UiT The Arctic University of Norway

Faculty of Engineering Science and Technology
Department of Computer Science and Computational Engineering

A Method for Determining Optimal Charging Infrastructures in Public Ferry Transportation Networks

Joachim Kristensen

DTE–3900, Master thesis in Applied Computer Science, Narvik, May 2023

Abstract

In this thesis, we establish the theoretical foundation for determining the minimum requirements for charging infrastructures in networks where electric ferries operate on a fixed schedule. The problem—named the Multiple-Trip Electric Charging Station Placement (MT-ECSP)—is formulated as a 0–1 Integer Program, and is subsequently proven to be $\mathcal{NP}$-hard.

We construct several instances of the MT-ECSP, and solve them using Google OR-Tools CP-SAT solver in a simple application. The results show that the MT-ECSP is solveable both for theoretical and practical purposes.

Acknowledgements

The author would like extend his deepest gratitude to the supervisors Rune Dalmo and Aleksander Pedersen for being instrumental not only during the course of this thesis project, but throughout the entirety of the master's studies. Thank you for always keeping your office doors open, challenging my ideas, and being beacons of inspiration.

Next, the author wishes to thank his colleague, Johanne Holst Klæboe, with whom countless hours of joy and utter frustration has been endured together. Without you, I would not been able to finish even the first term of this study programme.

The author also wishes to thank Hyke for providing the thesis project. It has proved to be the most challenging, yet rewarding feat to date.

Last, but not least, thank you mom and dad for your immense love, support and guideance during my academic ventures. Thank you for encouraging me to pursue a degree, and motivating me to see it through.

Contents

List of Figures

List of Tables

1 Introduction

This master thesis is a research project centering around the theoretical groundwork for solving a practical problem concerning the optimal placement of charging stations in public transit networks. This work conducted in this thesis aims to showcase computer science as a tool for problem solving, by providing such a foundation and demonstrating its applications.

In this chapter, we cover relevant theory, state of the art papers and methodology, and the objectives of the research project.

The remainder of the thesis is organized as follows. In Chapter 2, the tools used for meeting the proposed objectives are presented. Chapters 3 to 5 are all chapters presenting theoretical and practical results. In chapter 3, we present a mathematical formulation of the problem, alongside model assumptions, nomenclature, and lastly prove its complexity. Chapter 4 show the nomenclature and model in a practical setting, where a program is devised in an attempt to solve the problem. Lastly, chapter 5 evaluates the performance of the program.

1.1 Background

Over half of the global population resides in cities, which contribute significantly to greenhouse gas emissions [1]. In 2014, the transportation sector was the world's second largest GHG emitter, accounting for roughly 23% of global

emissions and 21% of emissions within the European Union [2, 3, 4]. Moreover, this sector is linked to air pollution and traffic congestion issues.

A 2007 study by Schrank et al. revealed that in 2005, the average annual travel delay for rush hour trips in the United States was 38 hours, based on data from 437 urban areas [5]. According to INRIX Roadway Analytics, this figure has increased significantly since then, particularly in major urban centers. For instance, in 2022, London experienced the highest average annual traffic delay at 156 hours [6].

A response to the obligation of countries contributing to climate mitigation has sparked a growing interest in sustainable solutions. In his 2018 paper, Fulton [7] coined three revolutions in urban transportation as vehicle electrfication, automation and shared mobility as emerging trends in future mobility. In recent years, the Norwegian government has allocated millions of NOK in funding research and innovation in new climate-friendly technology [8].

From Norway's Climate Aciton Plan for 2021-2023, "Climate initiatives in Norway will be designed to bring about domestic emission reductions and to develop technology that can also be used internationally. Norway will play a part in creating climate solutions for the future." [9, p. 13].

The topography of Norway, often characterized by interconnected cities and waterways, including Drammen, Oslo, Fredrikstad, Trondheim, and Tromsø, presents opportunities for developing climate-friendly urban transport solutions. In 2015, the Norwegian *MF Ampere* was the world's first all-electric car ferry put into service. Tannum and Ulvensøen[10] gave a state of the art review of urban mobility projects in Norway centered around autonomous short-distance ferries in 2019. In addition, numerous publications[11, 12, 13, 14, 15] have emerged both in Norway and internationally, highlighting the growing interest, challanges, and advancements in this sector.

In the pool of newly sprung innovations in Norway emerges an Oslo based company developing electric, autonomous ferries (EAFs) for public transport in urban areas; Hydrolift Smart City Ferries, or Hyke [16].

This thesis is the formulated in joint partnership with Hyke, and concerns itself with the strategic placement of charging jetties in the area a ferry fleet is operating in. Hyke wishes to gain insight into the minimum requirements for charging jetties; how many jetties on a given route need to be charging jetties to ensure that the ferries can continously operate without having to charge for a prolonged amount of time.

In the following section, this thesis will explore various theoretical concepts,

such as complexity theory and optimization problems, which are essential for understanding the nature of the problem under investigation. These concepts will be applied to the practical problem, ultimately leading to a proof-of-concept implementation.

1.2 Theory

The field of computer science is built upon the core principle of problem solving. From its inception, computer science has aimed to develop methods and tools to address a wide range of practical and theoretical challenges. This thesis explores one such practical problem and demonstrates how computer science techniques can be employed to provide a comprehensive and efficient solution. Before delving into the specific problem, it is essential to understand the general process of problem solving in computer science.

The first step in a problem solving process is to transform a real world problem into a well defined problem that can be systematically addressed by computer science. This involves identifying the essential components and characteristics of the problem, such as its input, desired output, any constraints or conditions that must be met, and in many cases [17, 18, 13, 19] a mathematical formulation/model. A clear and precise problem statement is crucial, as it serves as the foundation upon which the rest of the problem solving process is built.

Once a problem is well defined, it can be classified based on its inherent characteristics. This classification helps determine which programming paradigm(s), algorithm(s), and technique(s) are most appropriate for solving the problem at hand. An understanding of the problem's classification is instrumental in guiding the subsequent steps of the problem solving process[20].

Different problems may require different solving strategies, and the efficiency of a solution often depends on the appropriateness of the chosen algorithm(s) [19, p. 17] [21]. This selection process involves considering factors such as time complexity, space complexity, and ease of implementation, as well as potential trade-offs between these factors.

1.2.1 Complexity Theory and NP-Complete Problems

Complexity theory is a branch of computer science that aims to understand the resources needed to execute computational problems efficiently. These resources can include time, memory, or any other factors that influence the performance of an algorithm. By classifying problems based on their inherent

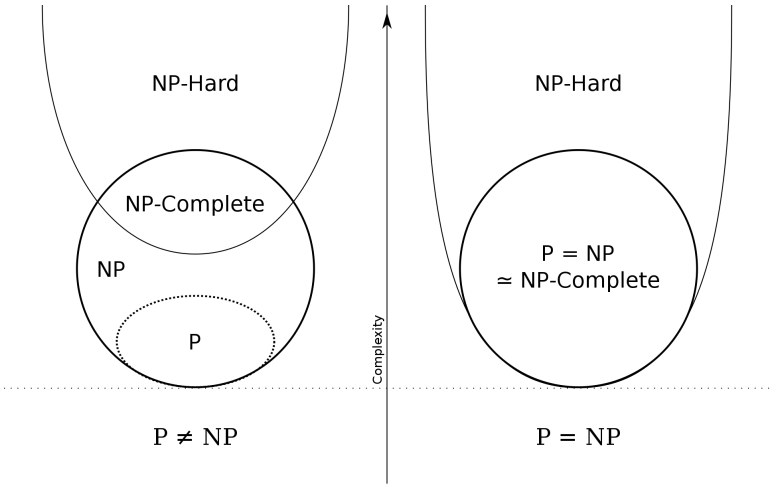


Figure 1.1: Euler diagram for P, NP, NP-complete, and NP-hard set of problems. The left side is valid under the assumption that P ≠ NP, while the right side is valid under the assumption that P = NP [23].

difficulty, complexity theory provides insights into the limitations of computational power and the solvability of various problems.

Two primary classes of problems are central to complexity theory: $\mathcal{P}$ and $\mathcal{NP}$. Problems in the $\mathcal{P}$ class are considered 'easy' to solve, as algorithms exist that can solve them in polynomial time—the time taken to find a solution is proportional to a polynomial function of the input size. In contrast, $\mathcal{NP}$ (nondeterministic polynomial time) problems are those for which a solution can be verified in polynomial time, but discovering the solution might not follow the same time constraint.

A subset of $\mathcal{NP}$ problems, known as $\mathcal{NP}$-complete problems are considered the most challenging within the $\mathcal{NP}$ class, and have a unique property: if an algorithm could efficiently solve any $\mathcal{NP}$-complete problem, it could be adapted to solve all other problems in the $\mathcal{NP}$ class with the same efficiency. The concept of $\mathcal{NP}$-completeness was first introduced by Stephen Cook in his seminal paper 'The Complexity of Theorem-Proving Procedures' [22], where he proved that the Boolean Satisfiability Problem (SAT) is $\mathcal{NP}$-complete.

Despite extensive research, an efficient algorithm for any $\mathcal{NP}$-complete problem has not yet been discovered. As a result, the question of whether $\mathcal{P}$ equals $\mathcal{NP}$ remains one of the most significant open questions in the field, illustrated in fig. 1.1. If $\mathcal{P}$ were proven to be equal to $\mathcal{NP}$, it would mean that every problem with a solution verifiable in polynomial time could also be solved in polynomial time.

Building upon Cook's work, Richard Karp's influential paper 'Reducibility Among Combinatorial Problems' demonstrated the $\mathcal{NP}$-completeness of 21 combinatorial problems by introducing the concept of polynomial-time reducibility [24]. This technique allowed Karp to show that if any one of the 21 problems he studied could be solved efficiently, then all of them could be solved efficiently. The polynomial-time reducibility showed $\mathcal{NP}$-complete problems are closely related in terms of their complexity, and that they shared the same level of inherent difficulty.

The significance of $\mathcal{NP}$-completeness was further emphasized in the classic book 'Computers and Intractability: A Guide to the Theory of NP-Completeness' by Garey and Johnson [20]. This book provided a comprehensive introduction to the topic, cataloging many more $\mathcal{NP}$-complete problems and offering insights into the study of computational complexity. The word *intractibility* in this context refers to problems for which no efficient (polynomial-time) algorithm is known or believed to exist. For NP-complete problems, intractibility is rooted in this fact.

It is, however, essential to recognize that their intractability is a reflection of our current state of knowledge rather than an inherent property of the problems themselves. Notably, 'intractable' does not imply that a problem is unsolvable; rather, it suggests that the problem is challenging to solve efficiently, especially for large input sizes.

As mentioned by Wayne[25], solving any NP-complete problem involves sacrificing one of three desired features.

i. Solve arbitary instances of the problem.

ii. Solve problem to optimality.

iii. Solve problem in polynomial time.

1.2.2 0–1 Integer Programming

One of the problems Karp proved to be $\mathcal{NP}$-complete was 0–1 Integer Programming (IP). However, the theoretical foundation of 0–1 IP dates back to the early works of George Dantzig, who laid the groundwork for Linear Programming (LP) [26], and later IP.

