TU WIEN Informatics

Computational Optimization Approaches for Distributing Battery Exchange Stations for Electric Scooters

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Bernhard Kreutzer, BSc
Matrikelnummer 0927086

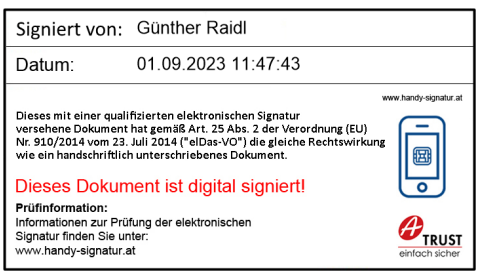
an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Günther Raidl
Mitwirkung: Projektass. Dipl.-Ing. Dr.techn. Thomas Jatschka, BSc

Wien, 31. August 2023

Bernhard Kreutzer



Günther Raidl

TU WIEN Informatics

Computational Optimization Approaches for Distributing Battery Exchange Stations for Electric Scooters

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Bernhard Kreutzer, BSc
Registration Number 0927086

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Günther Raidl
Assistance: Projektass. Dipl.-Ing. Dr.techn. Thomas Jatschka, BSc

Vienna, 31st August, 2023

Signed by:	Bernhard Kreutzer
Date:	31.08.2023 09:11:42

This document is digitally signed!
This document bearing a qualified electronic signature has the same legal standing as a handwritten document in accordance with Article 25 (2) of Regulation (EU) No. 910/2014 of 23 July 2014 ("eIDAS-VO").
Validation: Information for checking the electronic signature can be found at: www.a-trust.at/pdf

Bernhard Kreutzer

Günther Raidl

Erklärung zur Verfassung der Arbeit

Bernhard Kreutzer, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 31. August 2023

Signiert von:	Bernhard Kreutzer
Datum:	31.08.2023 09:12:57

Dieses Dokument ist digital signiert!

Dieses mit einer qualifizierten elektronischen Signatur versehene Dokument hat gemäß Art. 25 Abs. 2 der Verordnung (EU) Nr. 910/2014 vom 23. Juli 2014 ("eIDAS-VO") die gleiche Rechtswirkung wie ein handschriftlich unterschriebenes Dokument.

Prüfinformation: Informationen zur Prüfung der elektronischen Signatur finden Sie unter: www.a-trust.at/pdf

Bernhard Kreutzer

Danksagung

Zuerst würde ich mich gerne bei Prof. Günther Raidl bedanken der es mir ermöglicht hat an diesem fantastischen Projekt zu arbeiten. Zusätzlich gebührt ein spezieller Dank Thomas Jatschka bei dem ich mich besonders für seine ausgezeichnete Unterstützung bedanken möchte. Die ausführlichen Diskussionen kombiniert mit euren Ratschlägen haben die Arbeit zu einem erfolgreichen Projekt gemacht.

Des Weiteren möchte mich bei Honda R&D, für die Kooperation und Finanzierung dieses Projekt bedanken. Im speziellen gebührt Dank Yusuke Okamoto, Hiroaki Kataoka, Tadashi Hayashida von der Honda Motor Company und Tobias Rodemann vom Honda Research Institute Europe welche mit Ideen und Daten wertvollen Einfluss auf das Projekt genommen haben.

Weiters möchte ich mich bei Matthias Rauscher bedanken, welcher im Rahmen seiner Masterarbeit ebenfalls am Projekt beteiligt war.

Abschließend möchte ich mich noch bei meiner Familie und meinen Freunden bedanken für die Unterstützung, die sie mir haben zukommen lassen.

Acknowledgements

First, I would like to thank Prof. Günther Raidl, who allowed me to work on this fantastic project. In addition, a special thank goes to Thomas Jatschka, whom I would like to thank in particular for his excellent support. The hours of discussions combined with your advice made the work a successful project.

I would also like to thank Honda R&D for the cooperation and financing of this project. Special thanks go to Yusuke Okamoto, Hiroaki Kataoka, Tadashi Hayashida from the Honda Motor Company and Tobias Rodemann from the Honda Research Institute Europe, who have had a beneficial influence on the project with ideas and data.

Additionally, I thank Matthias Rauscher, who was also involved in the project as part of his master's thesis.

Finally, I would like to thank my family and friends for the continued support they have given me.

