . . 38
- 4.2 Implemented algorithm . . . 40
- 4.3 Motivation for the implemented algorithm . . . 43
- 4.4 Improving the performance . . . 44
- 4.5 Examples applications . . . 46

4.6 Evaluating the quality of a solution 47
5. Computational results **49**
5.1 Parameter tuning 49
5.2 Benchmarks 52
5.3 Generated datasets 54
5.4 Real-world instances 61
6. Conclusion **66**
A. Operators **68**
A.1 Intra route operators 68
A.2 Inter route operators 70
B. Inputs and parameters **71**
B.1 Cost functions and penalty functions 71
B.2 Parameters settings 73
Bibliography **75**

1

Introduction

Imagine yourself being the manager of a small courier service. To contribute to a more sustainable society, all vehicles included in your fleet are electrical. The business has been going exceptionally well, and the size of your fleet was recently enlarged, from two to ten. With the increase in capacity, the firm is now ready to serve more customers and today's orders require visiting a total of 100 customers. It is your responsibility to decide on the sequence of customer visits and the routes to be taken. You have set aside a quarter of an hour and sit down with a printed map, a pen, and paper. However, you quickly realize that the problem is more complex than you thought at first. To produce a good solution, you must approximate the fuel consumption and account for overtime costs, load capacity, time windows, and unforeseeable traffic congestions.

Indeed, the problem you are facing is a well-known combinatorial optimization problem, known as the vehicle routing problem (VRP). The VRP deals with the question "What is the optimal set of routes for a fleet of vehicles to traverse in order to deliver to a given set of customers?". One extension to the VRP is the inclusion of electric vehicles (EVs), for which shorter driving ranges and more restricted infrastructure support must be considered. This thesis, performed in collaboration with Iternio Planning AB, presents a framework for generating and analyzing solutions to multiple extensions of the electric vehicle route problem (E-VRP).

1.1 Background

The logistic sector in Sweden was responsible for 16.63 million ton carbon dioxide equivalents during 2019, which was one-third of the country's total greenhouse gas emission (Environmental Protection Agency, Sweden, 2020). The Swedish government is attempting to reduce these emissions whereby one of the Swedish national climate goals is to reduce the emission from domestic transports by 70% from 2010 to 2030 (Ministry of the Environment, Sweden, 2021). In the statistics presented by the Swedish Environmental Protection Agency (2020), there is a decreasing trend

in emissions from domestic transports, in large due to alternative fuels and more efficient vehicles.

However, to accomplish the goal, the decrease must be intensified. The agency argues that more transports should be performed by the marine or railway. When this is impractical, increasing the usage of alternative fuels can contribute to a reduction of emissions. Currently, light trucks are in almost 90% of the cases fueled with diesel and are thus an area where alternative fuel can have a big impact (Environmental Protection Agency, Sweden, 2020). To enable the progress towards the usage of renewable fuels, the agency underlines the importance of effective vehicles and general improvements through different transport solutions.

Electricity is an example of an alternative fuel. Major logistic firms such as FedEx are transitioning to deliveries using EVs. FedEx has set the explicit goal that, by the year 2025, 50% of the pickup delivery vehicle purchased shall be electric, rising to 100% by 2030 (Fedex Corporation, 2021). While EVs can be advantageous to include in a fleet, they are in some aspects more restricted than the standard internal combustion engine vehicles (ICEVs). These restrictions must be overcome to enable actors within the logistic sector to include EVs within their fleets.

1.2 Outlined report structure

The second chapter of the report presents the *theory* and previous literature related to this thesis. It gives an introduction to the VRP, E-VRP, and other extensions. Further, the VRP is proven to be NP-hard, literature on general purpose-solver is reviewed, and penalty functions are introduced. Lastly, an introduction to the most commonly applied algorithms is provided.

The third chapter presents the *outlined approach*. It begins with a discussion on how the objective function affects the performance of a solver. It continues discussing the inherent uncertainty present in the problem, followed by a section on the handling of charging stations and charging function. Implications of allowing for time-dependent variables and how these can be handled are then covered. The following section deals with that for some problem extensions investigated in this thesis, no publicly available datasets were available, and therefore custom datasets needed to be generated. Lastly, compromises performed to handle large datasets and aspects to consider when constructing an initial solution to the E-VRP are discussed.

The fourth chapter presents and motivates decisions for the implemented *framework*. Further, it discusses how to improve the performance of the algorithm and how it can be applied to a few selected extensions. Lastly, how the framework can be used to assess the quality of a solution is covered.

The fifth chapter presents the *computational results*. It evaluates the performance of the implemented algorithm on existing benchmarks. Results on custom-generated datasets are presented. Finally, the solver is applied to real-world settings using the API of Iternio Planning AB.

The final chapter of the thesis serves as a *conclusion* which summarizes the work and provides guidelines for feature research.

1.3 Purpose & research questions

The purpose of this master's thesis is to develop a framework for the E-VRP which can be used in practice by Iternio Planning AB. This thesis will therefore answer the following research questions:

RQ 1: What aspects are to be considered and which components should be included in a practically useful framework for solving the E-VRP?

RQ 2: How can an algorithm for solving a wide variety of E-VRP extensions be constructed?

RQ 3: What determines the quality of a solution to the E-VRP?

1.4 Why is the problem hard?

There are multiple challenges in solving and studying the E-VRP. The origin of the difficulty is that the problem is NP-hard, meaning that no known method of solving it in polynomial time exists. The simpler version of the problem, *the traveling salesman problem* (TSP), has $(n-1)!$ number of possible solutions where n is the number of customers to visit. This number increases for the E-VRP, whereby brute-force approaches are unlikely to be successful. Since the E-VRP is a combinatorial optimization and integer programming problem, the search space will be discrete. Further, for some instances, large parts of the search space will be unfeasible due to constraints. These properties must be taken into account when formulating the optimization method. Exploiting the structure of the problem is not trivial, especially not when extending the problem with several attributes. Further, since the number of available states in a partial solution grows upon adding more attributes, it becomes difficult to evaluate the quality of a certain action and compare partial solutions.

The algorithm must be constructed to allow for the support of multiple problem extensions. Firstly, the algorithm should allow for including and excluding attributes dynamically. Secondly, the algorithm is not allowed to use extension-specific knowledge for improving the performance, unless it is adequate for the problem formulation. This drastically decreases the speed at which one can evaluate a solution. As the most effective methods for solving the E-VRP use guided searches to explore the search space, the increased evaluation time hinders their performance.

The quality of a solution depends on how the objective function is defined, and the objective function may vary depending on the problem formulation. As a result,

the performance of an algorithm depends on the problem formulation and should be evaluated on each variation of the problem separately. The performance of an algorithm also depends on how it is measured, for example, speed of convergence, the best solution obtained, or the most diverse set of solutions. In summary, there is no straightforward method for comparing two algorithms whereby it is difficult to conclude that an algorithm outperforms another.

1.5 Work processes

The project was split into four main phases: *planning phase, seed product development, feature implementation,* and *ending phase*. The *planning* and *ending* phases deal with practicalities related to the thesis while the main parts of the development were performed during the *seed product development* and *feature implementation* phases. The methodology used is feature-driven and inspired by the agile development methodologies.

The planning phase

The purpose of the *planning phase* was to specify the subject and scope of the thesis. Further, a target document and a project plan were developed.

The seed product development

The purpose of the *seed product development* was to develop a product that could be extended throughout the feature implementation phase. The developed solver was limited in functionality but could find solutions to the standard VRP.

The feature implementation

During the *feature implementation phase*, features were added to the seed product. The methodology used was inspired by agile practices and contained short sprints. Each sprint consisted of iterations following the plan-do-study-act cycle (PDSA-cycle).

During the *feature implementation phase* the seed product was extended through the implementation of additional features. This phase was the most important and constituted the largest part of the project. The methodology used was inspired by the agile methodologies and short sprints were used. Each sprint consisted of one or many iterations following the plan-do-study-act cycle (PDSA-cycle). The advantage of this approach is that it is flexible and includes the stakeholders in the development process. The cycles started with the *planning phase* in which possible features to implement were proposed. In consensus with the supervisors, the highest prioritized feature was selected for further investigation. The *do phase* started with a literature review. A wide search was conducted to identify multiple sources and topics of relevance. The most relevant references were identified and studied in-depth. After the literature review, the problem model and implementation were updated to

include the new feature. In the *study phase*, the implementation of the feature was evaluated. The stakeholders were provided with information about the outcome of the iteration. Lastly, a short evaluation was written on the most important lessons learned during the iteration and if any major adjustments should be made in the work process. The stakeholders received this in the form of the weekly update.

The ending phase

In the ending phase, a popular scientific report was written, and a project presentation and opposition were performed.

2

Theory

The purpose of this chapter is to introduce subjects relevant to the thesis and review previous literature. The chapter starts with a more formal introduction of the VRP, E-VRP, and its extensions through the framework of graph theory. Then, proof of NP-hardness is presented. Finally, previous general-purpose solvers are discussed, penalty functions are described, and commonly applied algorithms are introduced.

2.1 Vehicle routing problems (VRP)

The problem of how to optimally route a set of vehicles in order to visit a given set of customers is known as the VRP and was introduced in 1959 in a paper focusing on gasoline delivery trucks (Dantzig and Ramser, 1959). In the basic version of the VRP, a set of K identical vehicles are located at a central depot. The objective is to visit a set of customers while minimizing the total cost. The problem can be modeled using a graph $G = (V,A)$, assumed to be strongly connected, where $V = 0,...,n$ is the vertex set and A is the arc set. V_0 is defined to correspond to the depot, while $V_{1,...,n}$ corresponds to the customers. Let D denote the set of depots and I the set of customers. In the current settings $I = V \setminus \{0\}$. A non-negative cost c_{ij} is associated to each arc $(i, j) \in A, i \neq j$. Without loss of generality, G can be made complete by connecting all unconnected nodes using arcs with a cost of $c = +\infty$. Additionally, to avoid self-loops $c_{ii} = +\infty$ except for the depot where the cost of a self-loop is zero, $c_{00} = 0$.

If G is directed the cost matrix can be asymmetric, whereas if it is undirected the cost matrix is always symmetric, since $c_{ij} = c_{ji}$. The cost matrix is always expected to satisfy the triangle inequality $c_{ij} + c_{jk} \geq c_{ik}$ which implies that it is never beneficial to deviate from a direct link between two vertices.

The VRP consist of finding a set of K or less simple circuits which minimizes the cost function, defined as the sum of the cost of arcs traversed, such that

- each circuit visits the depot vertex,
- each customer vertex is visited by exactly one circuit.

A solution to the VRP will be denoted with S, where every S consists of a set of routes, $S = \{R_i\}$. A route R_i is defined by a vehicle and an ordered set of vertices$\{V_{0...0}\} \subseteq V$, which starts and ends in the depot.

Problem formulation

The VRP can be modeled using a mixed-integer linear programming (MILP) formulation. The formulation presented in this and the following section is of low practical usefulness and should be viewed as an introduction to how MILP is used for the VRP. The formulation for the basic version of the problem is presented below. In the equations c_{ij} denotes the cost of traveling from i to j. x_{ij} is a binary variable that has value 1 if the arc from i to j is a part of the solution and otherwise 0, K is the number of available vehicles. Further, recall that the set V contains all vertices, the set D contains all depots, and the set I contains all customers.

$$\text{(VRP)} \quad \min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \tag{2.1}$$

subject to

$$\sum_{j \in V, i \neq j} x_{ij} = 1, \qquad \forall i \in I \tag{2.2}$$

$$\sum_{j \in V} x_{0j} \leq K \tag{2.3}$$

$$\sum_{j \in V, i \neq j} x_{ji} - \sum_{j \in V, i \neq j} x_{ij} = 0, \qquad \forall i \in V \tag{2.4}$$

$$x_{ij} \in \{0,1\}, \qquad \forall i, j \in V \tag{2.5}$$

The objective function 2.1 is to minimize the total cost of traversed arcs. Constraint 2.2 imposes that a customer is visited exactly once. Constraint 2.3 imposes that the number of vehicles leaving the depot is less or equal to the number of vehicles available. Constraint 2.4 imposes flow conservation.

2.2 Electric vehicle routing problems (E-VRP)

The electric vehicle problem is an extension of the VRP where a limited battery capacity and charging stations are taken into account. The problem is modeled as a graph $G = (V, A)$ with $n+m$ number of vertices. V_0 corresponds to the depot, $V_{1,\ldots,n}$ correspond to the customers and $V_{n+1,\ldots,n+m}$ correspond to the chargers. Let the set of charging stations (CSs) be denoted as F. Since a CS is allowed to be visited more than once, a dummy set, F', is created where an arbitrary (possibly infinite) number of copies for each CS is included. Hence, when looking at the E-VRP we have $V = D \cup I \cup F'$. All vehicles are assigned a battery capacity, Q, and each arc is

assigned a fuel cost, e_{ij}. The current battery level will be referred to as the *state of charge* (SoC) which is defined as $\frac{\text{Current battery level}}{\text{Maximum battery level}}$. Further, partial recharging will be allowed and all chargers are assigned a charging function which represents the time it takes for a vehicle to recharge. The objective is the same as in the VRP but with the additional requirement of

- the SoC shall never be 0 unless the vehicle is at a charger or a depot.

The inclusion of EVs in the VRP is a relatively new topic and one of the earliest publications dates back to 2011 (Gonçalves et al., 2011). Fuel consumption is a central aspect of the E-VRP and is often assumed to be relative to the distance traveled. There are however examples of when the fuel consumption is dependent on more parameters, such as the vehicle load (Goeke and Schneider, 2015), (Lin et al., 2016). The E-VRP can be viewed as an extension that allows intermediate refueling stops. Therefore assumptions must be made about the charging policy and the charging function approximation. The former defines how much a battery should be charged and can be classified into full or partial charging policies. The latter decide how long it takes to recharge a battery to a certain level. The charging function can be either linear or non-linear. For the non-linear case, a piece-wise linear approximation can be used to obtain the approximated function. For a more in-depth description on the topic of charging functions, the reader is referred to Montoya et al. (2017).

As EVs are more heavily affected by the environment and have a shorter driving range they are susceptible to uncertainties in the environment. Uncertainty can be derived from limited availability and waiting times at chargers, difficulties in predicting fuel consumption, and other external factors such as congestion and traffic accidents. Further, some aspects of the E-VRP are time-dependent. Both waiting times at chargers (Keskin et al., 2019) and aspects of traveling between two locations can be expected to be time-dependent. The charging speed can decrease when multiple vehicles are using the same CS simultaneously. Therefore, the charging times can be expected to be longer during peak hours.

Problem formulation

The E-VRP will add additional constraints to the MILP formulation of the basic VRP. To formulate the problem, a few notations will be introduced. e_{ij} represents the energy consumption of traversing the arc between i and j, E_{min} the minimum SoC, and E_{max} the maximum SoC. c_j denotes the cost of visiting node i. The variables ε_i tracks the SoC upon arrival and $\hat{\varepsilon}_i$ the SoC upon departure from the vertex $j \in V$. σ_i is the recharged quantity, where $\sigma_i = 0$, $i \notin F'$. The E-VRP can now be formulated as

$$\text{(E-VRP)} \quad \min \sum_{i \in V} \sum_{j \in V} (c_j + c_{ij}) x_{ij} \tag{2.6}$$

subject to

$$\sum_{j \in V, i \neq j} x_{ij} = 1, \qquad \forall i \in I \tag{2.7}$$

$$\sum_{j \in V, i \neq j} x_{ij} \leq 1, \qquad \forall i \in F' \tag{2.8}$$

$$\sum_{j \in V} x_{0j} \leq K \tag{2.9}$$

$$\sum_{j \in V, i \neq j} x_{ji} - \sum_{j \in V, i \neq j} x_{ij} = 0, \qquad \forall i \in V \tag{2.10}$$

$$E_{min} \leq \varepsilon_i \leq \hat{\varepsilon}_i \leq E_{max} \qquad \forall i \in V \tag{2.11}$$

$$\hat{\varepsilon}_j \leq (\hat{\varepsilon}_i + \sigma_j - e_{ij}) x_{ij} + (1 - x_{ij}) E_{max} \qquad \forall i, j \in V,\ i \neq j \tag{2.12}$$

$$\hat{\varepsilon}_i = \varepsilon_i + \sigma_i \qquad \forall i \in V \tag{2.13}$$

$$\sigma_i = 0 \qquad \forall i \notin F' \tag{2.14}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall i, j \in V \tag{2.15}$$

The objective function 2.6 remains unchanged from the VRP formulation. The constraint 2.7 - 2.10 remains unchanged except for the addition of 2.8. Constraints 2.11-2.14 track the SoC, ensure that it never is negative or less than the previous SoC, and limit the charging.