A *Linear Program* (LP) is a classical optimization problem consisting of a set of continous variables and a set of linear constraints (equalities or inequalities), where the goal is to optimize (minimize/maximize) a function subject to the

constraints [27]. The LP can be expressed in its *canonical form* as

$$\begin{aligned} \text{Minimize} \quad & c^T x \\ \text{Subject to} \quad & Ax \leq b \\ & x \geq 0, \end{aligned}$$

where x is a column vector of n variables, c is a column vector of objective coefficients, A is a matrix of constraint coefficients, and b is a column vector of constraint bounds. The objective function $c^T x$ is linear with respect to the decision variables, and the constraints are linear inequalities.

If the variables in the LP is restricted to be integers, the problem is referred to as an *Integer Program* (IP). If the variables can take both continous or integer values, the problem is referred to as a *Mixed Integer Linear Program* (MILP), or *Mixed Integer (Linear) Program* (MIP).

0–1 IP is a special case of IP, in which the decision variables stored in x are binary, i.e., restricted to values of 0 or 1. The binary nature of the decision variables enables the modeling of complex combinatorial problems with inherent yes-or-no decisions, such as[11, 18].

A 0–1 IP problem can be formulated as follows:

$$\begin{aligned} \text{Minimize} \quad & c^T x \\ \text{Subject to} \quad & Ax \leq b \\ & x \in \{0,1\}^n, \end{aligned}$$

where $\{0, 1\}$ denotes the set of binary values.

Due to the expressiveness of IPs, many other classical problems such as *The Travelling Salesman Problem* (TSP) ans *The Knapsack Problem* (KP) are formulated as IPs[28].

1.2.3 Optimization Problems

Descision problems and optimization problems are two categories of computational problems that differ in their objectives and the nature of their solutions. A decision problem is a computational problem that asks a yes-or-no question about the input. The goal is to determine whether the input satisfies a specific property or condition, and so, decision problems can be formalized as a set of inputs for which the answer is ‘yes’ and another set for which the answer is ‘no’[20].

Optimization problems, in contrast, involve finding the best solution among a set of possible solutions according to some objective function or criterion. The goal is to either maximize or minimize an objective, depending on the problem at hand.

In relation to complexity theory, it is the descision version of a problem that refers to its computational complexity[29], whereas the associated optimization problem can be solved by answering the descision problem a polynomial number of times[28]. Generally, if the decision version of a problem is $\mathcal{NP}$-complete, then its associated optimization version is $\mathcal{NP}$-hard[28, 30]; that is, at least as hard as the hardest problems in $\mathcal{NP}$.

In this thesis, we are investigating an optimization problem which will be formally introduced in later sections. Optimization problems are ubiquitous in many fields such as mathematics, economics, engineering, and computer science, to name a few. They often involve finding an optimal solution to a problem that is constrained by limited resources, conflicting objectives, or other restrictions. *Discrete optimization* (DO), in particular, focuses on problems where the set of possible solutions is discrete or finite, rather than continuous. 0–1 IP describes such an optimization problem, as the binary nature of the decision variables denotes a yes-or-no decision to be made; *'should I go left or right?'*, *'invest in this stock or the other?'*, or *'install a charger at this location, and if so, which charger type?'*.

1.2.4 A Brief Overview of Constraint Programming

A paradigm used when solving optimization problems is constraint programming (CP). It is an approach that allows for the expression of combinatorial problems using constraints, which are relations that must be satisfied by the problems solution; that is, an assignment to the decision variables that satisfies the constraints[31]. By defining the problem in terms of constraints, *solvers* can apply specialized search techniques to efficiently explore the solution space, pruning branches that cannot lead to a feasible or optimal solution.

In [32], the CP paradigm is described as a two-phases approach; *modeling* the problem, and finding a solution to it. The modeling process involves introducing the variables of the problem, their domain(s), and defining the constraints which governs them. The solution phase is conducted by built-in solvers which interprets the model and (hopefully) finds an optimal solution. This creates a modularity to the problem solving process, allowing developers to focus on accurately representing the problem without worrying about the underlying search algorithms or optimization techniques.

The mid 80's saw the rise of the programming language *CHIP*, and towards the late 80's *Prolog III*, and the early 90's *CLP(R)*. These where the first languages defined as Constraint Logic Programming (CLP) languages, which emerged as a generalization of logic programming. Noteable features included arithmetic constraints, linear arithmetic over rational numbers, and constraints over sequences. Methods at the time for addressing constraint search problems involved developing specialized programs in procedural languages. This process demanded significant effort in program development, and the resulting programs frequently proved difficult to maintain, modify, and expand[33]. CHIP was an attempt to overcome these difficulties, and was the first language to offer finite domain constraints in its toolbox, and enabled a more declarative and flexible approach to constraint solving techniques.

Later, the software company *ILOG* had its advent. Unlike CLPs - which had programmable search - ILOG provided an *Optimization Modeling Language* (OML), enabling the developer to only model the problem, and letting the underlying ILOG Solver solve it[34]. This further emphasized the two-phase approach in the CP paradigm, and led to a alternative problem-solving process where developers could focus on problem representation and leave the intricacies of the search algorithms and optimization techniques to the underlying solvers. For more information regarding CP, CLP, and OML, we refer the reader to [31].

1.3 State of the Art

In this section, we discuss the results of our literature study. It is mainly divided into two categories, with some overlap. In section 1.3.1, we present a brief overview concering the use of optimization tools in literature. Section 1.3.2 concerns itself with literature on charging infrastructure for electric vehicles. Section 1.3.3 highlights a paper by Wang et al. [18], where they present a method minimizing the cost in charging infrastructures for electric buses.

1.3.1 Solving Optimization Problems

In section 1.2.4, we discussed the use of solvers in relation to CP. Over the years, a variety of both commercial and open-source solvers have become available, and their use has been prevalent in the literature. Many papers reviewed during the literature study of this thesis, such as[12, 35, 36, 37, 17, 38, 39], have utilized solvers as a tool to achieve their respective objectives.

From these papers, we identify two categories in which solvers have been em-

ployed; (1), utilize them to solve a model, or (2), propose algorithms which are compared with the solution of the solver (performance and quality of solution).

Lai and Lo[17], and Li et al.[37] implemented heuristic algorithms to solve a *Network Design Problem (NDP)*, and a routing problem with opportunity charging, respecitvely. Their results were subsequently compared with those obtained using the *IBM ILOG CPLEX Optimizer* (which we will refer to as *CPLEX* from here on). Schiffer and Walther[35], and Havre et al.[12] used the *Gurobi* optimization tool for solving routing problems. Other authors, such as Paparella et al. [39] used both Gurobi and *YALMIP(Yet Another LMI Parser)*[40] in a joint optimization model.

The use of solvers and optimization tools in relation to this thesis will be further discussed in chapter 2.

1.3.2 Charging Infrastructure

Literature considering charging station (CS) locality for electric maritime transport is relatively scarse. To the best of the authors knowledge, no literature exsits on CS locality considering EAFs for public transport in urban areas. In a recent master's thesis by Driessen [14], a CS placement method for inland shipping in the Netherlands was developed using a flow-refueling model; originally proposed by Kuby and Lim [38]. Driessen also mentions an absense of scientific publications in this area, as literature usually consider battery swapping location for ships, and the technical and financial feasibility of charging stations. On problems related to planning EV charging infrastructure, Driessen refers to a recent review of modeling options by Metais et al. [41].

In their review, 63 articles concerining infrastructure optimization were considered. These articles considered an optimization goal alongside a general problem. This could be maximizing the number of EVs that can be charged at a station, minimzing time spent waiting at a CS, or minimizing infrastructure costs; finding a configuration that allows to have as few stations as possible whilst covering the demand for charging. For the latter, location and sizing is a general problem.

Metais et al. defined three main categories for optimization models; node-based, path-based, and tour-based, where the main difference is the way in which they represent the location of facilities (charging stations).

A node-based location model represents the location of a facility as a node in a network. The model aims to identify a subset of nodes in the network

where facilities should be located to optimize a given objective function, such as guaranteeing facilities appear within a certain predetermined distance.

A path-based location model represents the location of a facility as a set of connected edges in a network, where the demand along an edge is represented as a flow running through the edge. Followingly, some edges have a higher flow than others, and so, the model aims to identify a subset of paths or routes in the network where facilities should be located in order to cover a maximum of flows. Contrary to the node-based model, facilities are placed alongside an edge. Driessen [14], utilized a path-based optimization model.

A tour-based location model represents the location of a facility as a complete tour or cycle in the network. The model aims to identify a subset of tours or cycles that visit a set of demand points and include the location of a facility in the tour. Facilities can be placed on any point along the tour, including both nodes and edges. This was also illustrated by Driessen[14, p. 1], as the number of CS in a feasable round trip could heavily depend on their location in the network.

From the 63 articles, only two considered buses or public transport as a use case, and only one[18] of the two minimized the infrastructure cost as an optimization goal, which we will discuss in the next section.

1.3.3 The Electric Charging Station Placement Problem

Wang et al. [18] mentions several contributions to the placement of EV charging stations. However a common property among them is the assumption that the vehicles can freely traverse the area to find an appropriate charging station. They proceed to formulate two problems concerning the charging infrastructre for electric buses, namely the *Electric Charging Station Placement (ECSP)* problem, and the *ECSP_LB* (limited battery) problem. These are both problems referring to a tour-based location model, where the candidate sites (where to place the charging stations) are limited to nodes.

A key difference between Wang et al. and the other papers included in [41] is the assumption that vehicles operate under a fixed shedule, and so, uncertainties regarding SoC and traversal patterns is quite low. Charging infrastucture for buses are considered easier to design due to this fact[41]. Two approches are briefly introduced: (1) If the vehicles can operate without charging in the entirety of an operational period, they can simply be charged at a depot. (2) Utilize fast charging stations at selected stops so that every vehicle can complete their tours. A similar argument was made by Masliakova[42], where bus routes and partial charging strategies was designed using Ant Colony Optimization

techniques with Genetic Algorthims.

Both the ECSP and ECSP_LB problems were designed such that the buses traversing their respective routes should have sufficient SoC to complete their trip. Their models considered the energy consumption for each stop sequence in a route, and then the total energy consumption for the forwarding trip and the returning trip, respectively. The recharging capabilities provided by the charging infrastructure had to be sufficiently high, such that the buses were always able to reach the next stop in the route.

Both the ECSP and ECSP_LB problems were proved to be NP-hard by reduction from the *Vertex Cover* (VC) problem. The ECSP was formulated as a LP, where heuristic algorithms were developed to approximate a solution. The ECSP_ LB was formulated as an ILP, where the general case considers multiple routes. In the special case considering a single route, a backtracking algorithm implemented to find an optimal solution. This algorithm could also be extended to the general case (a multiple backtracking algorithm), but at a higher complexity.

To the best of the authors knowledge, this paper is considered state of the art in the case where the vehicles operates on a fixed schedule.

1.4 Contribution to Literautre

The ECSP and ECSP_LB problems indeed introduce the concept of different charging rates, but there is room to elaborate on certain aspects. For instance, the nature of the chargers used could be more explicitly stated— whether it is a single type with varying charging intervals or involves multiple charger types with a fixed or varying interval. Additionally, the charging state of the buses could be further clarified— whether they are partially charged or charge to full.