Kurzfassung

Diese Arbeit betrachtet das *Battery Exchange Station Location Problem* 2 (BEXSLP2), welches zum Ziel hat, die optimale Platzierung von Batterietauschstationen für elektrische Scooter in einem dicht besiedelten Gebiet zu finden. Dazu wird eine Funktion verwendet, die mehrere Zielsetzungen gegeneinander abwägt, um so die Errichtungskosten der Batterietauschstationen, die Ladekosten und die Wegstrecken der Kunden und Kundinnen um die Batterie zu wechseln zu minimieren und gleichzeitig alle Kunden und Kundinnen mit Batterien zu versorgen. Benutzer können bei solchen Batterietauschstationen entladene Batterien gegen vollgeladene tauschen. Diese Batterien werden daraufhin wieder geladen und, sobald sie vollständig aufgeladen sind, wieder zum Tausch angeboten. Die Anzahl der Batterien, die gleichzeitig geladen werden können, hängt von der Anzahl der Module ab, die in der Batterieladestation verbaut sind. Jedoch ist die Zahl der Module, die in Batterieladestationen verbaut werden können, limitiert. Der betrachtete zyklische Planungshorizont wird in gleich große, aufeinanderfolgende Intervalle unterteilt. Des Weiteren wird der Weg, den Benutzer sich dabei bewegen müssen, durch einen Start- und Endpunkt definiert und es wird angenommen, dass sie den schnellstmöglichen Pfad wählen, wenn sie den Umweg zu einer Batterietauschstation fahren. Um die gewählten Umwege zu minimieren, ist es möglich, sowohl existierende Stationen mit zusätzlichen Modulen zu erweitern als auch neue Stationen an vordefinierten Orten zu errichten. Zur Lösung des Problems wird eine *mixed integer linear programming* (MILP) Formulierung entwickelt. Diese wird zusätzlich auch mit einem *iterated greedy* Algorithmus in einer *matheuristic* verschmolzen. Eine *matheuristic* kombiniert exakte Techniken der mathematischen Programmierung mit heuristischen Methoden. Der *iterated greedy* Algorithmus wird zum iterativen Zerstören und Rekonstruieren von Teilen der Lösung verwendet. Dabei wird eine Teillösung systematisch erweitert, um den Bedarf an Batterieladestationen zu erfüllen. Zum Testen der entwickelten Algorithmen werden zum einen Instanzen, die von Honda R&D zur Verfügung gestellt wurden, als auch künstlich generierte Instanzen verwendet. Das MILP-Programm ist dabei nicht in der Lage, zufriedenstellende Ergebnisse für die größten Instanzen zu liefern. Die *iterated greedy* Heuristik kann allerdings weiterhin bedenkenlos angewandt werden und liefert für eben jene Instanzen um 40% bessere Resultate. Beachtenswert ist hierbei weiters, dass selbst die initial gefundene Lösung der *iterated greedy* Heuristik für die größten Instanzen schon bessere Ergebnisse liefert als das MILP Programm.

Abstract

This thesis considers the Battery Exchange Station Location Problem 2 (BEXSLP2), which aims to find the optimal configuration for battery-swapping stations for electric scooters over an urban region. In order to do so, a multi-objective target function is used to minimize the setup cost, charging cost, and the distances customers must ride to swap the battery while providing all customers with batteries. When a battery is exchanged at a battery-swapping station, it will be charged and, once charging is complete, again provided to customers. The number of batteries that can be charged simultaneously depends on the number of battery modules built into the respective station. Furthermore, the number of modules that can be built into each station is limited. We consider a cyclical time horizon of one day discretized into equal consecutive time intervals. The starting and end locations define the trips that customers must travel. Users are always assumed to take the quickest (shortest) path when traveling to a battery-swapping station. When determining the optimal placement of stations, it is feasible to construct new stations (in predefined locations) and/or extend existing stations. A mixed integer linear programming (MILP) formulation is developed to solve this problem. Moreover, we also developed an iterated greedy matheuristic that combines the exact approach of the MILP with a heuristic. This algorithm uses greedy procedures to destroy and (re)construct a solution iteratively. In each iteration, a partial solution is extended to fulfill the current demand. A slight variation of the original MILP is used for extending solutions. Both algorithms are tested on two different sets of instances, one derived from data provided by Honda R&D and the other generated artificially. The results show that the largest tested instances are too difficult to solve with our MILP approach. However, in those cases, the iterated greedy metaheuristic is still applicable and outperforms the MILP by reducing the gap up to 40%. Additionally, even the construction heuristic outperforms the MILP in those instance sets.

Contents

CHAPTER 1

Introduction

Electric vehicles and their vast potential as environmentally friendly traffic alternatives have recently been increasingly acknowledged in all parts of the world. Nevertheless, their limited range and long charging times still prevent large-scale adoption. However, swapping batteries is a valid option for electric scooters, contrary to electric cars, since the batteries are sufficiently compact and light to be directly replaced by a customer in a few simple steps. Such an implementation would eliminate the need for charging times and would not take longer than refueling one's car. In such a scenario, supermarkets or conventional petrol stations could offer a battery-swapping service in which the customer would be able to exchange their nearly empty battery for a fully loaded one. Returned batteries are then recharged on site and, once fully charged, provided for exchange again. For companies offering such a service, an essential factor of consideration is the placement of such stations. On the one hand, the costs of constructing battery-swapping stations must be taken into account. On the other hand, if there are not enough stations, the adaptation of electric scooters will stagnate or even decrease. Therefore, finding the best locations for battery-swapping stations is crucial. In Figure 1.1 a depiction of such a battery swapping station can be seen.

1.1 Problem Statement

The main challenge of the issue addressed in this thesis is to determine the optimal placement for battery-swapping stations that satisfies a certain amount of customer demand while at the same time minimizing the costs of building the stations and charging the batteries and, eventually, minimize the induced delay for traveling to a station during a trip. In this work, we assume a time horizon of one day that is discretized into equal consecutive time intervals, for example, hours. Moreover, we consider a cyclical planning horizon, i.e., the predecessor of the first interval is the last one, and the successor of the last one is the first interval again. Customer trips are specified as origin-destination

Figure 1.1: A typical battery-swapping station developed by Honda[1]

(O/D) pairs, and it is assumed that customers always travel on the corresponding shortest path. Moreover, the number of customers that require new batteries during a trip for each O/D pair and time interval is known. Batteries can only be exchanged at certain time intervals (opening hours), which may differ from station to station. The number of batteries a station can handle, i.e., load simultaneously, is contingent on the number of battery modules built at the respective station. Each battery module can hold and charge a certain number of batteries. We specifically distinguish between the initial module and subsequent modules at a station, as the initial module generally tends to have a different capacity as well as higher setup costs than the subsequent modules. Due to production limitations, only a certain number of battery modules can be allocated among all the stations to extend them. In the production of modules, there is no difference between the initial and subsequent modules. Stations can only be built at designated locations. Moreover, existing stations can be further expanded unless the respective station is already at full capacity.