2.3 VRP extensions

The purpose of this section is to introduce extensions to the E-VRP. As it is intended to work as an overview, there will not be any MILP formulation or more technical details presented in this section. In practical use cases, several VRP attributes can be considered jointly, which yields the *multi-attribute vehicle routing problems* (MAVRP). In a survey by Vidal et al. (2013) on MAVRP, three main classes of attributes were presented, namely: *Assignment of customers and routes to resources* (ASSIGN), *sequence choices* (SEQ) and *Evaluation of fixed sequences* (EVAL). The list of extensions presented below is not exhaustive as only the most important extensions for this thesis have been selected.

ASSIGN-extensions

The first category, ASSIGN, contains attributes that impact the assignment of limited resources, e.g. vehicles, vehicle types, and depots.

Multi-depot vehicle routing problems (MD-VRP) deal with the cases where multiple depots are available. Contrary to the single depot case, vehicles are allowed to start and end in multiple depots. It is also possible to add more restrictions, such as allowing only certain combinations of start and end nodes.

Heterogeneous fleet vehicle routing problems (HF-VRP) deal with a set of vehicles with different properties. The HF-VRP can either be formulated with a fixed fleet size or with an unknown fleet size.

SEQ-extensions

The second category, SEQ, influences the structure of the routes and in which order the customers should be traversed.

The vehicle routing problems with backhauls (VRP-B) divide the customers into two groups: delivery customers (linehaul customers) and pick-up customers (backhaul customers) where all delivery customers must be served before the first pick-up customer.

The pickup and delivery vehicle routing problems (PD-VRP) divide the customers into pickup and delivery pairs. Both customers within a pair must be served by the same vehicle and the customer assigned pickup must be visited first.

EVAL-extensions

The last category, EVAL, contains attributes that must be checked once the routes are constructed.

Capacitated vehicle routing problems (C-VRP) assign each vertex with a demand, d_i, where $d_i = 0$, $i \notin I$. We define the total demand of a set, V', as $d(V') = \sum_{i \in V'} d_i$, $V' \subseteq V$. Given a homogeneous fleet, all vehicle are assigned the capacity C, which is assumed to satisfy $d_i \leq C$ for all $i = 1, ..., n$. Let $r(V')$ denote the minimum number of vehicles needed to construct a feasible solution. The minimum number of vehicles needed to serve all customers is defined as K_{min} and can be expressed as $r(V) = K_{min}$. Deciding the value of $r(V)$ is known as *the bin-packing problem* which is proven to be NP-hard and therefore it can be advantageous to replace it by a lower bound $K_{min} = d(S)/C$ which is easily computed. Hence, the C-VRP extends the VRP by requiring all circuits to satisfy that

- the sum of the demands of the vertices visited by a circuit does not exceed the vehicle capacity, C.

Vehicle routing problems with time windows (VRP-TW) is an extension to the VRP in which each customer is assigned a time window, $[a_i, b_i]$ for $i \in I$. Additionally, each customer is assigned a service time s_i. The service of a customer should start within the given time window. In the case of early arrival, the vehicle must wait until the start of the time window before beginning the service. Hence all circuits are required to satisfy that

- for each customer i, the service starts within the time window, $[a_i, b_i]$, and the vehicle stops for s_i time instants.

Miscellaneous-extensions

Certain extensions might not fit in the previously presented categories and are therefore presented separately.

Time-dependent vehicle routing problems (TD-VRP) allow parameters to be time-dependent. Time-dependent travel times were first discussed by Cooke and Halsey (1966) and the TD-VRP was first presented almost 30 years ago by Malandraki and Daskin (1992). Allowing parameters to be time-dependent is a more realistic model as factors such as fuel consumption and travel time are dependent on weather- and traffic conditions which in turn are time-dependent. Further, as previously discussed, for the E-VRP waiting times and other properties of the CSs are affected by the time at which one arrives.

Stochastic vehicle routing problems (S-VRP) & Dynamic vehicle routing problems (D-VRP). The S-VRP deals with uncertainties through the usage of stochastic parameters. It can thus be categorized as a probabilistic combinatorial optimization problem. In the D-VRP, problem data changes during the execution of a plan. In practice, this can either be due to unexpected events, e.g. a traffic jam or a new customer request, or faulty assumptions, e.g. how much fuel it takes to travel between two destinations. The adjustments to the planned solution can be either event-based or time-driven (Ritzinger et al., 2016).

Pillac et al. (2013) performed a survey on the dynamic and stochastic VRP in which the following categorization was presented:

- *Deterministic and static* - all input are known exactly beforehand and does not change during the execution.
- *Deterministic and stochastic* - some of the input is only partially known as random variables but they do not change during the execution. Common stochastic input variables are customer demand and travel time.
- *Dynamic and static* - parts or all of the input is unknown beforehand and revealed during the execution.
- *Dynamic and stochastic* - similar to the dynamic and static case except that it is possible to exploit some stochastic knowledge on the dynamically revealed information.

Vehicle routing problems with uncertainty can be viewed as a superclass to the S-VRP. For the S-VRP random variables are assumed to be drawn from a specified probability distribution. In practical applications, the probability distribution is often unknown whereby other approaches must be used. An alternative problem formulation is the Robust VRP which takes the approach used in robust optimization and uses an uncertainty set (Bertsimas et al., 2011).

2.4 Fixed size homogeneous capacitated E-VRP with time windows

In this section, an incomplete MILP formulation to a multi attribute extension of the E-VRP is presented. The incompleteness is due to simplifications done to improve the readability. This trade-off is motivated as the formulation will not be used by the solver and the purpose of the section is to give the reader a deeper understanding of how different constraints and costs are handled. A list of notations is presented, followed by an explanation of how some of the variables are defined.

V set of all nodes

D set of depots

I set of customers

F' set of chargers

M set of vehicles

c_i cost incurred on an vertex from the visit to node i

c_{ij} cost incurred on arc from node i to j

P_k start cost of vehicle k

t_i arrival time at node i

t_{ij} travel time between node i and j

w_i wait time at node i

s_i service time at visit to node i

a_i earliest arrival time at node i

b_i latest arrival time at node i

m_k total number of vehicle of the same type as k

r_k maximum route time allowed for vehicle k

Q_k load capacity of vehicle k

q_{ij} quantity carried by a vehicle at departure from node i to j

d_i demand of customer i

E_{min_k} min SoC of vehicle k

E_{max_k} maximum SoC of vehicle k

e_{ij} SoC incurred on arc from node i to j

ε_i SoC upon arrival to node i

$\hat{\varepsilon}_i$ SoC upon departure from node i

σ_i amount charged at node i

P_k start cost for using vehicle k

The service time s_i can be viewed as either time serving a customer if $i \in I$ or as time charging if $i \in F'$. In the latter case, a function $g : t_i, \varepsilon_i, \Delta, k \to t$ which maps the arrival time, the arrival SoC, the amount needed to charge, and vehicle type to a time must be defined. A detailed presentation of how the charging function is constructed lies outside the scope of the thesis and will therefore be assumed to be provided in the dataset.

To allow for soft constraints the set $\mathscr{Z}_n = \{\{z_i, \mathfrak{C}_i\} | i \in n\}$ will be used to denote all soft constraints. $z_i \in \mathbb{R}^+$ represents the degree of violation and $\mathfrak{C}_i$ is positive non-decreasing function for which $\mathfrak{C}_i : z_i \to c$, where $c \in \mathbb{R}^+$ denotes the cost of the

violation. This notation will be used to add penalties to violations of soft constraints. The notation c_{ij} is used to represent the cost of traversing an arc and c_i to represent the cost of visiting a node. These constants can be viewed as the result of a function that takes an arbitrary number of route properties and maps them to a common cost value.

In the following equations the notation k will be used to denote a vehicle type. To clarify, the only time $x_{ij}^k = 1$ is when the a vehicle of type k travels between node i and j. The notation e_{ij}^k is used when the value depends on the vehicle type.

$$\textit{Minimize} \sum_{k\in M} P_k \sum_{j\in V} x_{0j}^k + \sum_{k\in M}\sum_{i\in V}\sum_{j\in V}(c_{ij}^k + c_j^k)x_{ij}^k + \sum_{\mathfrak{C}_i, z_i \in \mathscr{Z}_n} \mathfrak{C}_i(z_i) \tag{2.16}$$

subject to

$$\sum_{k\in M}\sum_{j\in V, i\neq j} x_{ij}^k = 1, \qquad \forall i \in I \tag{2.17}$$

$$\sum_{k\in M}\sum_{i\in V, i\neq j} x_{ij}^k \leq 1, \qquad \forall j \in F' \tag{2.18}$$

$$\sum_{j\in V} x_{0j}^k \leq m_k \qquad \forall k \in M \tag{2.19}$$

$$\sum_{j\in V, i\neq j} x_{ji} - \sum_{j\in V, i\neq j} x_{ij} = 0, \qquad \forall i \in V \tag{2.20}$$

$$\sum_{i\in V} q_{ji} - \sum_{i\in V} q_{ij} = d_j \qquad \forall j \in V \tag{2.21}$$

$$d_i x_{ij} \leq q_{ij} \leq (Q_k - d_j)x_{ij} \qquad \forall i,j \in V, i \neq j, \forall k \in M \tag{2.22}$$

$$0 \leq q_{ij} \leq Q_k \qquad \forall i,j \in V, k \in M \tag{2.23}$$

$$\sum_{i\in V}\sum_{j\in V, i\neq j} x_{ij}(t_i j + s_i + w_i) - z_r \leq r_k \qquad \forall k \in M \tag{2.24}$$

$$x_{ij}(t_i + t_{ij} + s_i + w_i) \leq t_j \qquad \forall i,j \in V, i \neq j \tag{2.25}$$

$$a_i \leq (t_i + w_i) - z_{tw} \leq b_i \qquad \forall i \in I \tag{2.26}$$

$$E_{min\_k} \leq \varepsilon_i \leq \hat{\varepsilon}_i \leq E_{max\_k} \qquad \forall i \in V,\ k \in M \tag{2.27}$$

$$\hat{\varepsilon}_j \leq (\hat{\varepsilon}_i + \sigma_j - e_{ij}^k)x_{ij} + (1 - x_{ij})E_{max\_k} \qquad \forall i,j \in V,\ i \neq j,\ k \in M \tag{2.28}$$

$$\hat{\varepsilon}_i = \varepsilon_i + \sigma_i \qquad \forall i \in V \tag{2.29}$$

$$x_{ij} \in \{0,1\}, \qquad \forall i,j \in V \tag{2.30}$$

$$\sigma_i \geq 0 \qquad \forall i \in F' \tag{2.31}$$

$$\sigma_i = 0 \qquad \forall i \in V \setminus F' \tag{2.32}$$

$$z_n \geq 0 \qquad \forall n \tag{2.33}$$

The objective function is to minimize the total cost which includes a sum describing the initial cost for each vehicle, a sum for the cost of traversing the vertices and arcs, and a sum asserting a penalty for the exceedance of a soft constraint. Constraint 2.17 imposes that a customer is visited exactly once. Constraint 2.18 imposes that a charger is visited at most once. Constraint 2.19 imposes that the number of vehicles leaving the depot is less or equal to the number of vehicles available. Constraint 2.20 imposes flow conservation. Constraints 2.21 to 2.23 ensure the correct capacity change, ensure that the route is always feasible, and restrict the starting capacity to be within allowed limits. Constraint 2.24 ensures that an exceedance of the max time is penalized. Constraint 2.25 imposes limitations on the arrival times

and 2.26 ensures that a time window is either met or penalized. Constraints 2.27 to 2.29 ensure fuel feasibility and tracks the SoC changes.

2.5 NP-hardness

The TSP can be shown to be an NP-hard problem by reducing it to a well-known NP-complete problem, namely the Hamiltonian cycle problem. A Hamiltonian cycle is a path in a graph that visits all vertices exactly once. The Hamilton cycle problem is the problem of determining if such a path exists or not. To prove that the TSP can be reduced to the Hamiltonian cycle problem use a graph $G = (V, E)$ where $|V| = n$ and $c(e) = 1$ for all $e \in E$. Then add E' to G in order to make the graph complete and let $c(e) = 2$ for $e \in E'$. These actions can be performed in polynomial time. Determining if there exists a solution that has the cost n is the TSP decision problem, and since it is shown to be reducible to the Hamilton cycle problem it is NP-complete. Therefore, the problem of minimizing the cost of the TSP is NP-hard as there is no way in polynomial time to verify that the obtained solution is optimal. Further, since the VRP is a generalization of the TSP, the VRP is also NP-hard.

2.6 General-purpose solution approaches

Multiple general-purpose solvers have been presented in the literature. Vidal et al. (2014) presents three main approaches for achieving generality: *rich solvers*, *modelling and solution frameworks*, and *component-based frameworks*. *Rich solvers* are solvers that are designed to address a MAVRP formulation through generalizing several variants associated with subsets of attributes. Well received example of rich solvers are a solver using tabu search presented by Cordeau et al. (1997), a solver using adaptive large neighborhood search presented by Ropke and Pisinger (2006), and a solver using iterative local search presented by Hashimoto et al. (2008). *Modelling and solution frameworks* transform the general properties of the attributes and transform them into machine-readable components. For example, Desaulniers et al. (1998) presented a solver in which attributes are formulated as resources that are extended to successive customer visits through resource extension functions. *Component-based frameworks* can be used to increase the flexibility of the solver. Components can incorporate specific problem attributes and modify the applied algorithm based on the problem formulation. For more information and examples on *component based framework* the reader is referred to Vidal et al. (2014).

2.7 Penalty functions

Since some versions of the VRP are highly constrained it can be beneficial to visit unfeasible regions as this can facilitate the exploration. This can be allowed through modifying the original cost function $f(x)$ according to $f_p(x) = f(x) + p(d(x,F))$ where $d(x,F)$ is a distance metric from the infeasible point, x, to the feasible region F. The penalty function p is a monotonically increasing function that is zero for any value within the feasible region. This allows for constraints to be removed and instead accounted for in the cost function.

Smith et al. (1997) classifies penalty functions as either *constant, static, dynamic* or *adaptive*. In the *constant* case a constant penalty is added upon exceeding a constraint. In the *static* case the cost function can be expressed as $f_p(x) = f(x) + \sum_{i=1}^{m} C_i d_i^{\kappa}$ where κ is a constant to be chosen, commonly either 1 or 2. The distance metric used in this report will be on the form $d_i = \delta_i \alpha_i g_i(x)$ where δ_i is 1 if the constraint is exceeded, otherwise 0, α_i is a scaling constant and $g_i(x)$ is the exceedance. In the *dynamic* case the cost function can be expressed as $f_p(x,t) = f(x) + \sum_{i=1}^{m} s_i(t) d_i^{\kappa}$ where $s_i(t)$ commonly is a monotonically increasing function. As $s(t)$ increases in value infeasible solutions are more heavily penalized, whereby the solutions will gradually move towards the feasible region. Lastly, *adaptive* penalty functions uses information from previous search results to adapt the penalty function.

2.8 Introduction to algorithms

The VRP has received much attention during the last 60 years and multiple different approaches to solving the problem have been presented. A classification of available algorithms for solving the VRP is summarized in Figure 2.1. Naturally, this classification can be performed in multiple ways, but the one presented is based on a survey by Lin et al. (2014).

Exact algorithms can be divided into three broad categories: *direct tree search methods, dynamic programming*, or *integer linear programming*. While guaranteeing to find the best possible solution these methods are often infeasible due to computing time on larger instances. They will not be discussed further as they often are inapplicable to the problem presented.

Approximate algorithms are most commonly used in practice and try to find a near-optimal solution within an acceptable computation time. The remaining parts of this section discuss the approximate algorithms in more detail.

Classical heuristics

According to Toth and Vigo 2002, classical VRP heuristics can be divided into three categories.

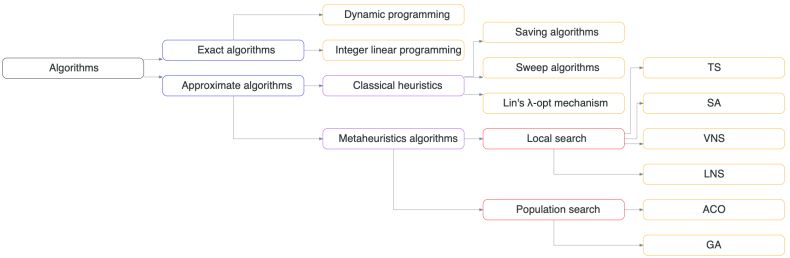


Figure 2.1: A classification of available algorithms for solving the VRP.

The first category is called *constructive methods* and it aims to construct initial routes. This can be done either through merging existing routes or by gradually assigning customers to paths through insertion. *Clark and Wright saving algorithm* is an example of an algorithm that aims to merge routes based on the notion of savings.