The charging strategy currently revolves around the premise that the energy charged at a stop cannot exceed a constant, B, representing the bus's battery size. Further diversification of charging strategies could potentially enhance the practical applicability of the models. Finally, the designed use case of the ECSP and ECSP_LB problems could be expanded upon— whether they are meant for multiple route traversals or just a single one. We recognize this is an important distinction, as:

1. The charging infrastructure desgined for a single traversal has to make sure that the vehicle can reach the next stop in its route, once. This

implies when reaching the final stop in a route, the vehicles SoC could—in theory—be close to 0%, or some other threshold value.

2. For multiple trips where the charging interval is fixed, the charging infrastructure has to make sure that the vehicle can reach the next stop in the route multiple times. In other words, the recharging capabilities must be at least as high as the energy consumed, or the SoC loss between every traversal of the route must be small enough to support multiple trips, as discussed in [42].

3. If the charging interval is not fixed, then the vehicle can charge to full at any leg of its route, or choose an appropriate charging interval based on the current energy needs.

We will draw inspiration from Wang et al., and propose a modified problem formulation. Specifically, we will consider the case where the vehicles traverse their route indefinetly, with multiple chargers types available, and a fixed charging interval.

1.5 Objectives

The objectives of this thesis are both scientific and practical. The primary objective is to determine the computational complexity of identifying the minimum requirement for charging stations in a network of electric ferries. The minimum requirement refers to the same objective of Wang et al., and is to minimize the infrastructre cost, while providing sufficient charging capabilities.

Determining the computational complexity involves framing the problem in a manner that allows it to be related to a known problem and attempting to prove its complexity through reduction. This will subsequently allow for solving strategies to emerge.

The solving strategies encompasses the practical objective of this thesis. We will consider optimization tools which can be used to solve our model of the problem. Exploiting the model, we will demonstrate its feasibility by implementing an application, and conduct several experiments as a proof of concept. This is meant to compliment the primary objective, as well as showing that the problem has a practical applicability.

We hereby refer to the problem as the Multiple-Trip Electric Charging Station Placement (MT-ECSP) problem.

2 Tools

In this chapter, we present the tools used in this thesis. Since the objectives does not concern implementations/improvements upon already existing algorithms, but rather provide a proof of concept by means of programming, we will utilize already existing tools for solving the MT-ECSP. To meet this end, we will investigate optimization tools and programming languages suitable for a time limited research project, like this thesis.

The hardware used is a laptop running Microsoft Windows 11 Education (v. 10.0.22621), equipped with 32GB RAM, and an Intel Core i7–9750H CPU @ 2.60GHz.

The first requirement for any proprietary tool is compatibility with Windows 11, and preferrably integration with an IDE. Moreover, learning a programming language unknown to the author could severely slow down—or potentially hinder the progress of—the development phase. Finally, licensing constraints would have a greater negative impact, as it either involves a budget for using the tool, or is time limited by a trial period.

2.1 Programming Langugages

In choosing a programming language for the development phase, we prioritize languages in which the author has the most expertise and knowledge,

specifically C++ and Python. Furthermore, it is essential to note that the optimization tool itself carries out the heavy-duty computation in the experiment. The selected programming language will be employed for preparing and manipulating data and act as a bridge with the chosen optimization tool.

C++ is an open soruce, high level general-purpose language. It is a *compiled* langugage; the source code is passed to a compiler which generates machine code executeable by the CPU prior to runtime. Some of C++'s core strengths include low-level memory manipulation and template-based programming, granting developers granular control over their applications' performance. This makes C++ a powerful language to write performance-critical applications, examples being in the fields of game development, embedded systems, and optimization software.

Python is an open source, high level and general-purpose programming language. Unlike C++, Python is not a compiled language, but *interepreted*; source code is being translated to machine code line by line during runtime. As a result, interpreted languages tend to be slower in execution than compiled languages. However, Python is widely known for its modularity (in terms of libraries), and relatively simple and consistent syntax. This makes Python suitable for rapid prototyping, which is imperative for the development phase.

2.2 Optimization Tools

In this section, we discuss four optimization tools that were considered for solving the MT-ECSP; Gurobi, CPLEX, Google OR-Tools, and MiniZinc.

Gurobi [43] is an optimization solver that offer solvers for LP, MILP, Quadratic Programming (QP), Mixed-Integer Quadratic Programming (MIQP), Quadratically Constrained Programming (QCP), and Mixed-integer Quadratically Constrained Programming (MIQCP). It is widely used in industries such as finance, energy, logistics, and manufacturing for solving complex optimization problems. Gurobi supports Windows, Linux, and Mac OS X, and offers an API for C, C++, C#, Java, Python, and MATLAB.

Similar to Gurobi, the CPLEX optimizer is compatible with all three operating systems, and provides an API for the same programming languages.

MiniZinc[44] is a free, open source constraint modeling language which lets the developer model constraint satisfaction—and optimization problems. It was developed to create a standard modelling language for CP problems, as experimenting with different solvers required the developer to learn different,

incompatible modelling langues; making them hard to compare[45]. MiniZinc was then introduced as a solver independent modelling language which could support multiple solvers such as Gurobi or CPLEX. The distribution of MiniZinc is bundled with the MiniZinc IDE, compiler, and several pre-configured solvers. In addition, the MiniZinc distrubution offer interfaces for Python and JavaScript.

Google OR-Tools[46] is a free, open-source software suite for optimization. It provides a rich API featuring specialized solvers, models, and algorithms for solving problems such as linear and mixed-integer programming, constraint programming, vehicle routing, and graph algorithms. The Google OR-Tools' native library is implemented in C++, however, wrappers for the programming languages Python, C#, and Java are available.

2.3 Comparison

Table 2.1 compares the different optimization tools considered for the development phase. All four optimization tools offer a community, support, documentation, and example implementations. Notably, both MiniZinc and Google OR-Tools provide dedicated Google discussion groups, making it convenient for developers to ask questions about the respective tools by simply registering with a Google email account. Unfortunately, both Gurobi and CPLEX require a license for usage, immediately disqualifying them as potential tools for the development phase.

Both the MiniZinc distribution and Google OR-Tools comes seemingly ready out-of-the-box after some preliminary installation steps. It is important to note that, while using the MiniZinc API with a programming language, one cannot directly define a model using the programming languages' syntax. Instead, the model must be created using the MiniZinc language and subsequently loaded into the respective languages' API. Google OR-Tools, however, lets one directly define a model using the chosen programming language's syntax. Ultimately, this led Google OR-Tools to be selected as the optimization tool for the development phase.

As previously mentioned, both C++ and Python are compatible with Google OR-Tools. Taken into consideration that the programming language should only be used for pre-processing and interface with the underlying API, we consider Python to be the most suitable language for the development phase.

Feature	CPLEX	Gurobi	MiniZinc	Google OR-Tools
Python	yes	yes	partially	yes
C++	yes	yes	no	yes
Free	no	no	yes	yes
IDE Integration	yes	yes	yes	yes
User Community	yes	yes	yes	yes
Support	yes	yes	yes	yes

Table 2.1: A brief comparison of different optimization tools, with respect to some selected features.

3

Problem Formulation

In this chapter, we present the problem formulation of the MT-ECSP. First, the notations, characteristics, and a preliminary analysis of the solution space is presented. Then, we show that the MT-ECSP is $\mathcal{NP}$–hard by reduction from 0–1 IP.

3.1 Characteristics

This section presents the characteristics of the MT-ECSP. We are aware that uncertainties exist in several aspects of the problem, and so, deterministic conditions are applied for simplification.

- **All routes are bi-directional.** The routes of the transit service are known, and are bi-directional, i.e, the ferries travels to their end destination, and then retraces their path back to the starting point, covering each stop in reverse order. Moreover, we assume every route to be designed in such a way that the ferries are able to traverse once without running out of energy. This way, we neglect the battery constraint.

- **The energy consumption is known.** The energy consumption for each route is assumed to represent the worst-case scenario, which in our model implies that the ferries are operating while being fully loaded, driving at a constant speed. In reality, actual energy usage may be influenced by

various factors such as ascending or descending inclines, driving with or against the current, and harnessing solar energy for recharging purposes.

- **The charging duration is fixed.** Charging occurrs whilst loading/unloading passengers at a stop equipped with a charger. Since the ferries are assumed to be operating at full capacity, we also assume that the time spent docked is the same for every stop, and so, we assume a partial charging strategy.

- **The ferries are homogenous.** We assume the ferries to be homogenous; they use the same amount of energy (if traversing the same route), and have the same capacity.

- **The charging effect is consistent.** Conditions that affect the amount of power recharged, such as battery temperature, is disregarded.

- **Any stop can be equipped with a charger.** A stop can only be equipped with one charger, and each route must have at least one charger installed. This implies that a vehicle cannot travel to a nearby stop outside its route to charge.

- **Grid availability is disregarded.** We assume that the cost of installing a charger is equal to the cost of purchasing it, and is not affected by grid availability.

- **Vehicles are operational at all times.** The charging infrastructure is designed in such a way that the vehicles are able to continously operate their assigned route without having to stop for a prolonged amount of time, i.e., the charging infrastructure must provide ample charging capabilities to compensate for energy loss associated with a route traversal.

- **Charger types are unlimited.** We assume there can be an umlimited amount of charger types used in the charging infrastructure.

3.2 Notation

The following notation is used to describe the MT-ECSP:

- A set $\mathcal{S}$ denoting all unique stops in the network, where $\mathcal{S} = \{s_0, \ldots, s_n\}$, where $n > 1$. Each stop s_i is represented as a tuple consisting of a value and a cost. The functions $v(s_i)$ and $c(s_i)$ can be utilized to obtain the value and cost of a stop, respectively.

- A set $\mathcal{R}$ denoting all routes in the network, where $\mathcal{R} = \{r_0, \ldots, r_p\}$, where $p > 0$. Each route r_w is represented by a sequence of stops, such that $\mathcal{R} \subseteq \mathcal{S}$ and $r_w \subset \mathcal{S}$.

- A set $\mathcal{Q} = \{Q_0, \ldots, Q_p\}$ denoting the multiplicity of stops in a route. Each element $Q_w = \{q_{w,0}, \ldots, q_{w,i}\}$ represents the occurrence of s_i in route r_w.

- A set $\mathcal{K}$ denoting the energy consumed for traversing each route in the network, where $\mathcal{K} = \{k_0, \ldots, k_p\}$. Each element k_w corresponds to the energy consumption of route r_w.

- A set $\mathcal{V}$ denoting the available charger types, where $\mathcal{V} = \{v_0, \ldots, v_m\}$, where $m > 0$. Each element v_j denotes the charging power (kW) delivered when charging for a fixed time interval.

- A set C denoting the costs associated with purchasing each charger type, where $C = \{c_0, \ldots, c_m\}$. Each element c_j corresponds to the cost of purchasing charger v_j.

The set $\mathcal{Q}$ serves to emphasize the value of installing a charger at a specific stop s_i in the network. For example, consider an instance in which stop s_i is visited twice during the traversal of route r_w. In this case, the vehicle would charge at the stop both during the forwarding trip and the returning trip. Consequently, the charging capabilitiy of v_j on s_i is effectively doubled, while the cost c_j is unchanged. It is important to note that we assume the vehicle to complete the docking and charging procedures before departing from the first stop in route r_p. As a result, the first stop in r_w is not included in Q_w.