The problem is inspired by a similar research project - the Battery Exchange Station

[1] https://global.honda/newsroom/news/2022/p221025eng.html

Location Problem (BEXSLP) of a partner in the industry, Honda R&D, Japan. In contrast, their problem formulation considers individual users in a more detailed manner, which makes it hard to find solutions to larger instances of the problem within an adequate time frame. Therefore, in this work, the formulation mentioned above is adapted by considering users in an aggregated manner inspired by [JORR20]. Consequently, the formulation is expected to scale substantially better to problem instances with more customers and more potential locations for stations than the originally proposed formulation.

1.2 Aim of the Work

This work aims to formulate and solve the Battery Exchange Station Location Problem 2 (BEXSLP2), which uses features from both the BEXSLP and the Multi-Period battery-swapping Station Location Problem (MBSSLP) [JORR20]. All three apply different approaches to find the best placement for stations. Compared to the BEXSLP2, the BEXSLP uses a more detailed model of customer behavior. The MBSSLP is an approach that generalizes the BEXSLP by aggregating customers and the routes they ride. However, the MBSSLP needs to account for the same level of detail, i.e., modules or opening hours. Compared to the MBSSLP, the BEXSLP2 uses a multi-objective objective function. The objective function weighs in construction cost, charging cost, and induced delay for customers. The main challenge is to combine the in-depth customer modeling of the BEXSLP with the performance-enhancing approach to aggregate customers.

Initially, we will present a mixed integer linear programming (MILP) formulation for the BEXSLP2 and show that the problem is $\mathcal{NP}$-hard. Subsequently, a matheuristic [BMRBR09] combining an iterated greedy approach and mathematical programming techniques is developed to generate solutions to larger instances. Iterated greedy heuristic is a concept that was first introduced by Ruiz and Stützle [RS07], in which greedy procedures are used to destroy and (re)construct a solution iteratively. Our iterative greedy procedure constructs/extends a solution period-wise, i.e., in each iteration, a partial solution is extended to fulfill not only the current demand but also demand that had so far been unconsidered at a chosen time interval. A MILP is used for extending solutions. Afterward, various strategies for destroying a solution will be investigated. A solution is generally destroyed by selecting a subset of the demand variables and setting it to zero, generating an infeasible/incomplete solution. Additionally, if a station is destroyed, all associated demand variables must first be set to zero, and then the station can be set to zero. According to the iterated greedy scheme, these 'construct-and-destroy' procedures will be applied iteratively to improve the solution systematically. Different strategies will be applied for the destruction and the repairing of the solutions and compared to each other.

We evaluate our MILP formulation and the iterated greedy scheme on two sets of instances. One set is derived from data provided by Honda R&D, while the other is generated

artificially. Additionally, a comparison to solutions obtained by solving the BEXSLP is presented.

1.3 Key Results

The BEXSLP2 can achieve better solutions than the original BEXSLP in less time. We limited the program's execution time to four hours and compared the results afterward. Most instances generated from the data provided by Honda R&D were easily solvable in the given time frame. The most difficult part is to prove the optimality of a solution. In the Honda instances, the optimal solutions were found within minutes, while it takes hours to prove the optimality of the given solution. This, however, was not always possible in the given time frame. Only the two smallest benchmark sets could be solved optimally for the artificially generated instance set.

The different weightings of the objective function substantially influence the program's performance. In one Honda instance, the problem went from unsolvable within the four-hour time frame to being solved within seconds by using different weightings for the objective function. When increasing the delay weight by 200 times, the program finished after 38 seconds. Increasing it even further, we were able to solve the problem even faster. The problem is hardest to solve when the cost of delay and construction are balanced. In the artificially generated instances, we can see that higher weighting of delay leads to longer solving times and increased optimality gaps.

However, the largest instance set of the artificially generated set proved too hard to solve for the exact approach, resulting in average gaps up to 82.7%. The implemented iterated greedy heuristic achieved better results than the MILP program for these benchmark sets and configurations by a considerable margin. All construction heuristics proved very effective in providing a better starting point. Even the worst construction heuristics provided an average gap of only 65.6%. Additionally, we noticed that only destroying stations and their associated demand does not lead to good quality solutions. We improved our destroy operators by destroying additional demand from the remaining stations weighted by the objective function. Using the worst construction heuristic combined with two hours of repair and destroy, the destroy operators that destroy additional demand could significantly outperform those without this feature. The average gap for the most challenging benchmark set improved from around 60% to 50%.

When repairing a solution, we noticed that selecting not a single time period but instead selecting an interval containing multiple consecutive time periods performs best. It strikes a good balance between the quality of the solution and the time consumed to construct the solution. This also holds true for the construction heuristic.