The second category is *two-phase methods*, which can be further categorized into *cluster-first, route-second* methods and *route-first, cluster-second methods*. In *cluster-first, route-second* vertices are organized into feasible clusters from which routes then are created. Within this category algorithms such as the *sweep algorithm* and the *truncated branch-and-bound algorithm* can be found. In *route-first, cluster-second methods* a tour including all vertices are first constructed and later organized into feasible clusters.

The last category of classical heuristics is *improvement methods* in which a constructed solution is improved. An improvement method can either treat each route separately, *single-route improvements*, or several routes simultaneously, *multi-route improvements*. Most of the *single-route improvements* methods can be described by *Lin's λ-opt mechanism* (Lin and Kernighan, 1973). Within the *multi-route improvements* vertices or arcs are exchanged between routes. In the implemented algorithm *improvement methods* are important and a description of the implemented methods can be found in Appendix A.

Metaheuristics

Metaheuristics are general algorithmic frameworks, designed to solve complex optimization problems. They offer general procedures that explore the solution spaces in a structured manner. Metaheuristics are non-deterministic and commonly use search experience to guide the search. It is in practice common to combine the metaheuristic algorithm with a problem-specific improvement heuristic to enhance the performance. A short overview of metaheuristic algorithms, used in combinatorial optimization, will be presented based on an article by Blum and Roli (2003).

Metaheuristics can be categorized as

- nature-inspired vs. non-nature inspired,
- population-based vs. single point search,
- one vs. various neighborhood structures,
- memory usage vs. memory-less methods,
- multi-start vs perturbation,
- constructive vs. improving methods.

Nature-inspired vs. non-nature inspired classifies the algorithm based on the origin. *Population-based vs. single point search* classifies the algorithm based on if the algorithm works on a population of solutions or a single solution. The single point search can also be referred to as *trajectory methods*. The only algorithm using *various neighborhood structures* presented will be Variable Neighborhood Search (VNS). *Memory-less methods* perform a Markov process where the next action is solely based on the previous state. There are different ways of making use of memory, for example through tracking recently visited solutions or accumulating the previous results into parameters affecting the search. A further distinction can be made between *multi-start and perturbation algorithms*. They have the same purpose; to increase the search space through operators that do not easily allow the algorithm to fall back into the same local minimum. *Multi-start* is characterized by that at the start of each iteration a new initial solution is created from scratch. Algorithms using *perturbations* will instead modify an existing solution and use the modified solution as a starting point. Both procedures must result in a new solution that is not too similar to the previous solutions, as this limits the search space. Still, the algorithm should use information gained from previous iterations as it otherwise would be a random restart. Lastly, metaheuristics can, similarly to the classical heuristics, be divided into *constructive and improving methods*. The former can generate a new solution while the latter must be supplied with an initial solution.

2.9 Algorithms in depth

In this section, a few algorithms commonly applied to the VRP are presented. The algorithms are divided into either *constructive algorithms* or *improvement algorithms* depending on if they can be used to construct an initial solution or not.

Constructive algorithms

There are multiple algorithms that can be used to construct an initial solution. A selected sample is presented below.

Nearest neighbor algorithm The nearest neighbor algorithm is a straightforward algorithm in which the next traversed arc is the one that has the lowest cost. While simple to implement and quick in execution, the routes produced can be of relatively low quality, especially those created in the later stages. As the solutions can be of low quality, the solution generated can be infeasible even if feasible solutions exist. Further, to generate diverse solutions one would also need to introduce some randomness.

Clarke and Wright savings algorithm The Clarke and Wright savings algorithm is conceptually easy to grasp and extensively used (Clarke and Wright, 1964). The algorithm is described in the single depot setting where V_0 is the depot and n is the number of customers:

1. Create n routes as $V_0 \rightarrow V_i \rightarrow V_0$ where $i \in I$.
2. Calculate the savings for merging delivery i and j according to $s_{ij} = c_{0i} + c_{j0} - c_{ij}$ for all $i, j \in I$.
3. Sort the savings in descending order.
4. Starting from the top, merge two routes with the largest remaining savings provided that:
   - The customers are not already in the same route.
   - Both customers are directly connected to a depot.
   - Combining the routes results in a feasible route.
5. Repeat step 3 until no additional savings can be achieved.

Sweep algorithms Sweep algorithms create a sorted list according to the angle between the customer location and the depot. Routes are then constructed using the sorted list. The sweep algorithm can be applied in multiple ways. A simple application to the C-VRP is to cluster nodes using the sweep algorithm (Suthikarnnarunai, 2008). A new cluster is initialized when the addition of the next node would violate the capacity constraint. It can however be difficult to generalize this procedure to a wider range of problem extensions and the effectiveness can be questioned. Still, it is possible to extend the sweep algorithm and combine it with other heuristics to improve its effectiveness (Na et al., 2011).

Ant colony optimization Ant colony optimization (ACO) is a population-based metaheuristic that can be applied to combinatorial optimization problems (Dorigo et al., 2006). ACO uses swarm intelligence (SI) which relies on the collective behavior of a decentralized system, consisting of simple agents. SI algorithms are often inspired by nature, and ACO is inspired by the foraging behavior of an ant colony. An ant who has found a source of food communicates the location of the source using stigmergy, more precisely through laying down pheromones on the way back

to the nest. The pheromone stimulates the other ants, and it is more likely for an ant to follow a trail with a high degree of pheromones. Hence, with the help of a chemical substance, the ants have created a decentralized system for coordinating their foraging. There are multiple versions presented in literature but in this report $MAX-MIN$ Ant System (MMAS) will be used (Stützle and Hoos, 2000). The general algorithm for ACO is presented in Algorithm 1.

Algorithm 1 ACO

```
1: Set parameters, initialize pheromone trails
2: while termination conditions not met do
3:     ConstructSolution
4:     ApplyLocalSearch   % optional
5:     UpdateTrails
6: end while
```

ACO uses either consecutive or simultaneous construction of routes to construct a solution. The next vertex to be visited is a probabilistic decision, in which an unvisited customer or charger is chosen. The probabilistic choice is biased by the pheromone trail τ_{ij} and by a locally available heuristic information η_{ij}. The probability of choosing an already visited customer is zero and the probability of the next vertex being j, given the current vertex i for the ant k is

$$p_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in N_i^k}[\tau_{il}]^{\alpha}[\eta_{il}]^{\beta}} \quad \text{if} j \in N_i^k$$

where α and β are two parameters that determine the relative importance of the pheromone trail and the local heuristic information. N_i^k is the feasible neighborhood of ant k, i.e. un-visited customers and nodes which does not exceed the stop condition. The construction can be performed either one route at a time or n routes simultaneously. In the simultaneous case, which route to be updated is chosen in either a sequential or a heuristic manner.

The *pheromone trail updating* on MMAS is done according to

$$\tau_{ij}(t+1) = \rho\,\tau_{ij}(t) + \Delta\tau_{ij}^{best}$$

where ρ ($0 < \rho \leq 1$) is the speed of evaporation, $\Delta\tau_{ij}^{best} = C/c(S^{best})$, C is a constant and $c(S^{best})$ is the solution cost of either the iteration-best, S^{ib}, or the global-best, S^{gb}. Through being able to vary between the usage of iteration-best and global-best one can control how the history of the search is exploited. Using the global-best pheromone in the update will increase the speed of convergence but reduce the exploration. A natural way of handling the balance between iteration-best and global-best is to begin with mainly applying the S^{ib} and then dynamically increase the frequency of using S^{gb} during the search. In MMAS a pheromone limit is set

to ensure that search stagnation will not occur. Stagnation can happen if one choice trail is significantly higher for one choice compared to the other. Therefore, $\tau_{min} \leq \tau_{ij} \leq \tau_{max}$ is defined and if any of the limits are exceeded that trail is set to the value of the limit.

Improvement algorithms

In contrast to the constructive algorithms, the improvement algorithms must be supplied with an initial solution, which is to be improved. As a result, constructive and improvement algorithms can be combined.

Iterated local search Iterative local search contains four components: *initial solution, perturbation, local search*, and *acceptance criterion* (Lourenço et al., 2003) . The *initial solution* is constructed using a constructive algorithm which has already been extensively covered in a previous section. The *acceptance criterion* decides the requirements for accepting a new solution. The *local search* component will apply operators on an existing solution to improve it. The *perturbation* is applied to escape local minima and should generate a new solution from which it is difficult for the local search to reach the previous minimum.

Variable neighborhood search Variable neighborhood search (VNS) is a metaheuristic method for solving combinatorial optimization problems (Mladenović and Hansen, 1997). VNS systematically changes the neighborhood of a solution combined with a local search. It relies upon the following observations:

1. A local minimum with respect to one neighborhood structure is not necessarily a local minimum for another neighborhood structure.
2. A global minimum is a local minimum with respect to all possible neighborhood structures.
3. For many problems local minima with respect to one or several neighborhoods are relatively close to each other.

A neighborhood $N(x)$, is defined in relation to the initial solution x. Thereafter, a descent direction within the neighborhood $N(x)$ is chosen and used in the next iteration. In this report, a first improvement heuristic will be practiced, whereby a move is made as soon as a direction of descent is found. To handle large instances, as the evaluation of a solution is costly and hence also a local search operation a *reduced variable neighborhood search* (RVNS) will be used. This extension generates a point x' at random where $x' \in N$. It then evaluates if the generated point is an improvement. If so is the case, the generated point is used for the next iteration. If no improvement was found, it either repeats the procedure until a stop condition is met or changes the neighborhood. The two most important steps in the VNS can be summarized as follows:

- *Shaking* - Generates a new solution from a given neighbourhood.

- *Local Search* - Improves a supplied solution.

3

Outlined approach

The purpose of this chapter is to give an intuition of the main ideas behind the implemented solution. The first section describes the objective function. The following sections describe how uncertainty, charging stations, and time dependence are handled. Thereafter, the next three sections describe how problem instances are handled, how large datasets can be managed, and considerations regarding the constructions of an initial solution.

3.1 The objective function, constraints and costs

The objective function decides the quality of a solution and will thus impact the properties of the generated solutions. In previous MILP formulations, the objective function was formulated as the sum of the cost of all traversed vertices and arcs, and the cost of constraints exceedances. In this section, we will extend this definition, make the formulation more detailed, and discuss how the construction of the objective function can facilitate finding a high-quality solution.

The objective function can be optimized on one parameter or multiple parameters. Examples of parameters to optimize are total distance, total travel time, or fuel consumption. The optimization using multiple parameters can either be performed by combining parameters to a common cost unit or through using multi-objective optimization. In some cases, attributes can intuitively be converted to the same cost unit. For example, fuel consumption, total travel time, and overtime can be converted using fuel cost, hourly wages, and overtime wages. For other instances, the conversion might be hard to perform or for other reasons unwanted, whereby multi-objective optimization can be used. Examples of aspects difficult to convert are customer satisfaction, the risk of running out of fuel, the total number of vehicles used, emissions, and route balance. An example of a multi-objective solution is presented by Baños et al. (2013) in which the VRP was solved using three objective functions that minimized: the total cost, the carbon-dioxide emission, and the emission of air pollutants. Another example is Miranda et al. (2018) who presents a solver for the multi-objective VRP with time windows, stochastic travel times and

stochastic service times. In most cases, an optimal solution that dominates all other solutions (is better for all objective functions) does not exist. A solution that is not dominated by another solution is called Pareto optimal. For more information on the multi-objective VRP, the reader is referred to Jozefowiez et al. (2008).

Penalty functions are a method for including the exceeding of a constraint as a penalty in the form of an added cost in the objective function, described more in-depth in section 2.7. Penalty functions are motivated by the fact that the search space for the E-VRP, and its extensions, is heavily constrained. Therefore, feasible solutions constitute only a small fragment of all possible solutions and the constrained objective function is non-convex. Another property of penalty functions is that they provide a natural framework for relaxing the constraints, which is beneficial for two reasons. Firstly, it can be challenging to create an initial feasible solution without the ability to relax the problem. Secondly, through relaxation permutations can be performed to escape local minimums.

When formulating an optimization problem constraints can be either soft or hard. Soft constraints are allowed to be exceeded at a penalty cost while the exceedance of hard constraints is forbidden. Typical examples of hard constraints for the E-VRP is vehicle load capacity and fuel limitations. Hard constraints should heavily penalize, at least when not relaxed, even a small exceedance. If the exceedance of a constraint is forbidden, a jump discontinuity can be used. The difficulty of approximating certain properties, such as fuel consumption, makes it natural to model them using penalty functions and safety margins instead of strict limits. Some constraints have a practical interpretation, such as maximum allowed travel time after which overtime wage is paid. The magnitude of these costs can be supplied by the decision-maker. Penalty functions and the relaxation schema will have a large impact on the performance of the algorithm and the properties of the solutions generated. However, as these are problem-specific and it is difficult to draw any general conclusions. More detailed information about how these concepts are implemented can be found in conjunction with the presentation of the proposed algorithm in chapter 4.

In the implemented framework it is possible to use separate costs during the algorithm execution and evaluation of final solutions. This decoupling can counteract unwanted behavior and improve the performance of the algorithm. One example is the overtime wage for exceeding the maximum travel time limit. This attribute is natural to declare as a linear cost proportional to the overtime. However, in the algorithm an exponential penalty should be used, to prevent assigning all overtime to the same vehicle.

3.2 Dealing with uncertainties

Uncertainties are unavoidable when planning a set of routes for EVs. Uncertainties can be derived from factors such as traffic conditions, waiting times, changes in

customer demands, and weather conditions. One of the parameters that have an inherent uncertainty is the fuel consumption of an EV. According to Li et al. (2016) the fuel consumption is influenced by multiple factors which can be divided into six categories:

- technology and vehicle factors,
- artificial environment factors,
- natural environment factors,
- driver factors,
- travel type factors,
- measurement factors.

Each of these categories contains multiple aspects which impact the driving range of an EV. While it is possible to create a reasonable model of fuel consumption there will always be some uncertainty in the model, both due to the number of influential factors and the fact that some factors must be predicted.

In practice, there are two methods for handling these uncertainties. The first method will be referred to as *a priori optimization* in which the routes are not changed or updated after the algorithm execution. The randomness is instead accounted for when planning the routes prior to the plan realization. One common approach within the category is stochastic programming. The second option, referred to as a *re-optimization strategy*, allows the plans to be updated during the execution. The update can either be done based on pre-calculation for different outcomes or by optimizing based on the new state at run time.

In most use cases, the distribution of the stochastic parameters is difficult to estimate. Hence, we should assume that our knowledge about the underlying probability distribution is sparse or inexact. As a result, the framework of stochastic programming will be difficult to apply since it requires known probability density functions. Thus, if we are to perform *a priori optimization* we must find a method that does not depend on the exact distribution of the probability density function. The most straightforward approach would be to use a safety margin when constructing the routes, e.g. the SoC is not allowed to go below 10%. The advantages of this method are that it is easy to both understand and implement. The drawback is that the cost of solutions obtained might be increased, in some instances drastically. A more nuanced method would be to apply a cost reduction for unused resources. The marginal gain in safety margin is more important close to exceeding a constraint. As a result, the cost reduction should be a concave non-decreasing function. The last proposed approach would be to use tools from the field of robust optimization. In robust optimization no distributional assumptions are made, instead, parameters are assumed to belong to a deterministic uncertainty set.

All of the mentioned approaches could be combined with online optimization, which re-optimizes the routes during execution. In practice, this would mean taking a snap-shot of a certain point in time and optimizing based on the new problem state. This would result in a multi-depot E-VRP with a fixed-size heterogeneous fleet.

3.3 Charging stations and partial recharging

Each CS is associated with a charging function which maps an arrival time, a start SoC, an end SoC, and a vehicle, to a departure time. The marginal charge time with respect to the SoC is monotonically increasing whereby the charging function will be concave. In practice, this implies that the battery charges faster with a low SoC. In addition to the influence of SoC, different vehicles can have different charging functions. To decrease the computational efforts all charging functions will be discretized. Creating realistic charging functions for different types of vehicles and chargers is deemed to lie outside the scope of this thesis. Therefore, when the charging functions are not provided in the dataset, the same charging functions will be used for all types of vehicles. The charging functions are obtained from Montoya et al. (2017) and includes three types of chargers. The used values are presented in Table 3.1, where it is shown how long time it would take to charge to a specific SoC from 0%. The charging function is assumed to be linear between the displayed values and the time is presented in hours. Lastly, it should be stated that partial recharging will be allowed as this increases the quality of solutions.

Table 3.1: The charging time (hours) used in this report based on the charger type and SoC.