3.3 Instance

The goal of the MT-ECSP is to find the assignment of v_j to s_i such that

$$\sum_{s \in r_w} v(s) \geq k_w \quad \forall w = 0, 1, \ldots, p \tag{3.1}$$

and the total cost c is minimized. We denote the solutions violating and adhering to (3.1) as *infeasible* and *feasible* solutions, respecitvely. The solution(s) with the lowest cost and while adhering to (3.1), are refered to as *optimal*. All the feasible solutions including the optimal denotes a feasible region.

An instance of the MT-ECSP instance is given by defining the sets $\mathcal{R}, \mathcal{K}, \mathcal{V}$ and C. During a preprocessing step, we compute the reminding sets $\mathcal{S}$ and

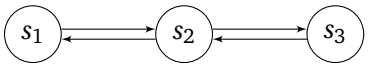


Figure 3.1: A route, r, with three stops, s_0, s_1, s_2

Q.

We illustrate a simple instasnce with only one route and three charger types, based in the network shown in fig. 3.1. Then, we get:

- $\mathcal{R} = \{r_0\} = \{s_0, s_1, s_2, s_1, s_0\}$
- $\mathcal{S} = \{s_0, s_1, s_2\}$
- $\mathcal{Q} = \{Q_0\} = \{1, 2, 1\}$, corresponding to s_0, s_1 and s_2, respectively.
- $\mathcal{K} = \{k_0\} = 30$.
- $\mathcal{V} = \{v_0, v_1, v_2\} = \{5, 15, 50\}$.
- $C = \{c_0, c_1, c_2\} = \{10, 100, 1000\}$,

where the values of the elements of $\mathcal{K}$, $\mathcal{V}$, and C are arbitrary numbers.

Given that the energy consumption $k_0 = 30$ and the multiplicity of stop s_1 is $Q_{0,1} = 2$, the optimal solution to this instance has a cost of 100, and corresponds to assigning charger type v_1 to stop s_1, as $2v_1 \geq k_0$. No other feasible solutions are able to meet this end.

While the solution to this instance can be found by obeservation alone, there are in fact 63 possible solutions. In the next section, we show that the solution space of any instance grows exponentially with respect to the size of $\mathcal{S}$ and $\mathcal{V}$.

3.4 Charger Configurations

Given a network of any size, n, a maximum of n chargers can be installed. We express this as

$$\sum_{e=1}^{n} \binom{n}{e}, \tag{3.2}$$

e	$\binom{n}{e}$	Combinations	Options
1	$\binom{3}{1}$	$\{s_1\}$ or $\{s_2\}$ or $\{s_3\}$	3
2	$\binom{3}{2}$	$\{s_1, s_2\}$ or $\{s_1, s_3\}$ or $\{s_2, s_3\}$	3
3	$\binom{3}{3}$	$\{s_1, s_2, s_3\}$	1

Table 3.1: Options for selecting e candidate sites from a route total of 3 stops.

which denotes the number of ways to equip n stops with e charging stations. Then, for every instance of e, there are m available charger types to choose from. We extend (3.2), and get

$$\sum_{e=1}^{n} \binom{n}{e} m^e. \tag{3.3}$$

Now, we continue with the example used in the previous section; a single route with three stops, such that $m = n = 3$.

$\binom{n}{e}$ implies the number of ways to select candidate sites for different values of e, shown in table 3.1.

Ultimately, (3.3) yields a total of $3 * 3^1 + 3 * 3^2 + 1 * 3^3 = 63$ possible charger configurations for this instance.

By experimenting with different sizes of $n = \{1, 2, 3, 4, 5, 6\}$, respectively, we observe (3.3) defines the sequence $3, 15, 63, 255, 1023, 4095$. We express it as a function of n

$$f(n) = 4^n - 1, \tag{3.4}$$

where -1 denotes at least one charger has to be installed.

Recall $m = 3$ in the example. Since m can be any positive integer in the general case, we rewrite (3.4) as a function of n and m such that

$$f(n, m) = (m + 1)^n - 1, \tag{3.5}$$

where $(m + 1)$ denotes the number of choices to be made for each stop, consisting of every charger type (m) and no charger type ($+1$).

A graphical representation of (3.5) can be seen in fig. 3.2. Each instance is represented as a rectangle, and the height and color of the rectangle denotes the size of the solution space for that instance.

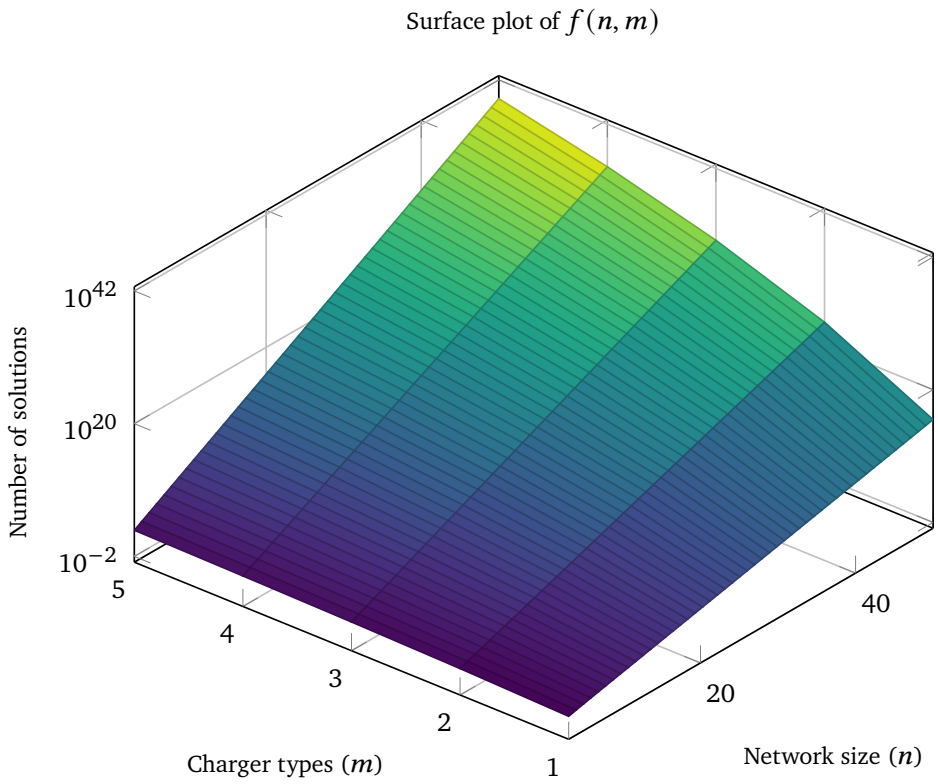


Figure 3.2: Number of solutions for 250 instances of the MT-ECSP, respecitvely.

We have now shown that the number of possible charging configurations grows exponentially with respect to the size of the network. This exponential growth, known as *the combinatorial explosion*[28], is present in many other classical combinatorial optimization problems. Examples being TSP, KP, and VC, where they, too, exhibit an exponential growth.

3.5 Complexity Analysis

In this section, we prove that the MT-ECSP is $\mathcal{NP}$-hard. Recall from section 1.3.1 that if a problem is $\mathcal{NP}$-complete, then its corresponding optimization version is $\mathcal{NP}$-hard. Thus, the procedure for proving $\mathcal{NP}$-hardness is first showing that the MT-ECSP is in $\mathcal{NP}$. Then, show that an already known $\mathcal{NP}$-complete problem, B, can be reduced to the MT-ECSP, in polynomial time, $B <_p$ MT-ECSP, or MT-ECSP $<_p B$. We demonstrate the latter.

Theorem 1. *The Multiple-Trip Electric Charging Station Problem (MT-ECSP) is $\mathcal{NP}$-hard.*

Proof. To prove the hardness of the MT-ECSP, we show that it is in $\mathcal{NP}$ and is $\mathcal{NP}$-complete by reduction from 0-1 IP.

1. **MT-ECSP $\in \mathcal{NP}$:**

   Given a solution to the MT-ECSP, we can verify in polynomial time whether a solution is valid. Since the definition of the MT-ECSP is an optimization problem, we define its decision version as the following:

   Is there an assignment of v_j to s_i such that

   $$\sum_{s_i \in r_w} v(s_i) \geq k_w \quad \textit{and} \quad \sum_{s_i \in S} c(s_i) \leq t \quad \forall w = 0, 1, \ldots, p,$$

   where t is a threshold value for the objective function.

   Let Z be a certificate to the MT-ECSP verification algorithm, containing the charger type assigned to the unique stops in the network. For the instance previously created using fig. 3.1, $Z = \{0, v_1, 0\}$, and $S = \{s_0, s_1, s_2\}$. Then, the verification algorithm assigns z_i to s_i, and checks whether

   $$\sum_{s_i \in r_w} v(s_i) \geq k_w \quad \text{and} \quad \sum_{s_i \in r_w} c(s_i) \leq t \quad \forall w = 0, 1, \ldots, p$$

   where each step of the algorithm can be computed in polynimial time

with the respect to the input size. Therefore, the overall complexity of the verification algorithm is polynomial, and followingly, MT-ECSP $\in \mathcal{NP}$.

2. **MT-ECSP is $\mathcal{NP}$-complete:**

   Recall from section 1.2.2 that the 0-1 IP in its canonical form is given by

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t} \quad & Ax \leq b \\ & x \geq 0, \end{aligned}$$

   where

   - A is the coefficient matrix. It contains the coefficients of the decision variables in the inequality constraints.
   - b is the constraint vector. It represents the upper bounds on the inequality constraints.
   - c is the cost vector. It contains the coefficients of the decision variables in the objective function.
   - x is the decision variable vector. It represents the variables in the problem that must take binary values, as required by the 0-1 IP formulation.
   - b, c, and x are column vectors.

   We construct the vector x by transforming each original variable from S into a set of binary decision variables, one for each assignment of v_j to s_i, such that x $= \{x_{00}, x_{01}, \ldots, x_{0m}, \ldots, x_{nm}\}$. These binary decision variables, x_{ij}, indicate whether s_i takes v_j. If all x_{ij}'s are 0 for a specific variable s_i, it means that s_i takes no value, i.e., the stop is not equipped with a charger. This transformation takes $O(nm)$ time, where n is the size of S and m is the size of V.

   Recall that at most one charger can be installed at a stop. We formulate this as n constraints:

$$\sum_{j=0}^{m} x_{ij} \leq 1 \quad \forall i \in 0, 1, \ldots, n$$

   Lastly, we introduce p energy constraints such that the charging infrastructure installed on every route provides recharging facilities at least

equal to the energy consumed.

$$\sum_{i=0}^{n}\sum_{j=0}^{m} q_{wi}v_j x_{ij} \geq k_w \quad \forall w \in 0, 1, \ldots, p$$

Then, the 0-1 IP formulation can be written as:

$$\begin{aligned}
&\text{minimize} && \sum_{i=0}^{n}\sum_{j=0}^{m} c_j x_{ij} && && (3.6)\\
&\text{subject to} && \sum_{j=0}^{m} x_{ij} \leq 1 && \forall i \in 0, 1, \ldots, n, && (3.7)\\
& && \sum_{i=0}^{n}\sum_{j=0}^{m} q_{wi}v_j x_{ij} \geq k_w && \forall w \in 0, 1, \ldots, p, \text{ and} && (3.8)\\
& && x_{ij} \in \{0, 1\} && \forall i \in 0, 1, \ldots, n, \forall j \in 0, 1, \ldots, m, &&
\end{aligned}$$

which is the model of the MT-ECSP.