Generally speaking, one of the most complex parts of the problem is the combination of minimizing the delay while at the same time being restricted by the number of modules that can be built. Not limiting the number of modules leads to smaller gaps and solving times.

CHAPTER 2

Methodology

This chapter provides the terminology and the foundations of the algorithmic concepts used throughout this work. We review exact techniques, namely linear programming (LP) and mixed integer linear programming (MILP) models. Afterward, we examine heuristic methods, which scale better than exact methods.

2.1 Exact Methods

In this section, we give a brief introduction to LP and MILP. It is mostly based on the works of Bertsimas and Tsitsiklis [BT97], Bertsimas and Weismantel [BW05], Dantzig [DT03], Nemhauser and Wolsey [WN99], and Schrijver [Sch98].

2.1.1 Linear Programming

Initially, we start by defining an LP. Suppose that we have a vector of n continuous decision variables $x = (x_1, \ldots, x_n)$ with $x \in \mathbb{R}^n$ and an associated cost vector $c = (c_1, \ldots, c_n)$ with $c \in \mathbb{R}^n$. An LP problem looks, as defined by Bertsimas and Tsitsiklis [BT97], as follows:

$$\min c'x \tag{2.1}$$

$$\text{subject to } a_i'x \geq b_i \qquad \forall i \in M_1 \tag{2.2}$$

$$a_i'x \leq b_i \qquad \forall i \in M_2 \tag{2.3}$$

$$a_i'x = b_i \qquad \forall i \in M_3 \tag{2.4}$$

$$x_j \geq 0 \qquad \forall j \in N_1 \tag{2.5}$$

$$x_j \leq 0 \qquad \forall j \in N_2 \tag{2.6}$$

The objective of any linear program is to find a variable assignment x that minimizes the objective function given in Equation (2.1). Note that any minimization problem can be transformed into an equivalent maximization problem and vice versa by using the following equality:

$$\min c'x = \max -c'x \tag{2.7}$$

This corresponds to multiplication with -1. Nevertheless, the problem dealt with in this thesis is a minimization problem; therefore, this section only focuses on the minimization variant. The remaining Equations (2.2) - (2.6) are called the constraints of the program, which x needs to satisfy. If x satisfies all constraints, then x is a feasible solution. The set of all feasible solutions is called the feasible set and forms the feasible region. A feasible solution x that also minimizes the objective function is called an optimal solution. It follows that the value of the solution is $c'x$. It is possible that multiple optimal solutions exist or that no solution exists (if the feasible set is empty). Moreover, equalities may be converted into inequalities and vice versa. To transform an equality $a_i'x = b_i$ one must add the inequalities $a_i'x \geq b_i$ and $a_i'x \leq b_i$. In the opposite direction inequalities of the form $a_i'x \geq b_i$ can be reformulated using a slack variable $s_i \in \mathbb{R}$ as $a_i'x + s_i = b_i$. Consequently, we can formulate LP problems using terms of the form $a_i'x \geq b_i$ exclusively. Therefore, the given form can be written in a more compact way using matrix notation:

$$\min c'x \tag{2.8}$$
$$\text{subject to } Ax \geq b \tag{2.9}$$

Duality

An important concept that is especially relevant to solve MILPs are bounds. There are two types of bounds: primal and dual bounds. Primal bounds satisfy all constraints; therefore, a primal bound is an upper bound for a minimization problem. A dual bound refers to the theoretically best possible value, typically achieved with relaxation of the problem, and is consequently a lower bound for a minimization problem. (On the other hand, in the case of a maximization problem, the primal bound is the lower bound while the dual bound is the upper bound.) Additionally, every feasible solution to the dual problem also provides a valid dual bound for the primal problem, which is stated in the following theorems:

Theorem 2.1.1. *Weak duality. Let c and p be feasible solutions to the primal and the dual problem, respectively, then*

$$p'b \leq c'x \tag{2.10}$$

Moreover,

Theorem 2.1.2. *Strong duality. If a linear programming problem has an optimal solution, so does its dual, and the respective solution values are equal.*

Solving Linear Programs

Today's most relevant algorithm to solve linear programs is the simplex algorithm proposed by Dantzig [Dan90], which is widely used due to its good performance. The algorithm starts using an initial arbitrary basic feasible solution and moves iteratively to another adjacent feasible solution using a so-called pivoting step. A pivoting step replaces one active basic variable with a nonbasic variable. A basic variable is a variable that is part of the feasible solution. However, only variables that reduce the cost are eligible. Finally, such a step is no longer possible after a finite number of pivoting steps. In this case, we know that the current solution is optimal. Although this method has an exponential run-time in the worst case, it is the fastest solution method in practice. Compared to the simplex method, Khachiyan proved in 1979 [Kha79] that the ellipsoid method could solve LP problems in polynomial time. Consequently, it is known that LP problems are $\mathcal{P}$-hard. Nevertheless, the ellipsoid method is only of theoretical interest due to its poor performance in practice. Another algorithm that can solve an LP in polynomial time is the interior point method proposed by Karmarkar [Kar84]. Compared to the ellipsoid method, the interior point method is still relevant as it is used in combination with simplex algorithms in state-of-the-art LP solvers. Commercial LP solvers offer several algorithms and can often decide which algorithm performs best for a given problem.