	85%	95%	100%
Fast	0.31	0.39	0.71
Normal	0.62	0.77	1.01
Slow	1.26	1.54	2.04

3.4 Dealing with time-dependence

Parameters will be allowed to be time-dependent. In this work, we will limit the time dependence to attributes related to congestion, namely; fuel consumption and travel time. The choice of these parameters is due to that they have an obvious time dependence in reality. While more aspects, such as waiting time at charging stations and charging speed are time-dependent, they are not implemented. The reason for not implementing them is that the congestion already shows that the solver can handle time-dependent attributes. Once the solver can handle time dependence, implementing additional time-dependencies within the attributes should be simple.

When allowing for partial recharging and time dependence intricate relationship between the optimal amount to charge, fuel consumption, and travel time arises. Without time dependence, it was possible to optimize the charging at all CSs given that one knows the whole route. This could be done by first charging the required amount to reach the next CS, then continuing charging if, and only if, the current cost of charging is less than the cost of charging at the next CS. This decision will depend on the type of charger and the current SoC. Hence, the cost of a route, with respect to the amount charged at a CS, will be convex. Introducing time-dependence makes the problem of deciding how much to charge at a CS more difficult. The cost of a route as a function of the amount charged at a CS is no longer convex. As a result, a local minimum point can not be assumed to be the global minimum, and finding the optimal amount to charge is difficult. The reason is that the fuel needed depends on at what time a vehicle traverses the coming arcs, and the time at which a vehicle traverses the coming arcs depends on how much it is charged.

Computational complexity

Some constraints can be checked in $O(1)$ time, for example, load capacity. Further, battery penalties and time windows without time-dependence can be computed in $O(1)$ if only one node is inserted between two partial routes (Schiffer and Walther, 2018). While efficient ways of evaluating a change exist, most of these are not applicable due to the allowance of time dependence. Hence, to evaluate the cost of a modification, the affected routes are reconstructed from the insertion point. The cost is then evaluated by traversing all nodes within the routes and summing their costs.

3.5 Problem instances

All information needed to define a problem instance is defined as a dataset. A dataset contains information about the depots, customers, chargers, and vehicles to be used. In this thesis, multiple sources are used to find the datasets. Some of the used instances are publicly available and related to specific articles. In other cases, the dataset is generated for this thesis or created from a real-world problem setting. For

all real-world instances, the API of Iternio Planning AB will be used to estimate the fuel consumption and travel times. In all other cases, these parameters will be assumed to be provided in the dataset. More information about a dataset will be presented in conjunction with the results obtained through solving the dataset, see chapter 5.

Time-dependent parameters will be modeled using naive models since developing realistic models falls outside the scope of this work. The exact distribution of these variables will be described in detail when a specific dataset is presented. In general, they will be constructed through the discretization of a well-known probability distribution. The properties of the arc will solely be based on the time of the departure. This simplification is performed to reduce the computational complexity and ease of implementation.

3.6 Dealing with large datasets

In this section, we will use the number of vertices present in the dataset to define its size. Naturally, this is not the only possible measurement of the size of a dataset but it will serve its purpose in the following discussion. When the size of a dataset increases the two main difficulties arising are:

1. The number of arcs increases and thus also the number of attributes that must be approximated.
2. The execution time of the algorithm will increase as the number of possible routes and the number of routes within a solution increases.

The first difficulty originates from that the number of arcs in a complete, directed graph, with n number of vertices, is calculated by the formula $n^2 - n$. Therefore, the growth in the number of arcs is quadratic in respect to the growth in the number of vertices. For a dataset of 200 vertices, the number of arcs would thus be 39800. The number of vertices is problematic partly due to the number of API calls needed and partly due to the computational complexity in modeling randomness and time dependence for all arcs. There are two possible solutions to this problem. The first would be to reduce the number of calculations performed and not calculate all parameters related to each arc individually. This could be accomplished using an algorithm that estimates the parameters of an arc intelligently and therefore limits the number of lookups required. How to implement a reliable algorithm for this purpose would be a challenging task and is deemed to lie outside the scope of this work. The second solution would be to not include all arcs in the graph and set the cost of traversing the excluded arcs to infinity. This would require a method for deciding which arcs to include. A naive approach would be to only include the arcs which are connected to the closest vertices. As an example, allowing only visits to the 20 closest vertexes would decrease the number of arc look-ups from 39800 to 4000 or, in a more general notation, from $n^2 - n$ to $\min[n-1, 20] \times n$.

In regards to the second difficulty, the solution of limiting the number of look-ups makes the problem easier to solve as there are not as many available routes. Of course, this comes at the cost of possibly missing optimal routes that include traveling between two vertexes located far from each other. A more detailed explanation of possible methods for limiting the number of look-ups is presented in chapter 4.4.

3.7 Constructing an initial solution for the E-VRP

The initial solution acts as a starting point for the improvement algorithms. Therefore, it must produce feasible and diverse solutions rather than solutions with high quality, as this can be improved later. A common method for constructing a feasible solution is to use a constructive heuristic method, such as the Clark and Wright algorithm, or metaheuristics, for example, ACO. Another method is to construct an initial TSP solution and then use a splitting procedure to generate VRP solutions. For the E-VRP, charging stations must be included when constructing an initial solution.

We make the delimitation that a vehicle is assigned a single, specific start depot, while multiple end depots are allowed. Further, we assume that all fleets are of fixed size. This can be done without loss of generality as it is possible to assign any arbitrary number of vehicles to a depot, thus a fleet without fixed size can be replicated. The routes of the initial solution can be constructed successively (one at a time) or simultaneously (all at once). If the routes are created successively, a stop condition must be declared to ensure that the first vehicle does not visit all customers. If the number of vehicles to include in the solution is known, simultaneous construction can be advantageous. The reasoning is that it is difficult to control the number of vehicles included in a successively created solution.

The inclusion of chargers complicates the construction of an initial solution and can either be performed simultaneously as the insertion of customers or afterward in a fixed route of customers. In the former case, it is difficult to approximate the amount to charge and which charger to be visited as both these decisions are highly dependent on the subsequent sequence of nodes in the route, which at the time is unknown. In the latter case, it is difficult to construct a good sequence of customers without knowledge about the chargers as these will influence the quality of the chosen sequence. The problem of selecting which CS to insert and how much to charge at each CS in a fixed route is called the fixed-route vehicle charging problem (FRVCP). The objective of the FRVCP is to minimize the cost and ensure the feasibility of a fixed route of customers through the insertion of CSs. Hence, this operation is not allowed to alter the order of the customers. The FRVCP is a generalization of the fixed-route vehicle refueling problem (FRVRP). Since the FRVRP is proven to be NP-hard, the FRVCP is also NP-hard (Montoya et al., 2017). Another difficulty with the inclusion of chargers is that it might affect stop conditions such as maximum allowed travel time.

4

The framework

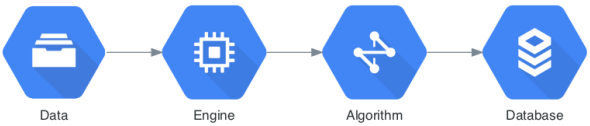
The purpose of the chapter is to describe the developed framework. The chapter starts with an overview of the implemented framework, followed by discussions on the implemented algorithm. Thereafter, how to improve the performance of the algorithm and application examples are presented. Lastly, a discussion on evaluating the quality of a solution is conducted.

4.1 Overview

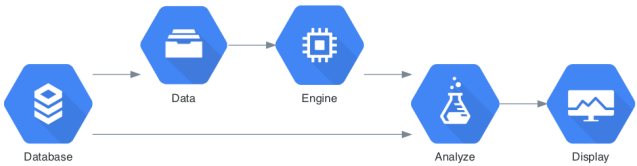
The framework implemented to solve the E-VRP consists of a set of modular components, written in Python. The structure of the framework can be viewed in Figure 4.1. The main advantage of the structure is that the result obtained can be analyzed post algorithm execution which makes the framework flexible.

The *data component* contains the datasets with information about the vertices, arcs, vehicles, constraints, and costs. The *engine* reads data from the data component and parses it into Python data structures. Its main purpose is to take a sequence of vertices in combination with a vehicle and return information about the corresponding route. The *algorithm component* creates solutions to a specified dataset. It uses the engine to evaluate the quality of the routes. Upon completing the execution of an algorithm the results are saved into a *database*. The *analyze component* fetches data from the database and creates an engine with the data used in the algorithm. It then analyzes the solutions created to obtain additional information. The *display component* displays information about the solutions obtained from an algorithm.

The algorithm, the parameters, the cost function, and the penalty functions are all read into the framework as configuration files. This is helpful as users will have different use cases and costs. Further, it facilitates the evaluation of the performance based on different configurations and costs. EVAL-extension attributes are added to the engine as features. The features are dynamically added based on the input files and thus only relevant features for the instance are included in the engine. A feature is, in simple terms, responsible for everything needed to be able to calculate the cost related to the attribute.



(a) The flow of the generating solutions



(b) The flow of parsing results

Figure 4.1: An outline of the flow of the implemented solution.

Algorithm flowchart

The algorithm component of the framework is constructed in a manner which makes the algorithms applied easy to substitute. The flow of the algorithm component is shown in Figure 4.2. The first step is to construct an initial solution. If previous solutions have been constructed, it is possible to include knowledge about the problem, learned in previous iterations. Once an initial solution has been constructed, an improvement algorithm is applied. On termination of the improvement algorithm, the final solution is saved, and the problem knowledge is updated. If more iterations are to be performed, either a permutation is applied to a previous solution or a new initial solution is constructed. Note that it is possible to substitute the algorithms applied at each step. As a result, the framework is compatible with multiple combinations of algorithms.

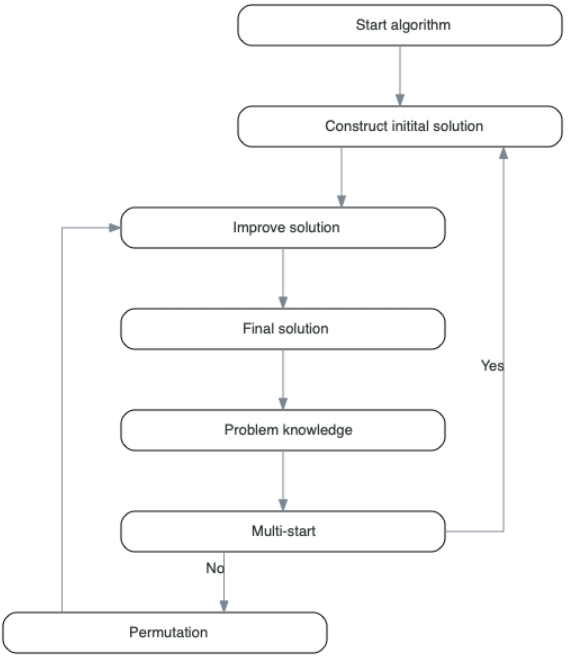


Figure 4.2: A flowchart of the general procedure for the algorithm component.

4.2 Implemented algorithm

This section describes the implemented algorithm in detail. A flowchart of the procedure is presented in Figure 4.3. At the beginning of each iteration, the size of the fleet is decided. If a previous solution including the selected number of vehicles exists, the best solution found is used as a starting point. When running multiple processes in parallel the best solution of all processes is used. If no previous solution exists, a new initial solution is generated. Local search is then applied to the existing solution. It begins with a permutation phase in which the penalties applied to constraint violations are lessened and deteriorating solutions are excepted. This is performed for a set number of iterations. When the permutation is completed, a local search phase is performed in which only improvements are accepted. The penalties applied to constraint violations gradually increase during this phase. When the improvement phase is completed, either a new permutation is performed or a new iteration is started.

Fleet size

It is difficult to construct a general algorithm for optimizing the fleet size as the optimal search pattern is heavily dependent on the problem formulation. Thus, a very

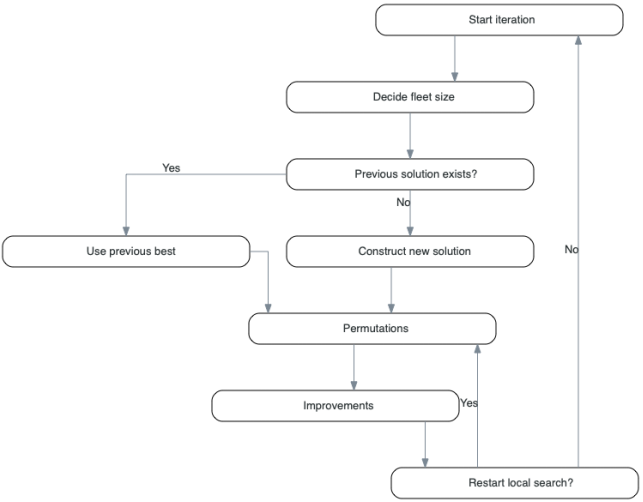


Figure 4.3: A flowchart outlining the implemented algorithm.

intuitive and simplistic algorithm was constructed which can easily be modified to improve the performance on specific problem instances. First, a lower bound for the number of vehicles used is established. The lower bound can be found through, for example, calculating the total capacity needed (if all vehicles have a max capacity) or by summing the time it takes to travel to the second closest node (if all vehicles have a max time). An upper bound is defined as the total number of vehicles available. The algorithm consists of two phases, the initial search, and the main search.

For the first solution created during the initial search, the number of vehicles used is set to equal the lower bound. Then, for each following iteration, the number of vehicles included is incremented by one. This is continued until κ solutions worse than the previous best have been found. The initial search is then completed, and the main search phase is started.

In the main search phase the number of vehicles to be used in the iteration is chosen according to:

1. Select $n_{init} \in \mathbb{N}$ at random with the probability

$$p(n_{init}) = \frac{(\frac{C_n^{min}}{C_{best}^{min}})^{\theta}}{\sum_{n' \in N}(\frac{C_{n'}^{min}}{C_{best}^{min}})^{\theta}}$$

where C_n^{min} is the minimum cost found using n vehicles, C_{best}^{min} is the lowest cost found for all values of n, N is the total set of the number of vehicles examined and θ is a constant.

2. Obtain the actual number of vehicles to use from the function $n_{final} = m(n_{init})$ where $m : \mathbb{N} \rightarrow \mathbb{N}$.

The parameters can be modified to improve the performance. κ decides how fast the initial search should be stopped, θ decides the weight of the previous results, and m is a function that can modify the number of vehicles used. m can be viewed as a method for investigating the regions around the best-found solutions. The reasoning behind including such a function is that, for some instances, a lower number of vehicles is expected to be preferred. It might, however, be difficult to find a good solution for the lower number of vehicles as a large portion of the solutions is infeasible.

Generation of initial solution

The generation of the initial solution is performed only when no previous solution for the selected number of vehicles exists. The first step of generating the initial solution is to decide upon the fleet composition. All vehicles can differ either in their properties or set of allowed depots. Hence, two vehicles will be handled as equal if they have the same properties and depots. As a result, the first step is to classify the vehicles accordingly. The construction of the initial solution follows a naive approach where the fleet composition (the type of the vehicles) is chosen at random. Thereafter, a random node is selected and inserted into a route according to the cheapest insertion principle. If the insertion results in the exceedance of the fuel limit, all chargers are removed and reinserted. The reinsertion is done by inserting a charger right before the node for which the fuel limitation was exceeded. If the insertion of a charger is possible, the cheapest allowed charger is inserted. If no charger is possible to insert, the same procedure is repeated for the previously visited node. This is done until either a charger is inserted or the previous charger or depot is reached. If no charger was found the best available charger is inserted before the node which exceeded the fuel limitation.

Local search

The local search phase is composed of two sub-phases, denoted the permutation phase and the improvement phase. Permutations are always applied at the beginning of the iteration phase and are followed by the improvement phase. This cycle, of first applying permutations and then improvement operators, can be repeated multiple times within a local search phase. The same operators are performed within both sub-phases. The inter route operators are applied stochastically, with each operator assigned a probability to be selected. During the permutation phase, the penalty of exceeding a constraint is decreased and a deteriorating solution can be accepted. During the improvement phase, only solutions improving the final cost are accepted. The intra route operators are applied to a route that has been modified by an inter route operator. A more detailed description of how the operators are applied can be found in Appendix A.

Optimizing the amount charged

When optimizing the charging routes with chargers included will be assumed to be supplied. The optimal amount to charge is estimated by first charging the amount required to reach the next charger or depot with the required SoC. Then continue charging while the current charging speed is faster than the charging speed of the next charger, keeping in mind the SoC at each charger. For the time-dependent case, the task of optimizing the amount to charge becomes more difficult. However, in this thesis, the same procedure as in the time-independent case will be used. The problematic aspects of this approach are described in section 3.4. The reasoning behind not developing a more sophisticated method is partly due to the limited time of the project and partly due to that it is unclear if the inclusions of such a method would improve the performance noticeably. The fact that the allowed computational time for the optimization is restricted limits the possibility of using numerical methods.