We express the above formulation in its canonical form, and get $\mathrm{c}^T\mathrm{x}$:

$$\begin{bmatrix} \mathrm{C}_0 & \mathrm{C}_1 & \ldots & \mathrm{C}_m \end{bmatrix}_{1\times m} \begin{bmatrix} \mathrm{X}_0 \\ \mathrm{X}_1 \\ \vdots \\ \mathrm{X}_{m,} \end{bmatrix}_{m\times 1}$$

where $\mathrm{X}_j = \begin{bmatrix} x_{j0} & x_{j1} & \ldots & x_{jn} \end{bmatrix}^T_{1\times n}$, and $\mathrm{C}_j = \begin{bmatrix} c_j & c_j & \ldots & c_j \end{bmatrix}_{1\times n}$. Lastly, $\mathrm{Ax} \leq \mathrm{b}$:

$$\begin{bmatrix} 1_n & 0_n & \ldots & 0_n \\ 0_n & 1_n & \ldots & 0_n \\ \vdots & \vdots & \ddots & \vdots \\ 0_n & 0_n & \ldots & 1_n \\ \mathrm{U} & \mathrm{U} & \ldots & \mathrm{U} \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{U} & \mathrm{U} & \ldots & \mathrm{U} \end{bmatrix}_{(n+w)\times m} \begin{bmatrix} \mathrm{X}_0 \\ \mathrm{X}_1 \\ \vdots \\ \mathrm{X}_m \end{bmatrix}_{m\times 1} \leq \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ -k_0 \\ \vdots \\ -k_w \end{bmatrix}_{(n+w)\times 1},$$

where 1_n and 0_n are n-dimensional vectors of 1's and 0's respectively, $\mathrm{U} = \begin{bmatrix} -u_0 & -u_1 & \ldots & -u_m \end{bmatrix}_{1\times n}$, and u_j is the product of $q_{wi}v_j$.

We observe, by multiplying the matrices and exploiting its components, we get the following system of equations

$$\begin{aligned}
x_{00} + x_{01} + \cdots + x_{0m} &\leq 1 \\
x_{10} + x_{11} + \cdots + x_{1m} &\leq 1 \\
&\vdots \\
x_{n0} + x_{n1} + \cdots + x_{nm} &\leq 1 \\
-u_0(x_{00} + x_{01} \cdots + x_{0m}) & \\
-u_1(x_{10} + x_{11} \cdots + x_{1m}) & \\
&\vdots \\
-u_m(x_{n0} + x_{n1} + \cdots + x_{nm}) &\leq -k_0 \\
&\vdots \\
-u_0(x_{00} + x_{01} \cdots + x_{0m}) & \\
-u_1(x_{10} + x_{11} \cdots + x_{1m}) & \\
&\vdots \\
-u_m(x_{n0} + x_{n1} + \cdots + x_{nm}) &\leq -k_w,
\end{aligned}$$

which matches the constraints given in the 0-1 IP formulation of the MT-ECSP. Note that u_j and k_p are multiplied by -1 to properly formulate the constraints in the $\mathrm{Ax} \leq \mathrm{b}$ form. We emphasize that 1_n, 0_n, U, C_j, and X_j are included to make the formulation more compact and readable.

The decision version of the MT-ECSP as a 0-1 IP can be formulated by introducing a threshold value of the objective function, t, and reads:

Is there a binary vector $\mathrm{x} \in \{0,1\}^n$ *such that* $\mathrm{c}^T\mathrm{x} \leq t$ *and* $\mathrm{Ax} \leq \mathrm{b}$*?*

Let x^* be a certificate to the verification algorithm. For instance,

$$\mathrm{x}^* = \{x^*_{00}, x^*_{01}, \ldots, x^*_{0m}, x^*_{1m} \ldots, x^*_{nm}\} = \{1, 0, \ldots, 0, 1 \ldots 1\}.$$

We denote y as the matrix-vector product of Ax^*, and z as the objective function value $\mathrm{c}^T\mathrm{x}^*$. Then, the verification algorithm can be expressed as follows:

$$y_i \leq b_i \quad \forall i \in 1, \ldots, m \quad \text{and} \quad z \leq t \tag{3.9}$$

If both conditions in eq. (3.9) hold true, then the verification algorithm outputs *yes*. Each step of the verification algorithm can be computed in

polynomial time with respect to the input size. Specifically, computing y and z involve a linear number of multiplications and additions, while both criteria in eq. (3.9) involve a linear number of comparisons. Therefore, the overall complexity of the verification algorithm is polynomial.

To summarize step 2, the MT-ECSP can be reduced to a 0-1 IP in polynomial time as follows. First, transform the set S into a binary vector x by combining S and V, which takes $O(nm)$ time. Then, introduce $n + p$ constraints, and formulate the 0-1 IP in its canonical form. Lastly, verify the solution in polynomial time.

Thus, we conlude that MT-ECSP $<_p$ 0-1 IP.

By combining the results of steps 1 and 2, we conclude that the MT-ECSP is $\mathcal{NP}$-complete. Thus, its corresponding optimization version is $\mathcal{NP}$-hard. □

4

Implementation

In this chapter, we present the results of our development phase, where Google Or-Tools' *Constraint Programming Satisfiability* (CP-SAT) solver is applied to solve the MT-ECSP. First, we demonstrate how the notation and model from chapter 3 interacts with Google OR-Tools' API in the devised program. Then, we show how the experiments are set up.

4.1 Overview

The implementation is written to follow the notation used in section 3.2, and the mathematical formulation of the model given in eqs. (3.6) to (3.8) as close as possible. The source code shown in the next subsections will be snippets from the complete implementation, which can obtained by a request to the author.

4.1.1 MT-ECSP Instance

The implementation consists of a single class, illustrated in fig. 4.1. The *init* function initializes the MT-ECSP instance with the sets $\mathcal{V}$, $\mathcal{C}$, $\mathcal{K}$, $\mathcal{R}$ and $\mathcal{Q}$. All experiments assumes bi-directional routes. Therefore, $\mathcal{Q}$ is an optional argument, and should be used to define an alternative route symmetry.

```
class MT_ECSP:
    def __init__(self, V, C, K, R, Q=None):
    def __define_variables(model):
    def __initialize_model():
    def search_for_all_solutions(min, obj_fun):
```

Figure 4.1: The MT-ECSP class with functions.

```
def __initialize_model(self):
    model = cp_model.CpModel()

    X, energy_constraints, x =
        self.__define_variables(model)

    for x_i in x.values():
        model.Add(sum(x_i) <= 1)

    for w, k_w in enumerate(self.K):
        model.Add(energy_constraints[w] >= k_w)
```

Figure 4.2: The model is initialized as an instance of *CpModel*, provided by Google OR-Tools' API. After a preprocessing step, the constraints defined in eqs. (3.7) and (3.8) are added to the model.

In fig. 4.2, we demonstrate how the model is initialized.

The creation of the binary decision variables x_{ij}, eqs. (3.6) and (3.8), are illustrated in fig. 4.3.

4.1.2 Solving the Model

After initilazing the model, it is then passed to the solver. We demonstrate this in fig. 4.4. This apporach allows for multiple solutions with an equal objective value to be found.

The entry point of the program is given in fig. 4.5. Observe, that the *search_for_solutions* function is called twice; first to find the optimal objective function value, and then to search the solution space.

```python
def __define_variables(self, model):
    X = [[[] for _ in r_p] for r_p in self.R]
    objective_function = 0
    energy_constraints = []
    x = {}

    for w in range(len(self.R)): # decision variables
        for i, s_i in enumerate(self.R[w]):
            if s_i not in x:
                x[s_i] = [model.NewBoolVar(f'x{s_i}_{j}')
                ↪  for j in range(len(self.V))]
            X[w][i] = x[s_i]

    for x_, Q_w in zip(X, self.Q): # energy constraints
        v_f = 0
        for x_i, q_w_i in zip(x_, Q_w):
            for j in range(len(x_i)):
                v_f += q_w_i * self.V[j] * x_i[j]
        energy_constraints.append(v_f)

    for i in x.keys(): # objective function
        for j in range(len(x[i])):
            objective_function += x[i][j] * self.C[j]

    self.objective_function = objective_function

    return X, energy_constraints, x
```

Figure 4.3: Defintion of variables and constraints. X is a transformation of the set $\mathcal{R}$, and contains the boolean descision variables x_{ij} instead of the stops s_i. x is the set of all unique x_{ij}.

```
    def search_for_solutions(self, minimize=None,
    ↪  objective_value):
        model = self.__initialize_model()
        if minimize:
            model.Minimize(self.objective_function)
            solver = cp_model.CpSolver()
            status = solver.Solve(model)

            return solver.ObjectiveValue()
        else:
            model.Add(self.objective_function ==
            ↪  objective_value)

            solver = cp_model.CpSolver()
            solver.parameters.enumerate_all_solutions =
            ↪  True
            status = solver.Solve(model)
```

Figure 4.4: The CP-SAT solver is solving the MT-ECSP in a two-stage process. First, the objective function is minimized to find the optimal objective value. Then, a new model is created, where the objective value is added as an additional constraint.

```
    def main():
        V = [5, 15, 50]
        C = [10, 100, 1000]

        K, R = generate_network()

        mt_ecsp = MT_ECSP(V, C, K, R)
        obj_val = mt_ecsp.search_for_solutions()
        mt_ecsp.search_for_solutions(minimize=True,
        ↪  obj_val)
```

Figure 4.5: The *main* function of the program. The sets $\mathcal{V}$ and $\mathcal{C}$ are defined. Then, $\mathcal{K}$ and $\mathcal{R}$ are initialized during the generation of the network.

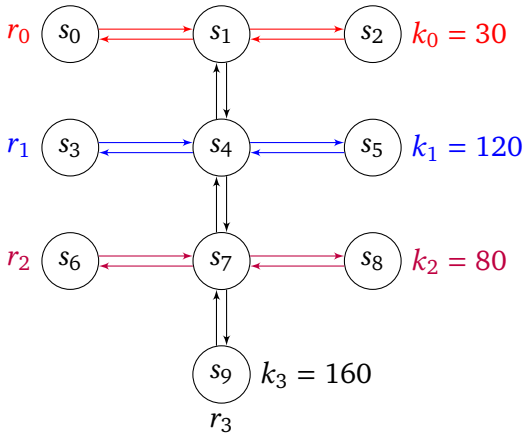


Figure 4.6: An arbitrary network consisting of 4 routes and 10 stops. Each route is distinguished by the coloring of the route name and edges between each stop s_i. Observe that the stops s_2, s_5, s_8 are shared between rotues.

4.2 Setup of Experiments

We devised several experiments to test the correctness of our implementation, the feasibility of the model, and the performance of the program. Three experiments were manually defined, the route illustrated in fig. 3.1, the network illustrated in fig. 4.6, and a network suggested by Hyke, illusrated in fig. 4.7. Table 4.1 contains metadata associated with fig. 4.7.