2.1.2 Mixed Integer Linear Programming

The previous section stated that LP problems are in $\mathcal{P}$, i.e., can be solved in polynomial time. The idea of MILP is to expand LP by adding discrete variables. Those can be either binary variables, i.e., only zero and one, or be contained in $\mathbb{Z}$. One benefit of using binary variables is, for example, that they enable the usage of solution components, i.e., in our case, this is used to describe whether a station is built or not. Nevertheless, this comes at the cost that the problem is no longer contained in $\mathcal{P}$. Therefore, MILP problems are $\mathcal{NP}$-hard, which means that we can no longer expect to solve a MILP in polynomial time unless $\mathcal{P} = \mathcal{NP}$. A MILP problem is defined as follows:

$$\min c'x + d'y \tag{2.11}$$
$$\text{subject to } Ax + By \leq b \tag{2.12}$$
$$x \geq 0 \tag{2.13}$$
$$y \in \mathbb{Z}^n \tag{2.14}$$

Solving Mixed Integer Linear Programming

Since MILP problems are $\mathcal{NP}$-hard, they can no longer be solved like the LP problems. Therefore, a typical approach to those problems is to consistently improve both the primal and the dual bound, i.e., for a minimization problem, decreasing the primal and increasing the dual bound. The optimal solution is found if the primal and dual bounds are equal. Modern computing typically achieves this if the difference between the primal

and dual bound is lower than some small nonnegative value ϵ. A prime example of this procedure is branch and bound (B&B). Such an algorithm partitions the search space into subproblems, a procedure called branching, and calculates the bounds for said subproblems called bounding. The best-found solution is called incumbent, which can also be used as primal bound. Different strategies are used to determine which subtree should be explored, e.g., depth-first search, best-node-first search, or breadth-first search. The advantages of the different strategies and more detailed information can be found in the book by Wolsey and Nemhauser [WN99]. An essential aspect of B&B is that the number of subproblems will be consistently reduced. There are three typical ways a subproblem will be pruned by either

- Prune by infeasibility, i.e., a subproblem does not contain any valid solutions
- Prune by optimality, i.e., primal and dual bound are equal; therefore, the solution is already known
- Prune by bound, i.e., the best known bound already is better than any solution possible in the subproblem

2.2 Iterated Greedy Heuristic

Typically, $\mathcal{NP}$-hard problems can no longer be proven to optimality if the problem is complex enough due to limitations of available computational resources. Consequently, heuristics are often the only solution available if one wants to achieve good results. The goal of a heuristic is to provide reasonable but only sometimes optimal solutions. In our case, we decided that the iterated greedy heuristic originally proposed by Ruiz and Stützle [RS07] suits the problem best. The main idea of the heuristic is as follows. At first, we create an initial solution with a greedy construction heuristic, i.e., in a traveling salesperson problem, for example, we could always pick the best available edge until we have completed the trip. Afterward, the heuristic makes use of destruction and construction phases iteratively. An algorithmic outline of the iterated greedy algorithm is given in Algorithm 2.1. The remaining chapter will be primarily based on Stützle and Ruiz [SR18]

2.2.1 Greedy Construction Heuristic

A greedy construction heuristic aims to build a solution step by step, starting from nothing or an already partial solution. Iteratively we add a solution component (i.e., the edge in a traveling salesperson problem) to the current partial solution until we obtain a complete and valid solution. It is called greedy because each time a component is added, a function estimates each component's value and the best component will be selected. This is further illustrated in Algorithm 2.2. Instead of always choosing the same best-rated solution, we can also use let randomness influence the chosen component. A common modification to the simple greedy construction heuristic would add a function

Algorithm 2.1: Iterated Greedy

Input: variable(s) v need to create an initial solution
Output: potentially improved solution s

```
1 s* = GenerateInitialSolution(v)
2 while termination criterion not met do
3 |   s_p = Destruction(s*)
4 |   s' = Repair(s_p)
5 |   s* = AcceptanceCriterion(s*, s')
6 end
7 return s*
```

that selects components at random; however, better components are preferred. Such a procedure would be a weighted random greedy construction heuristic. A possible example of such a function is tournament selection which will be explained later in Chapter 2.2.5

Algorithm 2.2: Greedy Construction Heuristic

Input: -
Output: a solution s with its objective value

```
1 s = {}
2 obj_value = 0
3 while s is not a complete solution do
4 |   choose a best-rated solution component c with value v
5 |   s = s ∪ c
6 |   obj_value = obj_value + v
7 end
8 return s, obj_value
```

The advantages of a greedy construction heuristic or a weighted random greedy construction heuristic include that they are relatively fast at generating an initial solution, which is typically better than completely randomly created solutions. Secondly, greedy algorithms are often used to seed local search methods such as iterative improvement algorithms, leading to improved local optima quality, faster identification of local optima, and a better trade-off between computation times and solution quality. Finally, in some cases, it is possible to obtain some guarantees of the quality of a solution due to the greedy construction heuristic. Note that the construction procedures used to generate the initial solution and the procedure used to repair a solution may differ.