4.3 Motivation for the implemented algorithm

The implemented algorithm is motivated by its flexibility and the fact that it works on a wide range of problem extensions. The operators included can easily be extended, modified, or substituted to adapt and improve the algorithm to a new problem setting. The number of vehicles to include in the solution can be known in advance or unknown. In order to handle both these cases, the decision regarding the fleet size is handled in a separate phase. It should be noted, that in the current implementation, a vehicle can be included in a solution without being assigned a customer. When no customers are assigned to a vehicle, and it has the same start and end depot, it is excluded from the cost of the solution. As a result, the number of vehicles gradually decreases. Note that since the number of vehicles to include in an iteration is stochastically chosen, the reduction of vehicles will result in a larger probability of selecting a reasonable number of vehicles. Thus, the solutions using the same number of vehicles, while initially being assigned a different number of vehicles can be viewed as a method for diverging the search within that fleet size.

The algorithm is conceptually easy to understand and the different phases have a clear purpose. While there are many parameters to be set, how these influence the solution is easy to comprehend. Penalty functions are used to make the search space easier to traverse through. In many cases, it would otherwise be difficult to modify a solution without making the new solution unfeasible. The magnitude of the penalties can be modified. This allows for the usage of the same operators within the permutation and the improvement phase. Further, the algorithm is parallelizable, which allows it to run on multiple CPU cores simultaneously.

A lot of effort was put into modifying an existing solution and a new solution is only constructed when no previous solution is available. Thus, the approach for constructing new solutions is simple and does not produce initial solutions of high quality. However, it is very flexible and easy to adapt to most extensions. This com-

promise was deemed appropriate since the quality of the initial solution quickly becomes unimportant.

An implementation of ACO was performed, to enable evaluating the pros and cons of the implemented algorithm. One of the drawbacks of ACO was the difficulty of finding feasible or close to feasible initial solutions. Therefore, it was costly to generate enough feasible solutions for the pheromone trails to become effective. Further, if the routes are constructed sequentially, a stopping condition is required. In some instances, there are no apparent stopping conditions. For others, the amount charged might make it more complex to handle. The optimal next vertex to visit depends on the current state of the solution. As a result, given the inclusion of multiple attributes, a single pheromone trail becomes inadequate for transferring the problem knowledge. One method for counteracting this could be to create several pheromone trails which are weighted based on the state at a node visit. But, in this method, the weighting and updating of the pheromone trails would be challenging to design.

4.4 Improving the performance

In this section, multiple methods for improving the performance of the algorithm are presented.

Parameter tuning

The performance of the algorithm is heavily dependent on the chosen parameter settings. In the following paragraphs, the settable parameters of the algorithm are presented. No guidelines for the parameter values are given here, as this is discussed in section 5.1.

Local search parameters In the local search phase, the parameters to be set can be divided into different categories. The first category is parameters that are related to altering the cost functions and acceptance rate during the local search phase. This includes the number of local search iterations during the permutation phase, the number of permutation phases, the acceptance rate during the permutation phase, the relaxation of penalty functions during the permutation phase, the relaxation schema during the improvement phase, and the number of iterations within an improvement phase. Further, when to apply intra route operators and stop conditions must also be set. The second category of parameters is those related to the operators used. Parameters to be set within the category are the probability for applying an operator and parameters specific to the different operators, such as deciding which routes and nodes to include.

Other parameters There are a couple of parameters to be set, in addition to those in the local search phase. Firstly, the total number of iterations must be specified.

Alternatively, this could be defined as allowed execution time or a number of iterations without improvement. Secondly, parameters related to fleet construction, the constructive method, and the limiting of look-ups for larger instances must be decided.

Improving the robustness

In order to improve the robustness of a solution , which was discussed in section 3.2, a negative cost was implemented. The negative cost is optional to include and in the presented computational results it is only included in section 5.3. The negative cost was implemented according to the formula:

$$c(x,\beta) = -\alpha\frac{x}{\beta+x}, \quad 0 \leq \alpha,\beta,x$$

where α is a scalar, β is a shape parameter and x is the amount left of the resource. Note that $c(x,\beta) = \frac{x}{\beta+x}$ is a function that takes on a value between 0 and 1 where $c(0,\beta) = 0$ and $c(\infty,\beta) = 1$, for all $\beta < \infty$. Further, the property of the shape parameter β can be viewed as $c(x,\beta) = 0.5$ given $\beta = x$. In other words, the function is non-decreasing, concave and yields a result between 0 and 1. By using negative functions cheap routes could potentially have a negative cost or a cost close to zero. Thus, the function actually implemented in the algorithm was on the following form:

$$c(x,\beta) = \alpha(1-\frac{x}{\beta+x}), \quad 0 \leq \alpha,\beta,x.$$

Limiting the number of look-ups and selecting how to apply operators

This section covers the implementational aspects of the topic of handling large instances, discussed in section 3.6. In summary, two problems arise when working with large datasets. Firstly, the number of attributes that must be approximated quickly grows, and secondly, the execution time will increase as the solution space increases. The two problems are in some aspects related but will be treated separately as they are handled differently within the implementation.

The first problem is related to the modeling of the problem and is treated when the dataset is constructed. In essence, the implemented solution to the problem is to reduce the number of arcs included in the model and set all arcs excluded to have an infinite cost. Using this approach the difficulty lies in designing an effective method for choosing how many and which arcs to include. The approach taken in this report is to sort the other nodes based on distance and only model those nodes that are the closest.

The second problem is a bit more intricate. The first thing to notice is that, as explained in section 3.4, the computational complexity of evaluating a solution is relatively high. It can however for some instances be improved, which is discussed in the section *improving the evaluation time*. Given the high computational cost

of evaluating a solution, the solutions to be evaluated must be chosen with care. Therefore, all operators are equipped with methods for deciding which routes and nodes to apply them to. Still, designing effective operators is challenging. For all implemented operators, the choice of which routes and nodes to apply it to is done probabilistically. For some operators, the probability of including targeting a node is relative to the cost of the node. As the performance of the solver is heavily dependent on the problem instance, it is difficult to conclude that this improves the performance. However, intuitively it should at least lead to an initially faster cost decline as the most expensive nodes have the highest impact on the cost. With regards to which routes to apply the operators on, this is for most operator chosen at random, except for the insert operators which have a higher probability of removing a node from a costly route. The exact value for the operators' parameters can be found in Appendix B.2

Improving the evaluation time

Evaluating a solution is costly in the implemented framework, see section 3.4. There are multiple methods and constraints which could be used to decrease the computational complexity of evaluating a solution. The motivation for not implementing these by default is that the solver should be as flexible as possible, at the cost of execution time. It is however possible, through small modifications, to allow for quicker evaluation. This can, for example, be achieved with an initial filter using constraints which never are allowed to be exceeded. For example, it is possible to perform an initial check to assure that the load capacity is never exceeded, the increase in distance or time is never too large, or that the SoC for the new route never is below zero. While the computation complexity is decreased, the ability to allow for unfeasible solutions becomes limited, which might affect the performance negatively.

Adaptive parameters

Adaptive parameters can be adapted dynamically to the problem settings before the execution based on the properties of the problem, for example depending on the number or density of the customers. Alternatively, the parameters can adapt during the execution based on the performance of the solver. In this report, no adaptive behavior will be investigated further but it is presented as a possible method of improving the performance.

4.5 Examples applications

In this section, the application of the algorithm to a few selected VRP extensions is discussed. These extensions are not exclusive and should be viewed as examples of how the algorithm handles or can be modified to handle different variations.

Pick-up delivery E-VRP with time windows

For the pick-up delivery E-VRP with time windows customers are divided into pairs where the same vehicles must visit both nodes and the customer labeled pick-up must be visited first. During the construction of the initial solution, the next node to be inserted is only chosen from the set of customers labeled as pick-ups. After inserting the pick-up customer, the delivery customer is inserted in the same route, at the position after the pick-up customer which minimizes the cost. During the local search phase, only operators moving the nodes as a pair while retaining their order are allowed.

Fixed size heterogeneous fleet multi-depot E-VRP

The fixed size heterogeneous fleet multi-depot E-VRP is important to consider as this extension arises when a solution is partially resolved. In practice, there are not too many differences between solving the regular E-VRP. The solver can handle different vehicle types with different start positions. An operator which selects the best endpoint is added to optimize the selection of end depot. If a vehicle is unused or has already finished its route the cost while waiting for an assignment is in all presented solutions set to zero. It should be noted that resolving a partial solution is easier to solve as the vehicle composition is already decided on and fewer customers are left to visit.

Time-dependent vehicle routing problem

The time-dependent vehicle routing problem is solved without major alterations in the algorithm. The only time dependence accounted for is in travel time and fuel consumption, but it is possible to implement additional time dependencies such as waiting time at charging stations. A difficulty when dealing with time dependence is to decide on the optimal charge amount. It is difficult to optimize the amount to charge as there is a co-dependence between the charge amount, the travel time, and the fuel consumption. While numerical methods could be used, the frequency of calculations is high whereby a low computational complexity is required. Therefore, the same method as in the non-time-dependent case is used. Given a relatively modest difference in departure time, the fuel consumption and travel time are expected to be relatively constant. Therefore, the time-independent case is justified to be a reasonable approximation.

4.6 Evaluating the quality of a solution

A central aspect of solving the E-VRP for practical applications is to evaluate the solutions obtained reliably. In this report we do not concern ourselves with checking for feasibility as exceedances of constraints will be treated using penalty functions. Thus, an infeasible solution is expected to be costly.

A web-based dashboard was developed to facilitate the assessment of obtained solutions. In the dashboard, different problem parameters can be modified. The main purpose of the dashboard is to get additional information about each solution, for example through displaying the solutions as a graph, showing how much each attribute contributes to the cost, and plotting the behavior of the attributes in the solution. Further, it is possible to simulate a solution with modification to the arcs to investigate the robustness of the solution. The modified arcs can further be used to generate new datasets and to change the safety margin used. Another functionality in the dashboard is to turn on and off congestion to see how a solution is affected by the inclusion or absence of the includes of congestion.

As previously discussed a generated solution for the E-VRP is seldom dominating all other solutions. The Pareto optimally can often be motivated in multiple ways, but one way to reason about it is that the solution that optimizes the objective function seldom is the most robust. In this section, we will not concern ourselves with how a set of proposed solutions was found but will instead discuss how one can analyze the solutions.

Assessing the robustness There is no single definition for the robustness of a solution and therefore several methods will be used to investigate the robustness. The first method is to apply a safety margin. A safety margin could be applied to the fuel limit as a minimum allowed SoC. As discussed previously, a safety margin could be applied when constructing the solutions, but it can also be used to assess the quality of an already constructed solution by keeping the same solution but increasing the minimum allowed SoC. A solution that performs well given an increase in required SoC can be argued to be more flexible as one has the option to increase the amount charged if the weather conditions are worse than predicted or the fuel consumption is higher than expected for other reasons.

As estimating the required time and fuel consumption needed to travel between two locations is a difficult task, it is of interest to investigate what happens if the estimated travel time or fuel consumption is slightly off. The more advanced method would be to construct models for each arc which includes a probability density function for different values of all attributes. As these probability density functions are seldom available a more naive approach is used. The naive approach uses a well-known probability distribution and multiplies the attributes of a route with a random value drawn from the probability function. For instance, a normal distribution with the mean value of one and a low standard deviation could be used to model slight variations in the arc attributes. Of course, it is possible to use any distribution and only apply the modifications to a selected number of arcs.

Assessing solutions with a different objective function It can be of interest to apply different objective functions to the solution. As discussed in section 2.1 there can be multiple objective functions that are conflicting and thus multiple Pareto optimal solutions are produced. The properties of a produced solution can be investigated using other different cost functions.

5

Computational results

In this chapter computational results from the framework are presented. Each section will be concerned with a specific problem extension. All sections, except the first, will follow the same structure where first an introduction to the problem is given, followed by a description of the parameter settings and algorithms used. Lastly, the results of executing the algorithm will be presented. The cost functions and solver parameters are presented in Appendix B. For the case of simplicity, the same cost and penalty functions will be used for all instances. Any modifications will be discussed explicitly.

For most graphs presented, the path from the start and end depot will be excluded, since the graph otherwise becomes cluttered. In all presented graphs red markers are depots, blue markers are customers and the green markers are chargers.

5.1 Parameter tuning

In this section computational results on the topic of parameter settings are presented. Note that the computations are only performed on a limited set of problems and it is thus unclear to what degree the results can be generalized. The instances used in this section are taken from Montoya et al. (2017). The investigated settings are presented in Table 5.1 and the results are presented in Table 5.2. The computational experiment is only performed on a selected set of parameters as the number of combinations otherwise would be unmanageable. The parameters to be investigated were selected based on the expected impact on the performance and the values used for the remaining parameters can be found in Appendix B.

As previously mentioned, one should be careful with drawing conclusions from the limited sample size. The result does however indicate some behaviors that intuitively make sense. Since the area for all datasets is equal no matter the number of customers included the density of customers will be higher for datasets with more customers. As a result, it can be expected that the permutation steps do not have to be as strong since it should in general be easier to escape a local minimum, which

Table 5.1: The settings used in the parameter tuning.

Name	1	2	3
Restart iterations	10	3	3
Restart softness	0.5	0.2	0.2
Restart acceptance	0.8	0.9	0.9
Permutation iterations	10	5	20
Permutation softness	0.4	0.4	0.1
Permutation acceptance	0.9	0.95	0.95
Start softness local search	0.4	0	0
End softness local search	-0.6	0	0

Table 5.2: The cost obtained from running the algorithm with different settings. The name indicates how many customers were present in the dataset.

Dataset	Settings	Cost
$c20_1$	1	4678
$c20_1$	2	4674
$c20_1$	3	4674
$c20_2$	1	4860
$c20_2$	2	4860
$c20_2$	3	4860
$c40_1$	1	9818
$c40_1$	2	9818
$c40_1$	3	9810
$c40_2$	1	8832
$c40_2$	2	8967
$c40_2$	3	8832
$c80_1$	1	14276
$c80_1$	2	14226
$c80_1$	3	14189
$c80_2$	1	15511
$c80_2$	2	15553
$c80_2$	3	15550
$c160_1$	1	28335
$c160_1$	2	28199
$c160_1$	3	28181
$c160_2$	1	26273
$c160_2$	2	26143
$c160_2$	3	26039

the result also indicates. Although the results are ambiguous, the third setting seems to perform the best and will therefore be used in the remainder of this chapter.

5.2 Benchmarks

In this section, the performance of the algorithm on two benchmarks is presented.

E-VRP - Monotoya

In Table 5.3 a comparison with the dataset and result presented by Montoya et al. (2017) is performed. The objective of the dataset is to minimize the total traveling time for the standard E-VRP with a maximum allowed travel time. It should be noted that the solver presented in this thesis works best with soft constraints which are not used by Montoya et al. The result presented is therefore not strictly comparable. The algorithm was therefore executed a second time, with cost functions that penalize exceedances more strongly. As a result, none of the best solutions found will exceed any of the constraints. The result is presented in Table 5.3 where the column named soft constraints uses the cost functions presented in Appendix B and the column named *hard constraints* uses cost functions which never allows exceedances.

The execution time is deliberately left out as it is heavily dependent on the chosen settings and can be changed drastically through small modifications. Nevertheless, as execution time is important in evaluating the performance of the algorithm the following approximations are presented:

- 10 customers takes approximately 60 seconds.
- 20 customers takes approximately 100 seconds.
- 40 customers takes approximately 250 seconds.
- 80 customers takes approximately 900 seconds.
- 160 customers takes approximately 3000 seconds.

Table 5.3: The shortest total travel time found by the presented algorithm compared to the solutions presented in the dataset source.