In addition, we created route initialization procedures both for single and multiple routes. The procedure for generating a single route is demonstrated in fig. 4.8. By not supplying any arguments, the function sets up the route shown in fig. 3.1.

Generation of multiple routes can be done randomly by passing arguments to the function shown in fig. 4.9. The default setting is the generation of the routes illustrated in fig. 4.6. We emphasize that the values of K are arbritary numbers, and denotes the energy consumption of traversing the route.

Lastly, fig. 4.10 demonstrates how the network suggested by Hyke can be initialized.

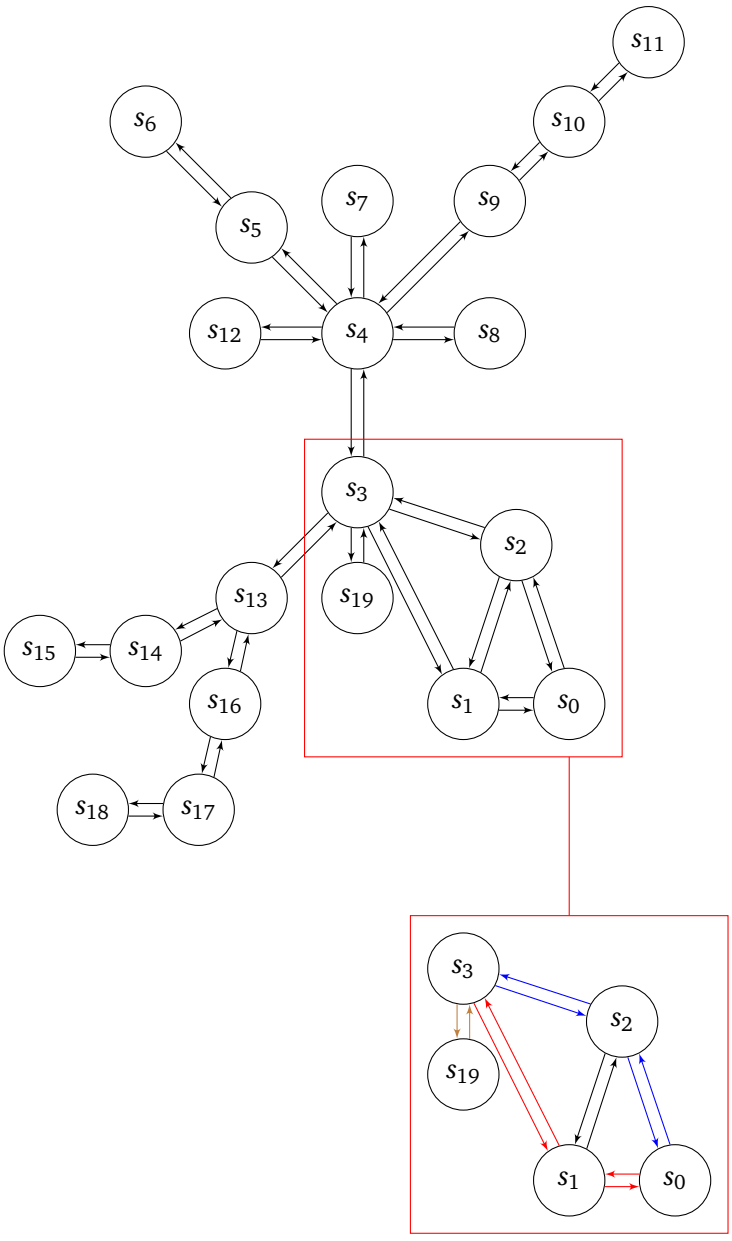


Figure 4.7: A suggested network provided by Hyke. The red boxes emphasizes the area covering s_0, s_1, s_2, s_3, and s_{19}. In the bottom red box, the colored edges denotes the different routes in that area.

r_w	$s_i \in r_w$	k_w
r_0	s_0, s_2, s_3	122
r_1	s_0, s_1, s_3	149
r_2	s_1, s_2	32
r_3	s_4, s_8	47
r_4	s_4, s_{12}	29
r_5	s_4, s_5, s_6	11
r_6	s_4, s_7	18
r_7	s_4, s_9, s_{10}, s_{11}	47
r_8	s_3, s_{19}	73
r_9	s_3, s_{13}, s_{14}, s_{15}	54
r_{10}	s_3, s_{13}, s_{16}, s_{17}, s_{18}	57

Table 4.1: The routes, stops, and the energy usage for the network shown in fig. 4.7.

```
    def generate_synthetic_route(n=None, K=None):
        if not n and not K:
            return [30], [[0, 1, 2]]
        else:
            return [K], [list(range(n))]
```

Figure 4.8: Single route generation procedures.

```
    def generate_synthetic_network(w=None, n=None):
        if not w and not n:
            return [30, 120, 80, 160], [[0, 1, 2], [3, 4,
            ↪  5], [6, 7, 8], [9, 7, 4, 1]]
        else:
            K = [random.randint(10, 180) for _ in
            ↪  range(w)]
            R = []

        for _ in range(w):
            r_w = []
            r_w_size = random.randint(3, n)
            while len(r_w) < r_w_size:
                value = random.randint(0, n - 1)
                if value not in r_w:
                    r_w.append(value)

            R.append(r_w)

        return K, R
```

Figure 4.9: Network initilization procedures.

```
    def generate_suggested_network(E):
        NM = 0.539956803
        NME = E * NM

        R = [[0, 2, 3], [0, 1, 3], [1, 2], [
            4, 8], [4, 12], [4, 5, 6], [4, 7], [4, 9, 10,
            ↪  11], [3, 19], [3, 13, 14, 15], [3, 13, 16,
            ↪  17, 18]]

        K_ = [8.25 * NME, 10.05 * NME, 2.20 * NME, 3.2 *
        ↪  NME, 2.0 * NME, 0.75 * NME,
            1.23 * NME, 3.20 * NME, 4.9 * NME, 3.67 * NME,
            ↪  3.88 * NME]

        K = [int(k_w * 2) for k_w in K_]

        return K, R
```

Figure 4.10: Initialization of the network provided by Hyke. The argument E denotes the energy consumption of the ferry. The variable NME denotes the energy usage per nautical mile. The numbers in $K\_$ denotes the total distance in kM for the route.

5 Performance

This section presents the results of the experiments. It is divided into two subsections, manually defined networks, and benchmarks, respecitvely. All the experiments were set up using three charger types, where the values v_j denotes charged energy for a fixed time interval. First, we present the results of the manually defined networks. Then, a benchmark of the application.

5.1 Manually Defined Networks

The first two experiments were conducted to verify the correctness of our implementation and model using the networks illustrated in fig. 3.1 and fig. 4.6, respecitvely. The third experiment was conducted to a test our solution on a real network, provided Hyke. All experiments were solved to optimality within 30 ms.

We show the optimal solutions for the experiments in fig. 5.1, fig. 5.2, and fig. 5.3, using the output from our application. Two optimal solutions were found for fig. 4.6.

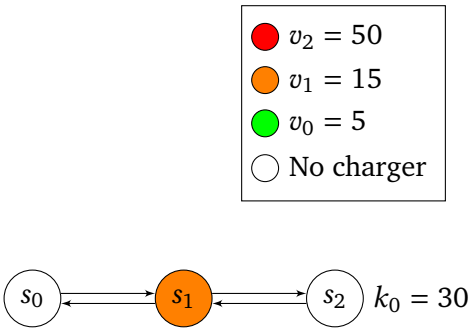
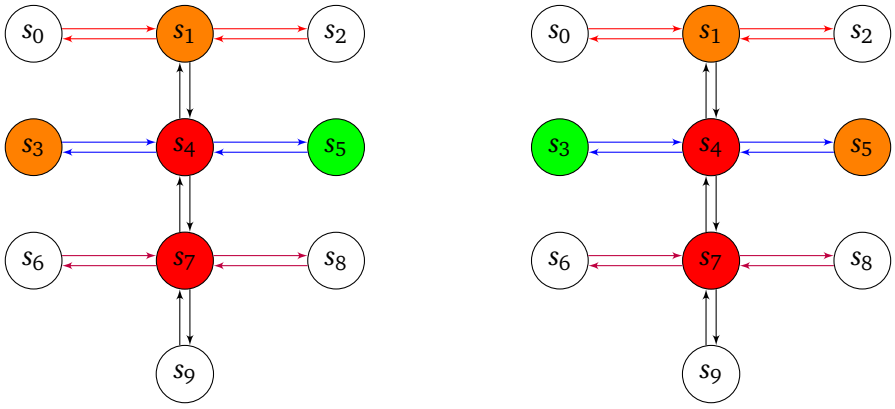


Figure 5.1: Optimal solution to the network illustrated in fig. 3.1.



(a) First optimal solution. **(b)** Second optimal solution.

Figure 5.2: Optimal solutions to the network illusrated in fig. 4.6.

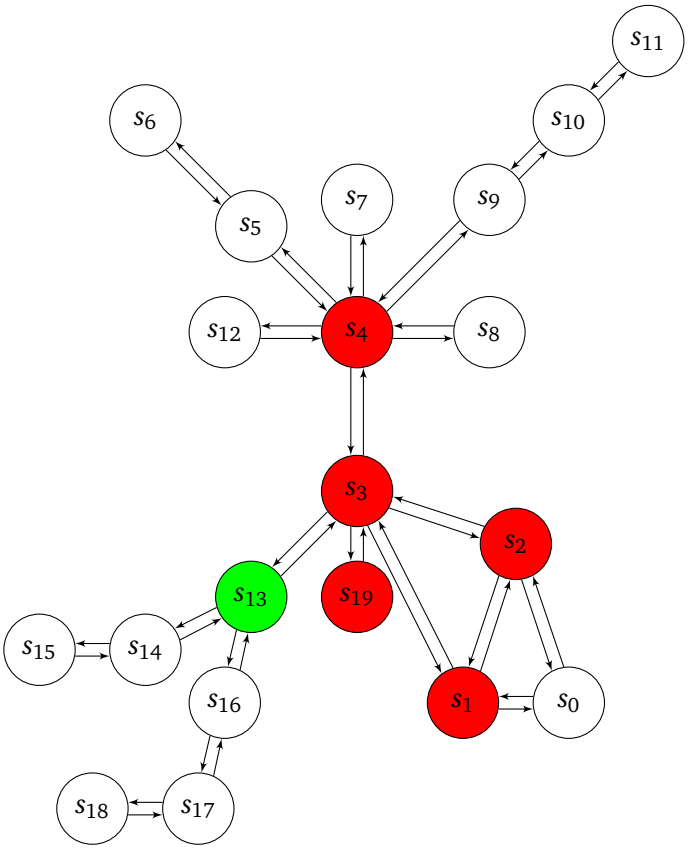


Figure 5.3: The optimal solution to the network suggested by Hyke.

5.2 Benchmarks

As all the results shown in the previous section were solved to optimality within a suprisingly short time frame, we conducted additional experiments with larger instances to gain additional insights into the performance of the application. This was achieved by benchmarking our application for 30 instances of the MT-ECSP for a single route.