2.2.2 Destruction

Two main aspects must be considered regarding the destruction of a given solution. The first interesting question is: How much of the solution should be destroyed? Neither destroying only one solution component nor destroying every solution component is

desirable. The first would result in a randomized local search, while the latter would result in the constant application of the construction heuristic. Everything in between is a trade-off between search intensification and diversification. Removing large parts of the solution components allows for more distant solutions, while smaller parts result in a more localized search. It is also possible to change or adapt the value between iterations proposed by Battiti et al. [BBM08]. The second important question is which components should be removed. Possibilities range from random selection to weighing function, which removes the perceived worst component of the solution. It is also possible to combine those ideas by selecting those components which are the worst with a higher probability.

2.2.3 Repair

A repair algorithm is used to restore a partial solution (because some components were destroyed before). Again, as was already mentioned with the construction heuristic for the initial solution, a straightforward way is using a deterministic greedy algorithm. However, it should be noted that every combination or idea that can achieve good results and repair a partial solution is a viable option for the repair heuristic.

2.2.4 Acceptance Criterion

The acceptance criterion defines which solution is the base of the subsequent destroy & repair cycle, i.e., either the new solution is accepted, or the old one remains the base. The most extreme acceptance criteria are that either only improvements will be accepted or every new solution will be accepted. This, again, is a trade-off between search intensification and diversification. A typical approach is the Metropolis criterion. If a solution is an improvement, it is automatically accepted. If the solution is not an improvement, it is accepted depending on the quality of the solution and the temperature. The temperature in the algorithm is a series of numbers approaching zero. The higher the temperature, the more likely it is that a worse solution will be accepted.

Gap

Another aspect we want to highlight is the computation of the optimality of a solution. Since we already know that solving MILP problems is $\mathcal{NP}$-hard, more complex problems cannot be solved optimally in feasible time. Consequently, we use gaps to measure the optimality of a solution. A gap, therefore, measures the quality of a solution. We use the following definition by Gurobi [1] to measure our MILP gaps.

Let z_p correspond to the primal bound, which in that case has the same objective value as the incumbent solution to our problem, and let z_d refer to the best found dual bound to the same problem. Then the gap of the solution is calculated using z_p and z_d by

$$\text{gap} = 100\% \cdot \frac{|z_p - z_d|}{|z_p|}. \tag{2.15}$$

[1] https://www.gurobi.com/

Nevertheless, an iterated greedy heuristic only improves the primal bound. Therefore, to calculate the solution gap, we use the value of the dual bound found by the corresponding MILP run. Henceforth, let $f*$ be the objective value of the solution generated by the heuristic and z_d the dual bound found using the MILP. Then we calculate the gap of the heuristic gap* as:

$$\text{gap}^* = 100\% \cdot \frac{|f^* - z_d|}{\tilde{|}f^*|}. \tag{2.16}$$

2.2.5 Tournament Selection

Tournament selection is a method of selecting individuals from a population. In this thesis, we often require a pseudo-random distribution. Therefore, we decided on a tournament selection. The reason for this is that we need some elements of randomness. However, we want to avoid complete randomness. We instead prefer to select better elements more often. In more detail, this results in the following; k elements of the whole population are selected randomly. Those k elements are then sorted using a function; for example, this could be the demand associated with the element. Consequently, the element with the highest weight is the first element on the list. The first element of the list has a probability of p to be chosen. If the first element is not chosen, the second element has a chance of p being chosen again. Therefore, the probability that the element at a given position pos in the list is chosen is defined as $p \cdot (1-p)^{pos-1}$.

2.3 Hybrid Methods

So far, only pure versions of exact and metaheuristic solution approaches have been discussed. However, in practice, hybrid approaches, i.e., approaches combining multiple techniques, often perform best. In this thesis, we want to focus primarily on matheuristics. A matheuristic is a technique that combines a metaheuristic with mathematical programming. The goal is to exploit the individual advantages of each approach. Puchinger and Raidl [PR05] describe that there are typically two possibilities to combine mathematical programming and a heuristic. Either use mathematical programming embedded in heuristic algorithms or metaheuristics to improve known mathematical programming techniques. The rest of the section is also based on Puchinger and Raidl [PR05]. Further information, as well as more detailed examples, can also be found there.

Regarding the first possibility, there are many applications. One of those is to solve relaxed problems exactly. Advantages include deriving promising initial solutions as well as heuristically guiding neighborhood search, recombination, mutation, repair and local improvement. Another application is to search neighborhoods in metaheuristics utilizing exact algorithms, also known as (Very) Large Neighborhood Search ((V)LNS). The benefit of incorporating exact methods is that if the neighborhood is chosen appropriately, they can search relatively large neighborhoods and still be an effective search method.

The second possibility (not used in this thesis) is to obtain incumbent solutions and bounds, cutting planes and the pricing of columns.

CHAPTER 3

Related Work

In this chapter, we present existing works related to this thesis. At first, we aim to classify the problem. Afterward, we present interesting projects that either solve similar problems or use similar aspects used in the thesis. Finally, this thesis is a continuation of works previously undertaken by the TU Wien and Honda R&D, Japan.

The BEXSLP2 can be classified in various ways, each describing and focusing on different aspects of the problem. According to Boloori et al. [BF12], it can be classified as Facility Location-Allocation Problem [BF12], i.e., an optimization problem where there is a finite set of users with demand for a service and a finite set of potential locations for facilities that will offer said service to the users. A Facility Location-Allocation Problem is the most basic version of the problem to which it can be reduced. It also resembles the Capacitated Multiple Allocation Fixed Charge Facility Location Problem investigated by Laporte et al. [NSdG19] in which customers need to be assigned to facilities to satisfy their demand while minimizing the costs for building facilities and serving customers. Another important aspect of this problem is that charging takes time, which limits the battery slots considerably. Therefore, it can also be classified as a Multi-Period Facility Location Problem [NSdG19] to highlight that the problem should be solved concerning a time horizon. When limiting the number of stations (or modules), the problem may be categorized as a p-Median problem, according to Laporte et al. [NSdG19].