Dataset	Soft constraints	Hard constraints	Montoya et al.
tc1c10s2ct2	9.37	10.87	10.70
tc1c10s2cf2	7.42	9.03	9.03
tc1c10s2cf3	14.12	16.37	16.37
tc1c10s2cf4	15.51	16.10	16.10
tc1c10s3ct1	10.8	10.8	10.8
tc1c10s3cf2	7.42	9.03	9.03
tc1c10s3cf3	14.12	16.37	16.37
tc1c10s3cf4	14.13	14.90	14.90
tc0c20s3ct2	17.08	17.08	17.08
tc0c20s3cf2	23.26	27.58	27.60
tc0c20s4ct2	16.99	16.99	16.99
tc1c20s4ct3	12.73	14.43	14.43
tc1c20s4ct4	13.34	17.00	17.00
tc0c20s4cf2	23.26	27.58	27.48
tc0c40s5ct0	27.44	29.23	28.72
tc0c40s8ct0	26.22	26.33	26.41
tc0c40s8cf4	27.67	28.29	29.32
tc0c80s8ct0	39.23	41.86	41.90
tc0c80s8ct1	43.46	45.28	45.27
tc0c80s8cf0	38.34	39.39	39.43
tc0c80s8cf1	44.41	45.55	45.23
tc0c160s16ct2	57.89	58.83	60.13
tc0c160s16ct4	75.01	79.41	82.37
tc0c160s16cf4	71.91	78.87	82.92
tc0c160s24cf2	56.55	58.48	59.27
tc1c160s16ct3	65.53	68.16	73.29
tc1c160s24ct3	64.23	66.97	68.72
tc1c160s24cf3	63.66	67.53	68.56

For all instances, the algorithm with soft constraints is equal in performance or outperforms the two others. This is to be expected as the problem formulation is less restrictive. The fact that it manages to find better solutions shows the usefulness of allowing soft constraints as it can be beneficial to exceed a constraint. The implemented algorithm with hard constraints is relatively close to the benchmark algorithm. For most instances of smaller size, the same or slightly worse results were obtained. For instances of larger size, the implemented algorithm seems to generally outperform the benchmark algorithm. It should be stated that the execution time of the benchmark algorithm is much faster and that for some instances the

presented algorithm was unable to find a feasible solution. The implemented algorithm is however more flexible as the benchmark algorithm would be inapplicable to most of the other extensions presented in this chapter.

VRP-TW - Solomon

The dataset from Solomon (1987) is one of the most commonly used benchmarks for the VRP-TW. The objective is to minimize the distance traveled and the result of running the solver on the dataset is presented in Table 5.4. For this benchmark, the cost for traveling time is excluded and instead a cost for the traveled distance is included.

Table 5.4: The shortest distance found by the implemented solver compared to the best known solutions.

Dataset	Best found distance	Best known solution
RC201_025	361.2	360.2
RC201_050	686.3	684.8
RC201_100	1312.7	1261.8
RC202_025	338.8	338.0
RC202_050	621.5	613.6
RC202_100	1135.4	1092.3

The result shows that the solver does not find the optimal routes for any of the datasets. It does however propose solutions that are reasonably close to the best known values and manages to find feasible solutions for all investigated datasets. No constraints were exceeded in the presented solutions.

5.3 Generated datasets

As the E-VRP is a fairly new extension the number of publicly available datasets is limited. Therefore, custom datasets were generated to enable the investigation of more aspects and possibilities of the algorithm. For all instances presented all chargers are of similar type.

Time-dependent E-VRP

Time dependence was included through varying travel times and fuel consumption depending on at what time the arc was traversed. This time dependence will be denoted as congestion. For simplicity, the start time was always chosen based on the departure time. For this instance, the travel times and fuel consumption were in half of the instances unmodified, in one-fourth of the instances increased using a discretized normal distribution, and in one-fourth of the instances increased using

a distribution which has the highest value in the tails and lowest in the center. This can be viewed in Figure 5.1. Note that the congestion is only affecting the solution negatively. This can be argued to be unrealistic in practice as a higher value would be selected for the static case. Still, this is ignored as the purpose is to show that the solver can account for time-dependent variables, and not to reasonably model congestion.

In Table 5.5 the result of including or excluding congestion or not in the algorithm execution is presented. The algorithm execution is separated from the calculation of the final result which enables running the algorithm without accounting for congestion. This enables investigating how the proposed solution would perform if congestion was present. Two types of costs are presented, static cost and time-dependent (TD) cost. The static cost is the cost without congestion and the TD cost is the cost with congestion included. All datasets presented are containing one depot, fifty customers, and eight chargers.

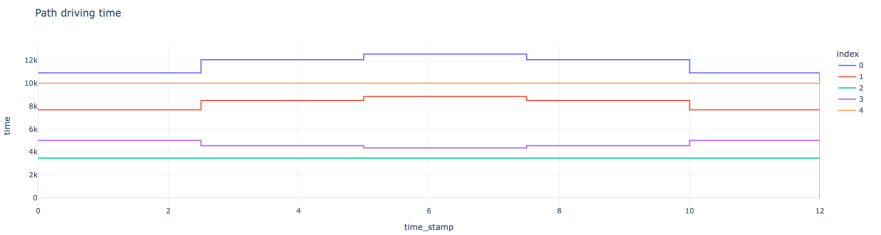


Figure 5.1: Time dependent driving times based on how many hours from the start time of the delivery the arc is traversed (timestamp). Each index represents an arc between two nodes and only the first five arcs are displayed. The time unit on the y-axis is seconds.

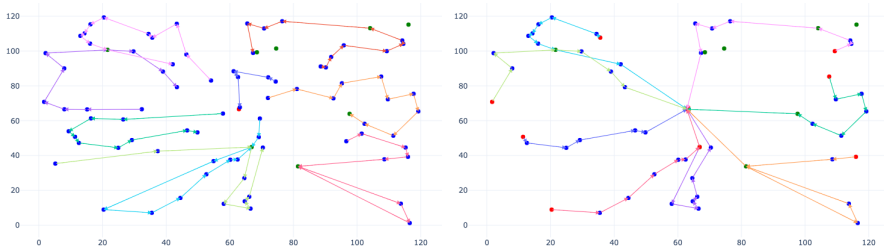
Table 5.5: Results obtained from two algorithm execution. The columns indicate if congestion was accounted for in the algorithm execution (TD-alg), the cost without congestion (static cost) and the cost with congestion (TD cost). The column optimal indicates whether the solution was optimal for the static or TD case.

Dataset	Optimal	TD-alg	Static cost	TD cost
1	static	No	6654	18348
1	TD	No	6837	8007
1	static	Yes	6973	7734
1	TD	Yes	6973	7734
2	static	No	6071	16024
2	TD	No	6421	8144
2	static	Yes	6614	7653
2	TD	Yes	6767	7505

The results indicate that the solutions obtained are better for the time-dependent case when it is included in the algorithm and the static case when excluded. This should be the expected behavior. Further, it is clear that when applying time-dependent variables to the optimal static solution the cost is significantly increased. This behavior can be expected as the optimal solution often has a trade-off in robustness.

Dynamic E-VRP

To handle dynamic settings, a snapshot of a partly solved dataset can be generated. The new instance is unaware of the previous solution. An example of a resolved problem is presented in Figure 5.2. For this instance, the state at the next node visited after three hours was saved and used to generate the new problem instance. All vehicles in the new dataset will have different starting positions and different starting times. As nothing more in the data was changed one can expect the new solution to be similar to the previous, which is also the case as the two produced solutions are identical.

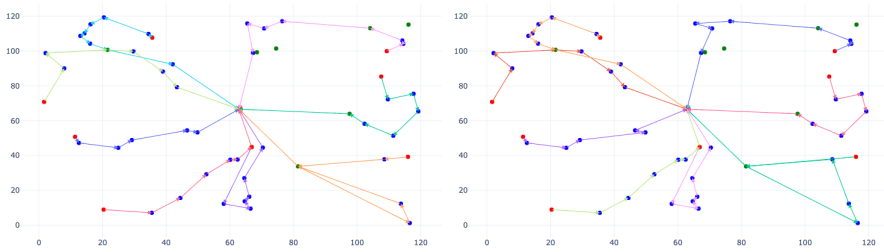


(a) Original solution

(b) Re-solved from snapshot.

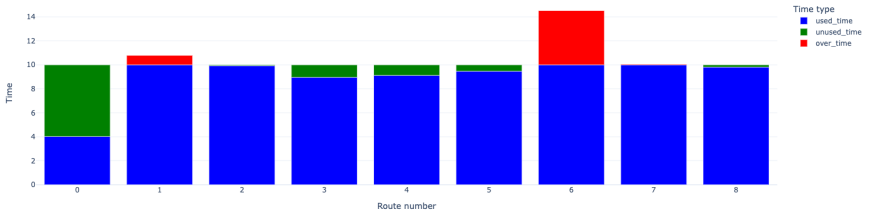
Figure 5.2: A snapshot was taken part-way through the solution and the new problem was solved without any knowledge about the previous solution. Note that in the original solution all vehicles start and finished in the same depot. However, the first and last route has been removed as the graph would be cluttered around the depot.

The partial solution can be used to handle changes in the input data. To investigate how the solver handles changes in the input data the ability to modify an existing dataset was implemented. The modified dataset increases the travel time of five random arcs, included in the obtained solution, with a factor of five. The quality of the original solution is decreased, with the total cost changing from 14330 to 16664 (16% increase). The best found solution for online-optimization on the modified instance problem has a total cost of 14573 and the routes chosen are presented in Figure 5.3. In the same figure, the maximum travel time for the original solution on the modified dataset is also presented together with the maximum travel time for the online calculated solution on the modified dataset.

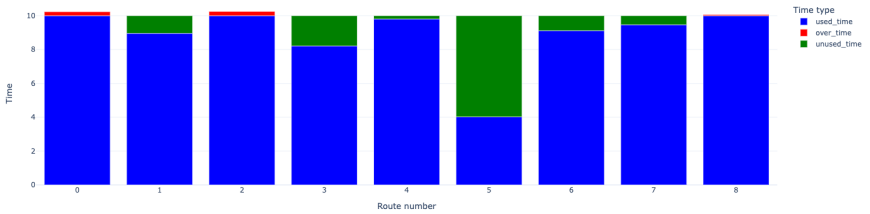


(a) Re-solved, original dataset.

(b) Re-solved, modified dataset



(c) Maximum travel time original solution

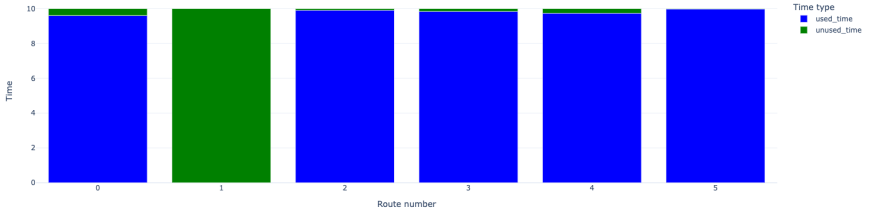


(d) Maximum travel time modified solution

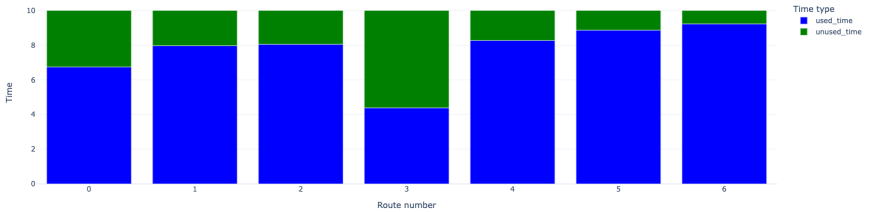
Figure 5.3: Results comparing the original solution and the modified solution applied on the modified dataset. Note that in, (c) and (d), the number of the route does not coincide between the two datasets.

Robust E-VRP

To generate more robust solutions, the way of handling of maximum travel time was modified according to section 4.4. The dataset presented was randomly chosen from the benchmarks presented by Montoya et al. (2017). For the maximum travel time $\alpha = 600$ and $\beta = 2 * 3600$. The maximum time of a regular solution and a robust solution is presented in Figure 5.4 and shows that there are larger safety margins for the robust solution.



(a) Total travel time - Regular



(b) Total travel time - Robust

Figure 5.4: Total travel time for the vehicles in a regular and robust solution. For this instance, ten hours is the maximum time.

In Table 5.6 computational result for the regular and robust solutions are presented. It presents the average cost of 10 000 simulations where the travel time of all arcs are multiplied with random variables independently drawn from a normal distribution. The mean and standard deviation of the normal distribution is presented in the table. The results indicate that the robust solution performs best when the variables are drawn from a normal distribution with a high mean.

Table 5.6: The average cost of the regular or robust solution when multiplying the travel time of the arcs with variables randomly drawn from a normal distribution.

Mean	Std	Regular	Robust
1.00	0	8880	9131
1.05	0.05	9137	9319
1.10	0.05	9602	9647
1.15	0.05	10117	10022
1.25	0.05	11150	10854

Pickup and delivery E-VRP with time windows

In Figure 5.5 a solution obtained to the pickup and delivery E-VRP with time windows is displayed. The pickup customers are blue and the delivery customers are orange. Each pickup customer is assigned a delivery customer, which the same vehicle must visit. Although not clear from the figure, this constraint is fulfilled and no time windows are violated. It is difficult to evaluate the quality of the solution presented but the performance could be improved through constructing operators specifically for this extension.

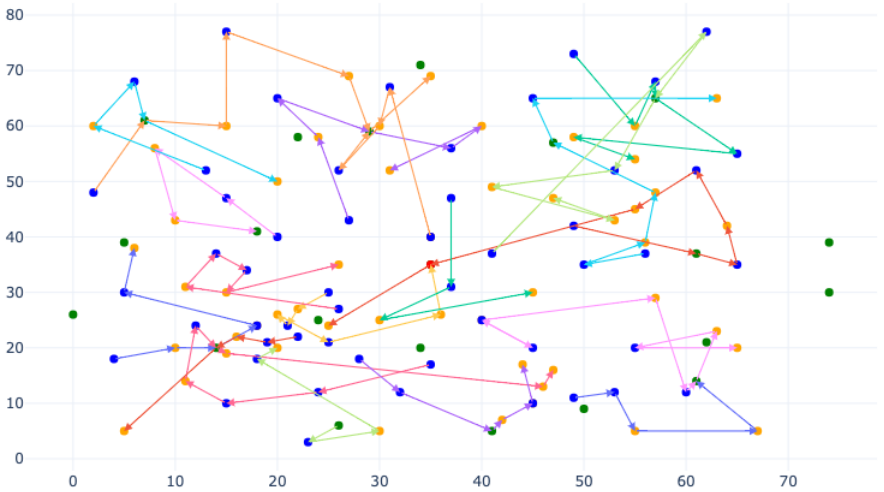
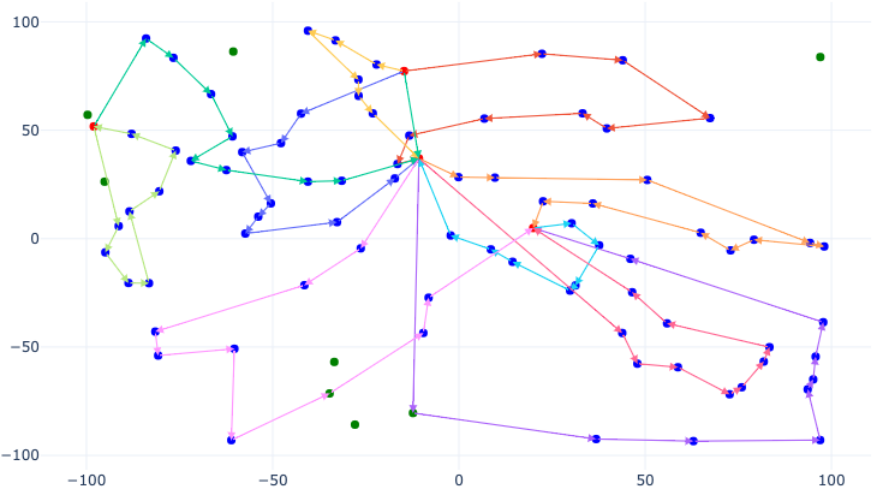


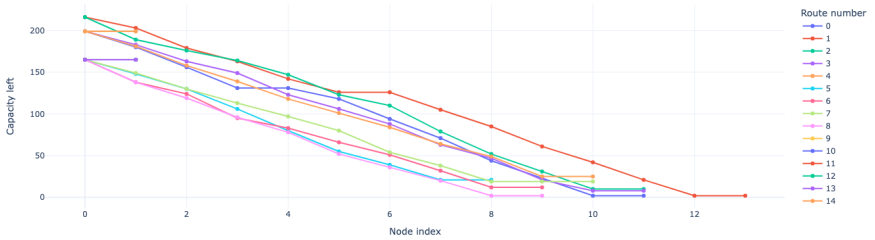
Figure 5.5: A solution obtained to the Pickup and Delivery E-VRP with time window.

Multi-depot heterogeneous fleet E-VRP

In Figure 5.6 a solution to the MD-HF-E-VRP is presented. In the generated dataset twenty vehicles of three different types were included, which can be seen in the plot of the vehicle capacity. Each vehicle was randomly assigned one start depot and two end depots. Th quality of the solution is hard to evaluate, but it seems resonable.



(a) A solution produced by the solver.



(b) The capacity left for each vehicle in the presented solution.

Figure 5.6: A solution to the Multi-Depot Heterogeneous fleet E-VRP.