The size of each route initially starts from $n = 3$, and incrementally increases to $n = 33$. K remains constant for every instance, and is set to 300. The runtime of the benchmark was monitored, alongside data retrieved from the solver. In the output below, we present the profiling result for the *main* function of our program using the **line_profiler**[47] tool.

```
Timer unit: 1e-06 s

Total time: 54.1581 s
File: Solver_routes2.py
Function: main at line 173

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
   ...
   180        30         26.1      0.8      0.0  for i in range(3, 33, 1):
   181        30        209.0      6.7      0.0      K, R = generate_synthetic_route(K=300, n=
      ↪ i)
   182
   183        30       1017.1     32.8      0.0      mt_ecsp = MT_ECSP(V, C, K,
   184        30          9.0      0.3      0.0                        R=R, Q=None)
   185        30     908346.7  29301.5      1.7      obj_val = mt_ecsp.search_for_solutions()
   186        30   53248460.1 1717692.3     98.3      mt_ecsp.search_for_solutions(minimize=
      ↪ False, objective_function=obj_val, i=i)
```

The benchmark ran for a total of 54.1581 seconds, where the second call to *search_for_solutions* used 98.3% of the execution time; when the solver searched for multiple solutions with the minimized objective value. In figs. 5.4 to 5.7, we show statistics provided by the solver for the same benchmark.

In 5.4, the optimal objective value increases from 0 to 4000 for $n = 3$ to $n = 4$. This is due to an infeasible solution being found for $n = 3$, as no objective value is at least K. We observe that the optimal objective value rapidly decreases from $n = 5$ to $n = 11$, and then slowly flattens from $n = 12$ to $n = 33$.

Figure 5.5 shows, that as the number of optimal solutions found increases, so too does the time taken to solve. This is, however, not always the case, as the highest number of optimal solutions was found when $n = 20$, whereas the highest time spent was for $n = 23$.

Figures 5.6 and 5.7 illustrates more clearly the factors contributing to time expenditure. Both the number of explored search branches and the number of conflicts reached their peak values when the time spent searching was at its highest.

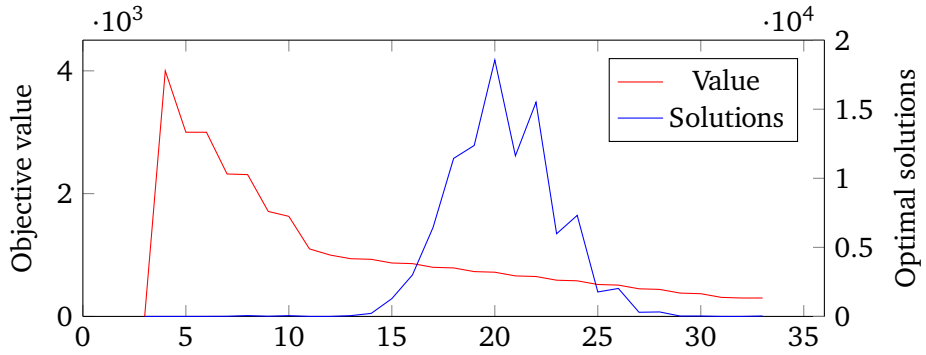


Figure 5.4: Optimal objective values, alongside the number of optimal solutions found for n.

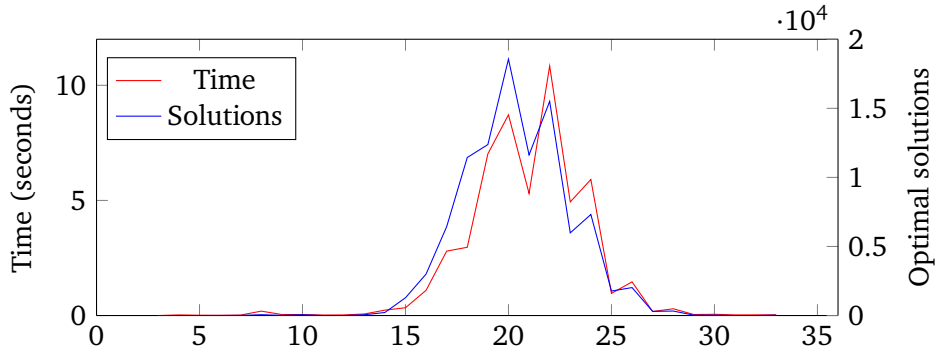


Figure 5.5: Time spent solving for n, alongside the number of optimal solutions found.

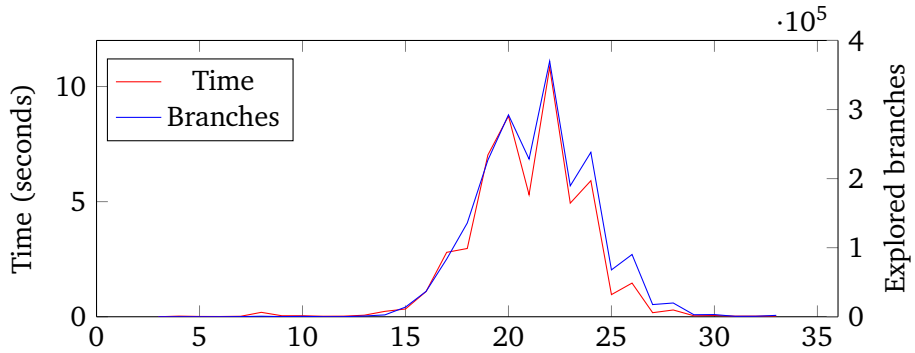


Figure 5.6: Number of search branches explored, alongside time taken to solve for n.

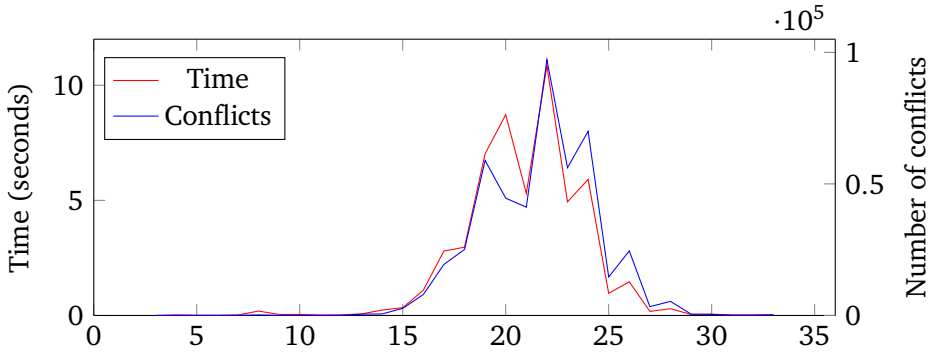


Figure 5.7: Number of occured conflicts for size n.

6 Discussion

In this chapter, we discuss the results found in chapters 3 to 5. First, we discuss how the results of chapter 3 were obtained. Then, we reflect on the results from chapters 4 and 5.

6.1 Computational Complexity

The primary objective of this thesis was to establish the computational complexity of the MT-ECSP, where the paper "Electric Vehicle Charging Station Placement for Urban Public Bus Systems' by Wang et al. [18] was used as a starting point. The first leg of determining the complexity involved formulating the nomenclature, mentioned in section 3.2. After closer inspection, we recognized that the stops in each route and the charging effect provided by the chargers could be seen as variables and values, respectively.

Subsequently, the procedure of assigning values to variables described a finite set of choices to be made, with contraints on those choices. For instance, the sum of all variables in a route should be at least some integer, denoting the energy consumption of that route. Moreover, only one charger could be installed at a stop, implying once a charger had been installed, all other choices would be disregarded. Section 3.3 did not quite capture the latter constraint, and so, we investigated methods for representing this relationship.

Wang et al. [18] used a binary decision varible x_i to denote whether a charging station was placed at stop i in their model. As a result, we extended x_i to x_{ij}, denoting both the conditions of whether a charger was installed, and which charger type. The study of IPs was a consequence of this result. Seeing that Karp proved the special case of IPs, 0–1 IPs to be $\mathcal{NP}$-complete [24], this became a natural candidate for the reduction, leading up establishing the computational complexity of the MT-ECSP.

6.2 Proof of Concept

The secondary objective of this thesis was to compliment the scientific results by investigating solution strategies for the MT-ECSP as a proof of concept. Ultimately, we utilized Google OR-Tools' API alongside their CP-SAT solver to meet this end. In chapter 4, we demonstrated how such a solution strategy could be implemented, and presented the results in chapter 5.

The findings in figs. 5.1 to 5.3 revealed valueable insights. Firstly, all three experiments were solved to optimality, suggesting that our model is properly defined, and the implementation is correct. The solution found for fig. 5.1 was also the same solution we found by observation in section 3.3, conforming to the above sentence. Lastly, all experiments were solved within the fraction of a second, showing that real world instances of the MT-ECSP can be solved quickly and precisely.

A powerful feature of the CP-SAT solver is the ability to search for multiple solutions. For the practical purposes of MT-ECSP, multiple solutions may provide valueable insights to the charging infrastructure. As mentioned in section 3.1, grid availability is disregarded. In practice, however, absense of grid availability may impose additional costs of installing a charger, as sufficient infrastructure may not be present at a candidate site. Finding multiple optimal solutions to the MT-ECSP allows for alternative strategies to be discovered, which may reduce the overall cost of the infrastructure.

This is reflected in the result shown in fig. 5.2, where different chargers installed on s_3 and s_5 can be switched, while yielding the same objective value.

Based the results shown in figs. 5.2a and 5.2b, r_2 and r_3 had a charging infrastucture yielding 20 and 50 kW more than the energy consumed by traversing the routes, respectively. This provides the possibility of exploiting the excess energy. Due to the deterministic assumptions of the MT-ECSP, the only exploitation would include increasing the average speed used to traverse those routes, which in turn would imply a higher energy consumption.

6.3 Benchmarks

In section 5.2, we presented a performance analysis of our implementation, consisting of a benchmark and a profiling test. The purpose of the analysis was to experiment with instances of the MT-ECSP where the solver would identify solutions from a larger solution space (determined by eq. (3.5)), and possibly a larger number of solutions. This way, we would be able to determine what impact—if any—the solution space had on the performance of the program.

The results presented in fig. 5.4 show that the solver found a maximum of nearly 20000 optimal solutions when $n = 20$, determined by the objective value used for the search procedure. Figure 5.5 clearly shows, however, that there is no correlation between the size of the solution space and the runtime of the application. The profiling result compliments this, as only 1.7% of the runtime was used to find optimal objective values for all instances.

Rather, figs. 5.5 to 5.7 suggest that the runtime is determined by the number of optimal solutions found *and* the procedures involved in identifying them. In other words, how *hard* it is to find solutions for a given objective value.

7

Concluding Remarks

This thesis set out to explore the problem of determing minimum requirements for charging infrastructures in networks where electric ferries operate on a fixed schedule. Drawing inspiration from literature, and particularly from Wang.et al. [18], we named the problem MT-ECSP; the Multiple-Trip Electric Charging Station Placement problem. This due to the similarities between the objectives of the problems.

The categorical nature of the MT-ECSP was formulated as an optimization problem, rooted in minimizing the overall cost of the charging infrastructure. A key characteristic of the MT-ECSP included finding a charging infrastructure that could—in theory—support the vehicles indefinetly, with partial recharging.