Problems regarding the optimal placement of recharging or refueling stations for electric vehicles are well-studied in academic literature. Yan et al. [YLK21] are looking for the optimal placement of battery charging stations concerning tourism activities in Taichung City, Taiwan. They formulated their model as an integer network flow problem with side constraints. Similarly, Efthymiou et al. [ECMA17] tried to solve the optimal placement of electric vehicle charging stations in Thessaloniki. They used linear programming with multi-objective optimization and a genetic algorithm. Lin et al. [LLKL22] developed a grid-based multi-objective stochastic allocation model to solve the problem of the

optimal placement of battery-swapping stations. In one of his previous works, Lin et al. [LLYL21] also proposed an optimal scooter battery-swapping station allocation model with uncertainty, which featured a Monte Carlo simulation to solve the uncertainty of battery-swapping event. Wang et al. [WYW+19] proposed an improved differential evolutional algorithm combined with the Monte Carlo searching method to obtain the optimal location of battery-swapping stations in a region in Beijing, China.

Using a MILP model to solve Facility Location or Location-Allocation problems is also very common. Due to the limited availability and accessibility of maternal healthcare clinics in India, Chousksey et al. [CAT22] proposed a MILP model to solve a hierarchical capacitated facility location-allocation problem to predict the optimal placement of such clinics. Additionally, they also made use of valid inequalities to improve the model further. Rathore et al. [RSS20] tried to improve urban solid waste management in India. The objective was to determine the number and placement of garbage bins required; therefore, it was necessary to solve a location-allocation problem they formulated as a MILP model. Boujelben et al. [BGM16] studied a multi-period facility location problem. A two-phase solution approach was developed to solve said problem. In the first phase, a clustering procedure based on set-partitioning formulation was used to calculate the distances and costs of transport from distribution centers to customers. In the second phase, the calculated costs are input to the facility location problem, formulated, and solved as a mixed integer linear programming program. After encountering problems to solve larger instances of said problem, they also proposed a linear relaxation-based heuristic.

Ruiz and Stützle [RS07] first proposed the iterated greedy algorithm. It essentially consists of two main phases, namely a destruction and construction phase, which are applied consecutively. This operation is executed multiple times on a given solution. For a problem similar to the BEXSLP2, Guo et al. [GYL18] used the iterated greedy heuristic. Compared to the BEXSLP2, they did not consider periods; thus, the time required for a battery to charge was also not part of their research. Consequently, it would be difficult to model whether a station is fully occupied by batteries (which are all recharging). Instead, they focus on the users' range anxiety and distance deviations, which they identified as major barriers to the mass adoption of electric vehicles. Gokalp et al. [Gok20] utilized the iterated greedy heuristic to solve an Obnoxious p-Median Problem. The work sought to maximize the distance between each non-facility location and its nearest facility.

A very similar approach to ours was used by Yang et al. [YS15] in their attempt to solve their battery swap stations location routing problem. For the optimal placement of the battery-swapping station, they, too, used a MILP model, which is used in an iterated greedy heuristic. However, before that, they needed to generate a routing plan with a modified sweep algorithm. This approach was expanded by Liu et al. [LGL19] by considering time windows. Jamshidi et al. [JCvEN21] pursued a different goal. Their work aims to optimize taxi requests; therefore, charging electric vehicles during the day

is necessary. They solve this problem using three MILP models sequentially, which use different granularity of the considered time frame. The first considers the whole day in an aggregated fashion. The second and third variants consider smaller pieces of a day and are used to optimize decision-making for the individual vehicle.

3.1 Previous Work

The BEXSLP2 is based on the MBSSLP [JORR20] and the BEXSLP developed by Honda R&D, Japan. The MBSSLP aims to identify optimal battery-swapping station locations and appropriately determine their capacities to cover a specified level of assumed demand at minimum cost. A MILP, modeling the customer demand over time in a discretized fashion and considering battery charging times, is proposed. Additionally, the MBSSLP also considers an inevitable dropout of customers when assigned to stations inducing long detours.

CHAPTER 4

The Battery Exchange Station Location Problem 2

In the *Battery Exchange Station Location Problem 2* (BEXSLP2), the task is to plan the setup of new stations to exchange batteries for electric scooters or extend existing stations to minimize three different objectives while satisfying an expected demand. The three objectives are (a) the setup cost for additional stations and extension modules, (b) the cost of charging batteries, and (c) the total duration of detours for users to exchange batteries.

We consider a time horizon of one day discretized into equally long consecutive time intervals, for example, hours. These intervals are indexed by $\mathcal{T} = \{1, \ldots, t_{\max}\}$. Moreover, we consider the planning horizon to be cyclic, i.e., the predecessor of the first interval is the last one, and the successor of the last one is the first interval.

We make the simplifying assumption that charging any battery always takes the same time, and only fully recharged batteries are provided to customers again. Moreover, as trips in an urban environment are usually relatively short, we further assume that trips start and end in the same time interval.