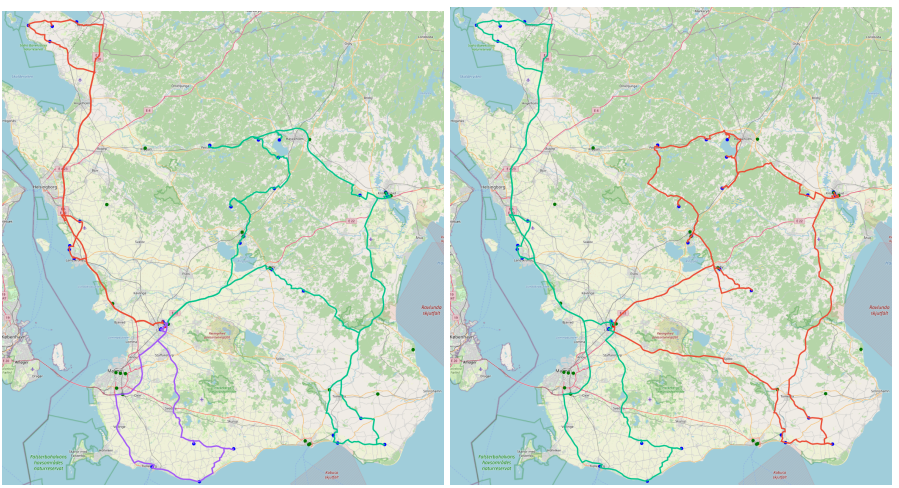
5.4 Real-world instances

This section presents real-world examples of applying the framework in combination with the API of Iternio Planning AB. The solutions proposed contain complete driving instructions. All instances were computed using a Nissan E-NV200 (40kWh). In comparison with the previous examples, the instances presented are relatively small in size and easy to solve as the main purpose is to show that it is possible to use the application in real-world settings. No information is provided regarding the settings used in the computation as the purpose is not to evaluate the performance of the solver. The chargers are located using the API of Iternio Plan-

ning AB and all chargers are set to be of an equal type, namely fast chargers. Note that the colors between the map plots and attribute plots are unrelated and do not coincide.

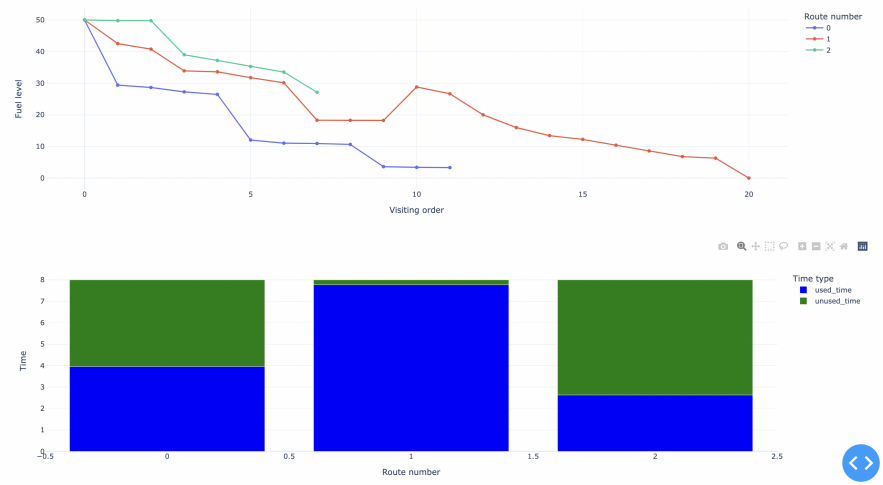
Landmarks in Scania

A set of landmarks located in Scania (Skåne) were selected as the locations to visit. For this problem two solutions will be presented. One with the initial SoC is 50% and one with the initial SoC is 90%. The only attribute assigned to the locations visited is a service time drawn from $\max[1, N(5,2)]$ minutes. Only one vehicle type is used with a maximum travel time of eight hours. The total number of locations to visit is 34, there are 20 chargers and only one depot. In Figure 5.7 the routes presented by the solver are displayed and in Figure 5.8 the fuel level and maximum travel time can be viewed. As the problem is relatively simple there is not too much to discuss regarding the quality of the solution more than that they look reasonable. With 50% initial SoC, three vehicles are used where only one of the vehicles visited a charging station (in Kristianstad). With an initial SoC of 90% only two vehicles were needed and no charging stops were required.

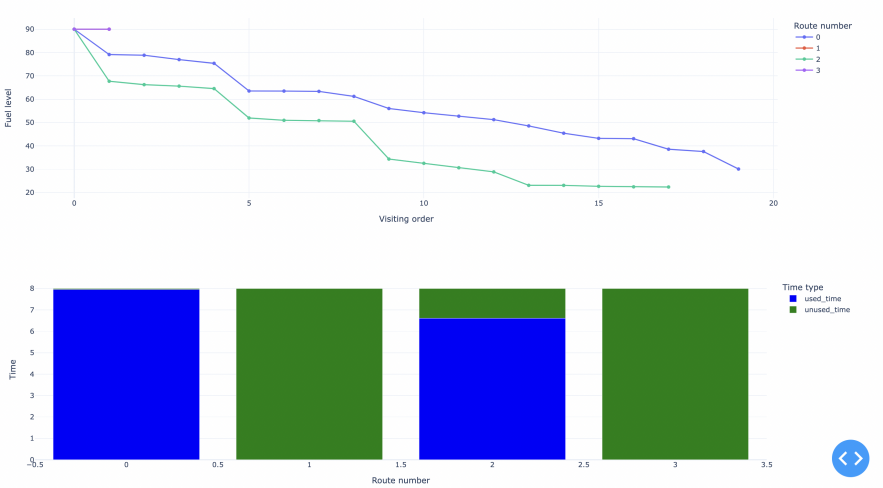


(a) 50% initial SoC (b) 90% initial SoC

Figure 5.7: The routes provided by the solver, plotted on a map of Scania.



(a) 50% initial SoC



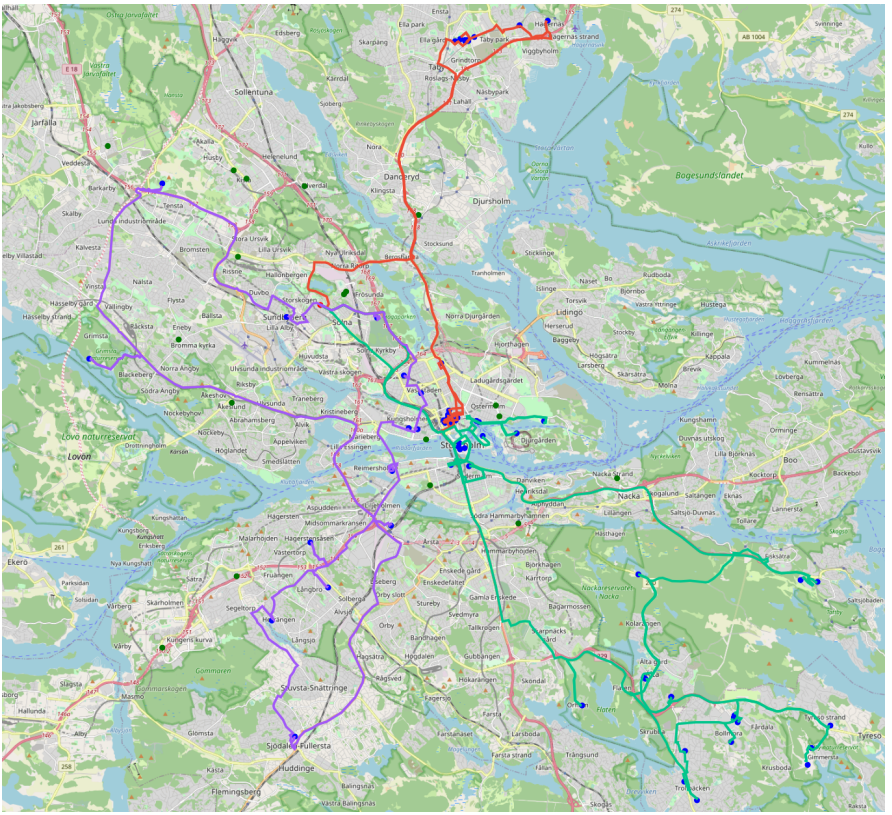
(b) 90% initial SoC

Figure 5.8: The status of the fuel level (SoC) and the total travel time of each vehicle in the Scania problem setting.

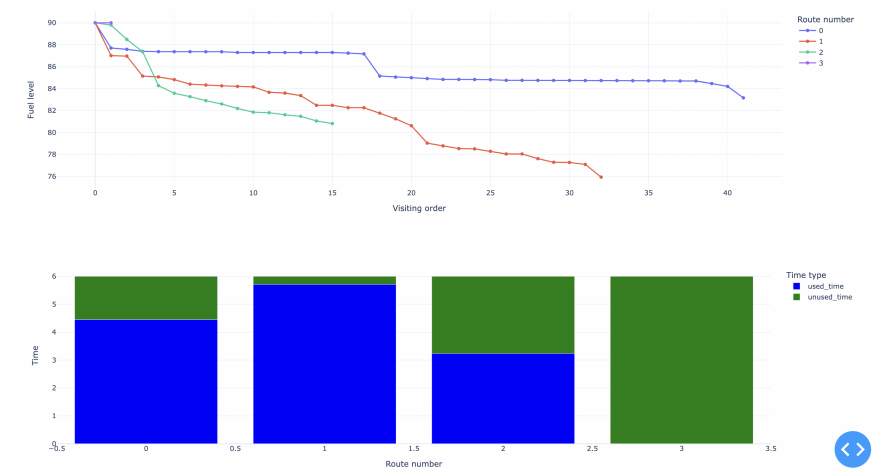
Coffee-shops in Stockholm

For this problem, a set of coffee shops located in Stockholm was chosen as the locations to visit. Each coffee shop was assigned a service time randomly drawn from $\max[1, N(5,2)]$ minutes. Only one type of vehicle was used with a maximum time of six hours and an initial SoC of 90%. The total number of customers to visit

is eighty-five, there are twenty chargers and only one depot. The result is shown in Figure 5.9.



(a) The routes provided by the solver, plotted on a map



(b) The status of the fuel level (SoC) and the total travel time of each vehicle

Figure 5.9: Applying the algorithm to a real-world setting in Stockholm.

6

Conclusion

In this thesis, a framework for solving the E-VRP has been developed. It is intended to be included in the services provided by Iternio Planning AB. To allow for solving a wide range of customer requests, much effort has been devoted to making the framework flexible. Further, as the framework should be practical to real-world problems aspects such as uncertainty and time dependence have been discussed. The developed framework is component-based and can dynamically include and exclude components depending on the provided problem instance. The objective function is situation based whereby configurable costs functions are used. This allows each user to customize the objective function according to their requirements. The algorithm component follows a pre-established flow but the algorithms used in all sub-components are interchangeable. Thus, it is possible to customize the algorithm component depending on the requirements of the user. When a set of solutions has been generated, the best solution is to be selected. For this purpose, a user interface was developed in which the user can analyze the proposed solutions. While the choice of solution could be as easy as picking the solution with the minimum value for the objective function, other aspects such as robustness can be accounted for with the help of the interface.

There are multiple methods for constructing an algorithm that can solve a wide variety of E-VRP extensions. One method is to utilize common properties of specific extensions and generalize the extensions. Another approach is to use a component-based framework for which components can be dynamically included and excluded. The algorithm can not be specifically tailored to certain extensions unless this is done in an excludable component. Although the solver has been developed as general and extendable as possible some extensions might still be cumbersome to include. Therefore, it is important to specify which extensions are most important, to assure that these can be supported. In this thesis, a flexible algorithm is proposed which is capable of solving a wide range of E-VRP extensions. The computational results show that the proposed algorithm performs well compared to other algorithms, constructed specifically for certain extensions. The compared datasets use different objective functions which easily could be adapted too. Fur-

ther, solutions to multiple custom-generated datasets were presented to display the flexibility of the algorithm.

There is no general method for determining the quality of a solution to the E-VRP, as the quality depends on the requirements and costs supplied by the user. Even when user-supplied costs are used, the best solution does not have to coincide with the solution that has the lowest objective function value. It could be that a solution of higher cost is more robust and thus preferable. It can be helpful for the decision-maker to be able to view different aspects of the solution and the attributes to get a better understanding of the property of a solution. Further, simulations can be helpful to investigate the behavior of changes or errors in the input data.

Future research

Possible areas for future research could be to investigate the performance of operator combinations in regards to specific problem instances. Further, new operators could be developed to target specific extensions using problem-specific knowledge. Another topic of investigation is how the performance on different problem instances is affected by the parameter settings. As of now, improvements could also be made to the method for deciding the size of the fleet. The lower bound for the number of vehicles to include is unable to account for some extensions, such as time windows. Additionally, the initial search can sometimes, for larger instances, terminate prematurely whereby too few vehicles are investigated. Using larger steps, a different interval scheme or other parameters for the stage could be a solution.

The performance of the algorithm is highly dependent on the speed of evaluating the cost of a route. As a result, two possible methods for improving the performance are to use problem-specific knowledge to improve the evaluation speed and implement the solver in a faster language. As of current, when the framework is assigned multiple CPU cores, all processes shares the best solutions found and use the globally best solution as a starting point for the next iteration. It would be interesting to allow for using a wider range of solutions as the initial starting point for an iteration as this might increase the exploration and help to escape local minimums. Lastly, it would be interesting to test the proposed route in a real-world setting to ensure that they are reasonable.

On a final note, the E-VRP is still a relatively new problem, but its importance should not be understated. To fully exploit the advantages of electric vehicles in delivery fleets, more work needs to be done. Benchmarks for a wider set of problem formulations must be constructed and the gap between theory and practice must be bridged.

A

Operators

In this appendix, the operators used within the algorithm are described. Which operators to include in an algorithm execution is specified in a configuration file. Further, note that as evaluating the cost of a route is expensive, comprehensive searches are avoided. A greedy approach is therefore applied for all operators, in which the modification is accepted as soon as an improvement is found. Certain extensions might limit the number of applicable operators or require modifications to those presented, for example, the ordering of some nodes might need to be accounted for.

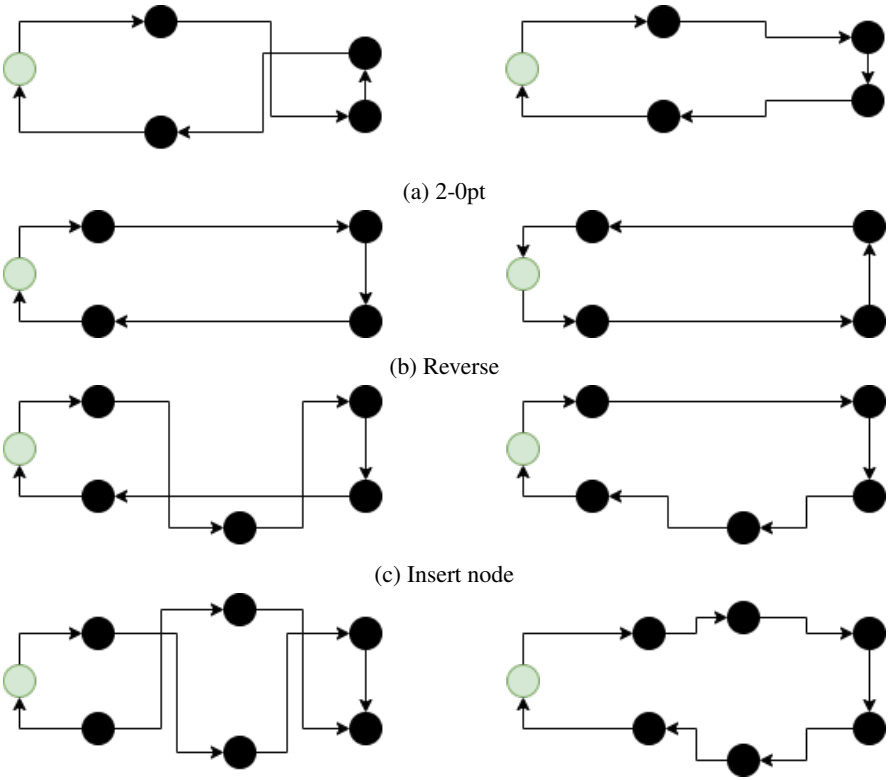
A.1 Intra route operators

Intra route operators alter the order of the vertices visited within a route. In this section the implemented intra-route operators are presented.

Sequence operators

In Figure A.1 a few commonly used intra route operators are presented. The figure in combination with the operator name should be self-explanatory. The operators presented have the common properties that they are simple. One reason for only using simple operators is that execution speed is highly prioritized and thus the number of evaluations is kept as low as possible. Further, as intra and inter route operators are used in sequence the marginal gain of improving a route containing a fixed set of nodes is limited as the nodes included in the route will vary. The *reverse operator* can improve a sequence of nodes if it contains chargers and the charging function is non-linear. The *2-Opt operator* will commonly be applied in a greedy version, which will be denoted as *greedy 2-Opt*. In *greedy 2-opt* the first arc is selected as starting point. The 2-Opt operation is then attempted at each following arc and upon improvement, the modified sequence is saved. If an improvement was found the same new arc at the previously selected index is used as a starting point. If no solution is found, the index of the starting arc is incremented. In summary, the greedy procedure will increase the starting index by one if no improvement was

found, always apply an improvement, and never try to apply the modification to arcs located earlier in the sequence.