The primary objective of the thesis was to determine the complexity of the MT-ECSP, and propose methods for solving it. To meet this end, we analysed the characteristics of the MT-ECSP, showing—including, but not limited to—that the solution space grew exponentially with respect to the size of the input. Subsequently, we proved that the MT-ECSP was $\mathcal{NP}$-hard, by reduction to the famous $\mathcal{NP}$-complete problem, 0–1 IP.

By exploiting the model, we implemented an application to provide a proof of concept, complimenting the results from chapter 3. This was the secondary objective of this thesis.

The proof of concept involved conducting several experiments, each focusing on specific instances of the MT-ECSP, using Google OR-Tools' CP-SAT Solver as a tool to solve them. The results showed that the MT-ECSP model is both feasible and solveable in an acceptable time frame for synthetic networks and real world instances.

As a concluding remark, the author is satisfied with the results, and most importantly, the learning outcome obtained during the course of this research project. A final note on future work is elaborated in the next section.

7.1 Future Work

The results discussed in the previous chapter show that the MT-ECSP can be solved to optimality within a very short time frame, both for custom and real ferry networks. While the MT-ECSP primiarily is meant to find optimal charging infrastructures for electric ferries operating under a fixed schedule, we argue that ferries are a specific use case, and not by any means a limitation for usage.

Wang et al.[18] tested their solution on a real public bus system with 115 bus routes, and over 900 stops. While we have not considered an instance of such magnitude in this thesis, it would without a doubt provide deeper insights into the solvability and possible limitations of the model and implementation. If the MT-ECSP proves to be solveable for such large scale networks, then comparing our results with Wang et al. would make an enticing project.

We mentioned in Sections 1.3.3 and 1.4 a thesis by Masliakova [42], where Ant Colony Optimization with Genetic Algorthims was used to determine optimal routing and charging procedures for electric buses. Key takeaways from the thesis in relation to this research project includes a parameter denoting the amount of trips a route is traversed by a bus during the span of a day, and varying charging intervals for each stop. The charging intervals were determined such that the bus always has enough energy to reach the next stop in a route.

We assumed the ferries to be operating at full capacity all day, implying a fixed charging interval. A possible improvement to our model would be to include optimal charging intervals, alongside the frequency in which a route is traversed. For instance, could the charging infrastructre cost be reduced by strategically charging for a prolonged amount of time at selected stops? This would result in a modified objective function, possibly involving the elements of the set c to be a determined by a cost function, where the cost no longer involves just the price of a charger, but the time spent charging as well.

A final improvement could include stochatic elements, such as customer demand to be added to the model. This way, the energy consumption of the ferries would no longer be known for certain, as well as the fixed charging interval. Consequently, the energy consumption associated with traversing a route would be a fixed number, with a varying degree of uncertainty.

Bibliography

[1] A. Gouldson, S. Colenbrander, A. Sudmant, F. McAnulla, N. Kerr, P. Sakai, S. Hall, E. Papargyropoulou, and J. Kuylenstierna, "Exploring the economic case for climate action in cities," *Global Environmental Change*, vol. 35, pp. 93–105, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0959378015300169

[2] I. E. Agency, "Co2 emissions from fuel combustion (2016 edition): Key co2 emissions trends," 2016.

[3] K. Zhang and S. Batterman, "Air pollution and health risks due to vehicle traffic," *Science of The Total Environment*, vol. 450-451, pp. 307–316, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0048969713001290

[4] G. S. Santos, I. Sundvor, M. Vogt, H. Grythe, T. Haug, B. Høiskar, and L. Tarrason, "Evaluation of traffic control measures in oslo region and its effect on current air quality policies in norway," *Transport Policy*, vol. 99, pp. 251–261, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0967070X19307462

[5] D. L. Schrank, T. J. Lomax *et al.*, "The 2007 urban mobility report," Texas Transportation Institute, Tech. Rep., 2007.

[6] INRIX. (2022) Inrix global traffic scorecard. Accessed: 2023-01-20. [Online]. Available: https://inrix.com/scorecard/

[7] L. M. Fulton, "Three revolutions in urban passenger travel," *Joule*, vol. 2, no. 4, pp. 575–578, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2542435118300941

[8] Samferdselsdepartementet, "Teknologi for bærekraftig bevegelsesfrihet og mobilitet," Tech. Rep. N0573B, 2019, accessed: 2023-01-17. [Online]. Available: https://www.regjeringen.no/contentassets/ccdc68196014468696acac6e5cc4f0e7/rapport-

teknologiutvalget_web.pdf

[9] Meld. St. 13, "Report to the storting (white paper): Norway's climate action plan for 2021-2030," 2020/2021. [Online]. Available: https://www.regjeringen.no/no/dokumenter/meld.-st.-13-20202021/id2827405/

[10] M. S. Tannum and J. H. Ulvensøen, "Urban mobility at sea and on waterways in norway," *Journal of Physics: Conference Series*, vol. 1357, no. 1, p. 012018, oct 2019. [Online]. Available: https://dx.doi.org/10.1088/1742-6596/1357/1/012018

[11] I. E. Aslaksen and E. B. Svanberg, "A combined ferry service network design and dial-a-ride system for the kiel fjord," Master's thesis, NTNU, 2020.

[12] K. L. Rødseth, H. Havre, U. Lien, M. Ness, and K. Fagerholt, "Optimal route to battery electric high-speed vessel services," *Available at SSRN 4330381*, 2023.

[13] D. Villa, A. Montoya, and A. M. Herrera, "The electric riverboat charging station location problem," *Journal of Advanced Transportation*, vol. 2020, pp. 1–16, 2020.

[14] F. Driessen, "A method for optimal charging station placement for ships: Combining a flow-refueling location model and an agent-based simulation," Master's thesis, Delft University of Technology, 2022.

[15] G. Bitar, M. Breivik, and A. M. Lekkas, "Energy-optimized path planning for autonomous ferries," *IFAC-PapersOnLine*, vol. 51, no. 29, pp. 389–394, 2018.

[16] H. S. C. Ferries, "Hyke," 2023, accessed: 2023-01-12. [Online]. Available: https://hydroliftsmartcityferries.com

[17] M. Lai and H. K. Lo, "Ferry service network design: optimal fleet size, routing, and scheduling," *Transportation Research Part A: Policy and Practice*, vol. 38, no. 4, pp. 305–328, 2004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0965856403001198

[18] X. Wang, C. Yuen, N. U. Hassan, N. An, and W. Wu, "Electric vehicle charging station placement for urban public bus systems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 1, pp. 128–139, 2017.

[19] K. Tangrand, "Optimal routing of electric vehicles in networks with charg-

ing nodes: Ant colony optimization with genetic algorithms," Master's thesis, Narvik University College, 2015.

[20] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979, vol. 174.

[21] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *Journal of the ACM (JACM)*, vol. 19, no. 2, pp. 248–264, 1972.

[22] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the third annual ACM symposium on Theory of computing*, 1971, pp. 151–158.

[23] B. Esfahbod, "Euler diagram for P, NP, NP-complete, and NP-hard set of problems," Wikipedia, n.d., accessed: 2023-04-25. [Online]. Available: https://commons.wikimedia.org/wiki/File:P_np_np-complete_np-hard.svg

[24] R. M. Karp, "Reducibility among combinatorial problems, complexity of computer computations (re miller and jw thatcher, editors)," 1972.

[25] K. Wayne, "Intractability iii," Accessed: 2023-02-15, 2005, revised version of the lecture slides created by Kevin Wayne, accompanying the textbook *Algorithm Design* by Jon Kleinberg and Éva Tardos. Original and official version distributed by Pearson. Copyright © 2005 Pearson-Addison Wesley. All rights reserved. [Online]. Available: https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/10ExtendingTractability.pdf

[26] G. B. Dantzig, A. Orden, and P. S. Wolfe, *Notes on Linear Programming: Part I: The Generalized Simplex Method for Minimizing a Linear Form Under Linear Inequality Restraints*. Santa Monica, CA: RAND Corporation, 1954.

[27] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*. Dover Publications, 1998.

[28] L. A. Wolsey, *Integer Programming*, 1st ed. Wiley-Interscience, 1998.

[29] J. Kleinberg and E. Tardos, *Algorithm Design*. Pearson Education India, 2006.

[30] R. G. Parker and R. L. Rardin, *Discrete optimization*. Elsevier, 2014.

[31] F. Rossi, P. Van Beek, and T. Walsh, *Handbook of constraint programming*.

Elsevier, 2006.

[32] K. Apt, *Principles of constraint programming*. Cambridge university press, 2003.

[33] M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier, "The constraint logic programming language chip," in *Proceedings of the International Conference of Fifth Generation Computer Systems*, 1988, pp. 693–702.

[34] P. Stuckey. (2020) From clp(r) to minizinc: There and back again. Accessed: 2023-04-01. Online conference. [Online]. Available: https://www.youtube.com/watch?v=FIyn99_HOPw

[35] M. Schiffer and G. Walther, "The electric location routing problem with time windows and partial recharging," *European Journal of Operational Research*, vol. 260, no. 3, pp. 995–1013, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221717300346

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

[37] X. Li, J. Huang, Y. Guan, Y. Li, and Y. Yuan, "Electric demand-responsive transit routing with opportunity charging strategy," *Transportation Research Part D-Transport And Environment*, vol. 110, SEP 2022.

[38] J.-G. Kim and M. Kuby, "The deviation-flow refueling location model for optimizing a network of refueling stations," *International Journal of Hydrogen Energy*, vol. 37, no. 6, pp. 5406–5420, 2012, optimization Approaches to Hydrogen Logistics. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360319911020337

[39] F. Paparella, T. Hofman, and M. Salazar, "Joint optimization of number of vehicles, battery capacity and operations of an electric autonomous mobility-on-demand fleet," in *2022 IEEE 61st Conference on Decision and Control (CDC)*, 2022, pp. 6284–6291.

[40] J. Löfberg, "Yalmip : A toolbox for modeling and optimization in matlab," in *In Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004.

[41] M. Metais, O. Jouini, Y. Perez, J. Berrada, and E. Suomalainen, "Too much or not enough? planning electric vehicle charging infrastructure: A review of modeling options," *Renewable and Sustainable Energy Reviews*, vol. 153, p. 111719, 2022. [Online]. Available: https://www.sciencedirect.

com/science/article/pii/S136403212100993X

[42] K. Masliakova, “Optimal routing and charging procedures for electric buses,” Master’s thesis, UiT - The Arctic University of Norway, 2016.

[43] Gurobi Optimization, LLC, “Gurobi Optimizer Reference Manual,” 2023. [Online]. Available: https://www.gurobi.com

[44] M. Team, “Minizinc,” Accessed: 2023-03-12, 2023. [Online]. Available: https://www.minizinc.org/

[45] N. Nethercote, P. Stuckey, R. Becket, S. Brand, G. Duck, and G. Tack, “Minizinc: Towards a standard cp modelling language,” in *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming, volume 4741 of LNCS*. Springer, 2007, pp. 529–543.

[46] L. Perron and V. Furnon, “Or-tools,” Google. [Online]. Available: https://developers.google.com/optimization/

[47] R. Kern, “line_profiler,” https://github.com/pyutils/line_profiler, 2023.