(a) 2-0pt

(b) Reverse

(c) Insert node

(d) Swap nodes

Figure A.1: Intra-route operators.

Swap chargers

The swap charger operator tries to improve the choice of the charger. It is applied to each charger in the sequence and tries to either remove, move or substitute the charger. It uses the current index of the charger as a starting point for the current and other chargers. It then tries to move the charger one step either forward or backward in the sequence. If the move results in an improvement, the new solution is kept and the operator tries to move one step further in the same direction. This is repeated until the new sequence is worse than the previous. It then selects the solution with the lowest cost.

Swap vehicle

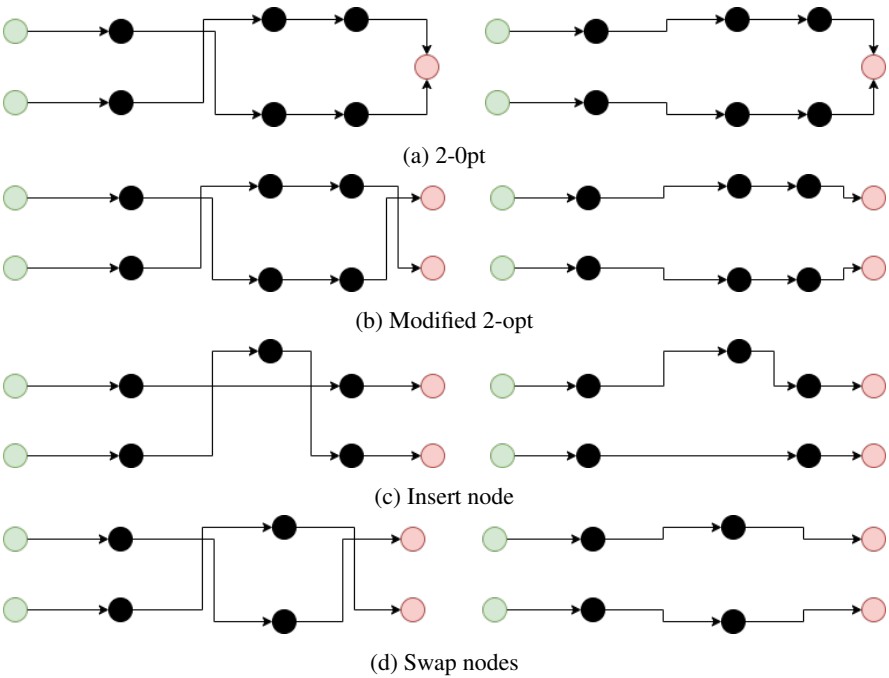
The swap vehicle operator attempts to change the vehicle used for the route. All available vehicles are assessed as a candidate. If a vehicle has multiple end depots, the cheapest end depot is always chosen.

Swap end depot

The operator tries to change the end depot to minimize the cost of the route.

A.2 Inter route operators

Inter route operators are applied to a pair of routes and modify the set of nodes included in each route. It is important to note that some extensions can prohibit the usage of certain operators. For example, if two routes have different end depots the regular 2-Opt operator is prohibited and must be modified as it alters the end depot. The *insert operator* can specifically target certain nodes, for example, costly nodes. It can also be targeted at nodes at certain indexes, for example, the last customer.



(a) 2-0pt

(b) Modified 2-opt

(c) Insert node

(d) Swap nodes

Figure A.2: Inter-route operators.

B

Inputs and parameters

In this appendix, the default input and parameter setting used in the framework is presented.

B.1 Cost functions and penalty functions

The cost functions for calculating the cost were set according to the values presented in Table B.1. The unit for measuring time is seconds and the unit for measuring the battery level is SoC.

When running the algorithm, some of the cost functions were modified to better fit as penalty functions. Further, for some functions, a relaxation coefficient, r, was introduced which takes on a value between 0 and 1. The penalty functions used are presented in Table B.2. If no penalty function is presented the original cost function was used. The only difference for the battery limit is an additional relaxation term. While the functions for the max travel time and for the time window might look different, they are fairly similar to the original cost functions with the exception that they have been converted into hourly costs, instead of seconds, and that an exponential term has been added. The exponential base in the max travel time increases by one each hour and the exponential base of the time window increases by one every

Table B.1: The cost functions used for the algorithm. Note that distances is only included for the Solomon benchmark.

Attribute	Threshold	Cost function	Parameters
Time	No	$c(x) = kx$	$k = 0.05$
Battery	Yes	$c(x) = (kx)^2$	$k = 1200$
Max Travel Time	Yes	$c(x) = kx$	$k = 0.055$
Quantity	Yes	$c(x) = kx$	$k = 300$
Time Window	Yes	$c(x) = kx$	$k = 0.167$
*Distance	No	$c(x) = kx$	$k = 1$

Table B.2: The penalty functions used for the algorithm.

Attribute	Threshold	Cost function	Parameters
Battery	Yes	$c(x) = (1-0.9r)^2(kx)^2$	$k = 1200$
Max Travel Time	Yes	$c(x) = (1-0.9r)k(\frac{x}{\alpha})^{1.2}$	$k = 200, \alpha = \frac{1}{3600}$
Time Window	Yes	$c(x) = (1-0.9r)k(\frac{x}{\alpha})^2$	$k = 300, \alpha = \frac{1}{1800}$

half hour. The reason for the added exponential term is to prevent all overtime to be added to the same vehicle.

B.2 Parameters settings

In Table B.3 the parameters used for the computational results are presented.

The *restart* and *permutation* attributes are used for the same purpose, namely changing the neighborhood of a provided solution. The difference is that *restart* is used for the initial change of neighborhood, i.e. the first time a permutation is applied. The *permutation* operator is used for all of the following restarts during the iteration. The *number of permutations* declares how many times the permutation phase should be applied. The *relaxation* parameter is the value of the coefficient r in the penalty functions and the *acceptance* is the factor that the cost of a new route is multiplied by (and if the cost after the multiplication is less than the previous cost the route is accepted). The *construction relaxation* is the relaxation factor used during the construction of the initial solution. The local search start and end relaxation are the initial respectively the last value of the relaxation coefficient. The value of the coefficient is during the local search phase linearly decreasing uniformly with the number of applied operators. The relaxation coefficient is not allowed to be negative and thus is set to zero if assigned a negative value. The parameter *local search improvement counter* is a boolean, when set to false it performs a fixed number of operators, while when set to true it counts operators applied without improvement. The *local search operators* decide how many fixed, or operators without improvements, that is performed before the next permutation. The neighborhood reduction function decides on what nodes to be considered as the next possible location. In this thesis, the function indicates that only the $f(N)$ closest customers will be allowed to be chosen as the next nodes. The variable κ, the function m and the variable θ are related to the vehicle fleet size and are described in section 4.2.

The inter route operators used per default were: swap nodes, 2-Opt, insert costly node and insert endpoint. Insert costly is an insert mechanism that prioritizes trying to move costly nodes while insert endpoints only try to allocate the first or last customer node in a sequence. In some cases operators were swapped to modified versions, for example the 2-opt operator for the multi-depot case and the swap and insert operators for the pick-up delivery extension. In Table B.4 the settings used are presented. The number of routes and nodes selected to apply the operator on is defined. In other words, if the number of routes is four and the number of nodes is three then three nodes are selected from four different routes and the operator is performed for each node. For a selected node the operator is attempted to be applied on all nodes present in the neighborhood of the node. If a probability in the Table is *constant* then all routes/nodes are assigned the same probability. If the probability is denoted with *cost* they are assigned a probability proportional to their cost.

For the intra route operators 2-opt, reverse, swap chargers, swap vehicles, and swap end depots were included by default. These were applied once at the start of each local search phase and upon a route being modified by an inter route operator.

Table B.3: The parameters used in the computational results for the implemented algorithm.

Parameter	Value
Number of permutations	2
Restart operators	3
Restart relaxation	0.2
Restart acceptance	0.9
Permutation operators	20
Permutation relaxation	0.1
Permutation acceptance	0.95
Construction relaxation	0.1
Local search start relaxation	0
Local search end relaxation	0
Local search operators	30
Local search improvement counter	True
Neighborhood reduction	$\mathrm{f(N)} = \begin{cases} N, & \text{if N} \leq 20 \\ \frac{N}{2}, & \text{if } N \leq 50 \\ \frac{N}{3}, & \text{if } N \leq 100 \\ \frac{N}{4}, & \text{if } N \leq 160 \\ \frac{N}{5} & \text{otherwise} \end{cases}$
κ	1
m	$m(\mathrm{x}) = \mathrm{x}$
θ	4

Table B.4: The parameters used for the inter route operators.

Operator	# routes	# nodes	Route probability	Node probability
Swap nodes	5	5	cost	constant
2-opt	3	5	cost	constant
Insert costly	4	4	cost	cost
Insert endpoints	all	all	constant	constant

Bibliography

Baños, R., J. Ortega, C. Gil, A. L. Márquez, and F. De Toro (2013). "A hybrid meta-heuristic for multi-objective vehicle routing problems with time windows". *Computers & industrial engineering* **65**:2, pp. 286–296.

Bertsimas, D., D. B. Brown, and C. Caramanis (2011). "Theory and applications of robust optimization". *SIAM review* **53**:3, pp. 464–501.

Blum, C. and A. Roli (2003). "Metaheuristics in combinatorial optimization: overview and conceptual comparison". *ACM computing surveys (CSUR)* **35**:3, pp. 268–308.

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

Cooke, K. L. and E. Halsey (1966). "The shortest route through a network with time-dependent internodal transit times". *Journal of mathematical analysis and applications* **14**:3, pp. 493–498.

Cordeau, J.-F., M. Gendreau, and G. Laporte (1997). "A tabu search heuristic for periodic and multi-depot vehicle routing problems". *Networks: An International Journal* **30**:2, pp. 105–119.

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

Desaulniers, G., J. Desrosiers, M. M. Solomon, F. Soumis, D. Villeneuve, et al. (1998). "A unified framework for deterministic time constrained vehicle routing and crew scheduling problems". In: *Fleet management and logistics*. Springer, pp. 57–93.

Dorigo, M., M. Birattari, and T. Stutzle (2006). "Ant colony optimization". *IEEE computational intelligence magazine* **1**:4, pp. 28–39.

Environmental Protection Agency, Sweden (2020). *Inrikes transporter, utsläpp av växthusgaser*. [Online; accessed 11-October-2021]. URL: `https://www.naturvardsverket.se/data-och-statistik/klimat/vaxthusgaser-utslapp-fran-inrikes-transporter`.

Fedex Corporation (2021). *Fedex fleet electrification*. [Online; accessed 18-October-2021]. URL: https://www.fedex.com/en-us/sustainability/electric-vehicles.html.

Goeke, D. and M. Schneider (2015). "Routing a mixed fleet of electric and conventional vehicles". *European Journal of Operational Research* **245**:1, pp. 81–99.

Gonçalves, F., S. R. Cardoso, S. Relvas, and A. Barbosa-Póvoa (2011). "Optimization of a distribution network using electric vehicles: a vrp problem". In: *Proceedings of the IO2011-15 Congresso da associação Portuguesa de Investigação Operacional, Coimbra, Portugal*, pp. 18–20.

Hashimoto, H., M. Yagiura, and T. Ibaraki (2008). "An iterated local search algorithm for the time-dependent vehicle routing problem with time windows". *Discrete Optimization* **5**:2, pp. 434–456.

Jozefowiez, N., F. Semet, and E.-G. Talbi (2008). "Multi-objective vehicle routing problems". *European journal of operational research* **189**:2, pp. 293–309.

Keskin, M., G. Laporte, and B. Çatay (2019). "Electric vehicle routing problem with time-dependent waiting times at recharging stations". *Computers & Operations Research* **107**, pp. 77–94.

Li, W., P. Stanula, P. Egede, S. Kara, and C. Herrmann (2016). "Determining the main factors influencing the energy consumption of electric vehicles in the usage phase". *Procedia Cirp* **48**, pp. 352–357.

Lin, C., K. L. Choy, G. T. Ho, S. H. Chung, and H. Lam (2014). "Survey of green vehicle routing problem: past and future trends". *Expert systems with applications* **41**:4, pp. 1118–1138.

Lin, J., W. Zhou, and O. Wolfson (2016). "Electric vehicle routing problem". *Transportation research procedia* **12**, pp. 508–521.

Lin, S. and B. W. Kernighan (1973). "An effective heuristic algorithm for the traveling-salesman problem". *Operations research* **21**:2, pp. 498–516.

Lourenço, H. R., O. C. Martin, and T. Stützle (2003). "Iterated local search". In: *Handbook of metaheuristics*. Springer, pp. 320–353.

Malandraki, C. and M. S. Daskin (1992). "Time dependent vehicle routing problems: formulations, properties and heuristic algorithms". *Transportation science* **26**:3, pp. 185–200.

Ministry of the Environment, Sweden (2021). *Sweden's climate policy framework*. [Online; accessed 11-October-2021]. URL: https://www.government.se/articles/2021/03/swedens-climate-policy-framework/.

Miranda, D. M., J. Branke, and S. V. Conceição (2018). "Algorithms for the multi-objective vehicle routing problem with hard time windows and stochastic travel time and service time". *Applied Soft Computing* **70**, pp. 66–79.

Mladenović, N. and P. Hansen (1997). "Variable neighborhood search". *Computers & operations research* **24**:11, pp. 1097–1100.

Montoya, A., C. Guéret, J. E. Mendoza, and J. G. Villegas (2017). "The electric vehicle routing problem with nonlinear charging function". *Transportation Research Part B: Methodological* **103**, pp. 87–110.

Na, B., Y. Jun, and B.-I. Kim (2011). "Some extensions to the sweep algorithm". *The International Journal of Advanced Manufacturing Technology* **56**:9, pp. 1057–1067.

Pillac, V., M. Gendreau, C. Guéret, and A. L. Medaglia (2013). "A review of dynamic vehicle routing problems". *European Journal of Operational Research* **225**:1, pp. 1–11.

Ritzinger, U., J. Puchinger, and R. F. Hartl (2016). "A survey on dynamic and stochastic vehicle routing problems". *International Journal of Production Research* **54**:1, pp. 215–231.

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

Schiffer, M. and G. Walther (2018). "An adaptive large neighborhood search for the location-routing problem with intra-route facilities". *Transportation Science* **52**:2, pp. 331–352.

Smith, A. E., D. W. Coit, T. Baeck, D. Fogel, and Z. Michalewicz (1997). "Penalty functions". *Handbook of evolutionary computation* **97**:1, p. C5.

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

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

Suthikarnnarunai, N. (2008). "A sweep algorithm for the mix fleet vehicle routing problem". In: *Proceedings of the International MultiConference of Engineers and Computer Scientists*. Vol. 2. Citeseer, pp. 19–21.

Toth, P. and D. Vigo (2002). *The vehicle routing problem*. SIAM.

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

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

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER'S THESIS

Date of issue
January 2022

Document Number
TFRT-6155

Author(s)
Johan Hellmark

Supervisor
Bo Lincoln, Iternio Planning AB, Sweden
Giacomo Como, Dept. of Automatic Control, Lund University, Sweden
Bo Bernhardsson, Dept. of Automatic Control, Lund University, Sweden (examiner)

Title and subtitle
A practical framework for the electric vehicle routing problem

Abstract

The routing of a delivery fleet is a classical optimization problem, known as the vehicle routing problem (VRP), which can heavily impact the quality of a logistic distribution process. Historically, the VRP formulation has mainly included internal combustion engine vehicles (ICEVs). However, due to their reduced environmental impact, the inclusion of electric vehicles (EVs) has become more popular. The inclusion requires accounting for a shorter driving range and limited infrastructure support. This thesis presents a framework for solving several practical extensions of the electric vehicle routing problem (E-VRP). Previously presented solvers usually target specific problem variants, optimize based on predetermined objectives, and display a general lack of discussion on their practical applicability. To counteract these shortcomings, the implemented framework allows for customizable objective functions, is capable of solving a wide range of practically relevant extensions, and provides an interface for investigating the properties of the proposed solutions. Examples of subjects treated are partial recharging, time-dependent variables, and dynamic settings. Solutions to real-world settings, modeled using the API of Iternio Planning AB, are demonstrated and the implemented solver shows promising results on a wide range of tested problem instances.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-77

Recipient's notes

Security classification

http://www.control.lth.se/publications/