We assume a battery-swapping station can be set up at any of n different locations referred to as $L = \{1, \ldots, n\}$. Each location $l \in L$ has an associated

- setup cost $c_l \geq 0$ for setting up a station with an initial configuration of BEX modules at this location;
- setup cost $c_l^{\text{modul}} \geq 0$ for each additional BEX module at a location where a station is set up or exists already;
- capacity in terms of the number of battery slots of the initial station configuration $s_l^{\text{ini}} \in \mathbb{N}$;

- maximum number of additional BEX modules allowed at location $e_l^{\max} \in \mathbb{N}$;
- timespan $\mathcal{T}_l^{\text{ex}} \in \mathcal{T}$, in which the station is open for customers and batteries may be exchanged;
- and charging costs $c_l^{\text{dch}} \geq 0$ and $c_l^{\text{nch}} \geq 0$ for batteries during daytime and nighttime (i.e., outside daytime) charging hours, respectively.

The capacity of subsequent modules s^{modul} is the same for all stations.

We also consider that at some locations $l \in L$, a station with a corresponding configuration of BEX modules may have already been set up at a previous time. In this case, the costs c_l for setting up the station are zero. The initial station configuration s_l^{ini} accounts for all existing slots, including the already existing extension modules. If feasible, such a station may still be extended by installing up to $e_l^{\max}$ additional BEX modules.

Customer travel demands are given for origin-destination (O/D) pairs Q; let $m = |Q|$ be the number of these O/D pairs. Moreover, let $w_q \geq 0$ be the expected travel time for each O/D pair $q \in Q$ when taking the most direct route without exchanging batteries. Furthermore, let $\tilde{w}_q^l$ be the expected travel time for the O/D pair $q \in Q$ when making the fastest possible detour to location $l \in L$ for exchanging batteries there. Clearly, $\tilde{w}_q^l \geq w_q$ will hold for any $q \in Q,\ l \in L$.

We only consider one type of battery but different vehicle types that require different numbers of batteries. We assume that all vehicle batteries are always exchanged together simultaneously. Let $\mathcal{I} \subset \mathbb{N}$ be the set of vehicle types represented by the corresponding numbers of needed batteries. The expected number of users with vehicle type $i \in \mathcal{I}$ that need to change batteries on trip $q \in Q$ during a time interval $t \in \mathcal{T}$ is denoted as d_{qi}^t.

All customer demand has to be fulfilled.

$$d_{\text{sat}} = \sum_{q \in Q} \sum_{t \in \mathcal{T}} \sum_{i \in \mathcal{I}} i \cdot d_{qi}^t. \tag{4.1}$$

Note that we weigh demands by the number of batteries of their respective vehicle type, such that vehicles with fewer batteries are not favored during the optimization, as vehicles with more batteries require more resources to satisfy their demand.

Moreover, the number of total BEX modules is restricted due to production limitations. Therefore, $z^{\text{modules}} \in \mathbb{N}$ refers to the maximum number of available BEX modules.

A solution is primarily given by a pair of vectors $x = (x_l)_{l \in L} \in \{0,1\}^n$ and $y = (y_l)_{l \in L}$ with $y_l \in \{0, \ldots, e_l^{\max}\}$ where $x_l = 1$ indicates that a swapping station is to be used at location l and y_l is the corresponding number of additionally installed BEX modules. BEX modules may only be allocated at locations where a swapping station is located, i.e., $x_l = 0 \to y_l = 0$, or expressed as a linear inequality

$$e_l^{\max} \cdot x_l \geq y_l, \qquad l \in L. \tag{4.2}$$

Consequently, the maximum number of battery slots at a location l is $s_l^{\text{ini}} \cdot x_l + s^{\text{modul}} \cdot y_l$.

It is assumed that customers who want to exchange batteries specify their trip data (origin, destination, approximate time) online and are automatically assigned to an appropriate station for the exchange (if one exists). This way, better utilization of the swapping stations can be achieved. Consequently, let assignment variables a_{qli}^t denote the part of the expected demand of O/D pair $q \in Q$ and vehicle type $i \in \mathcal{I}$ which we assign to a location $l \in L$ during time interval $t \in \mathcal{T}_l^{\text{ex}}$.

A battery returned to a station $l \in L$ during a period $t \in \mathcal{T}_l^{\text{ex}}$ can only be provided to a customer again after t^{c} periods from $\mathcal{T}$ have passed. We denote the set of times in which a battery is being charged when returned to a station at time t as $\mathcal{T}^{\text{ch}}(t)$ with t being the time in $\mathcal{T}$ at which the battery starts charging and $((t + t^{\text{c}} - 1) \bmod t_{\max}) + 1$ being the last time period in which the battery is being charged.

In the MBSSLP, returned batteries are unavailable for t^{c} periods by effectively reducing a station's capacity of batteries available for an exchange within the next t^{c} periods after an exchange. Similarly, for the BEXSLP2, it must hold that

$$\sum_{t' \in \mathcal{T}^{\text{ch}}(t)} \sum_{q \in Q} \sum_{i \in \mathcal{I}} i \cdot a_{qli}^{t'} \leq s_l^{\text{ini}} x_l + s^{\text{modul}} y_l \quad \forall l \in L,\ t \in \mathcal{T}_l^{\text{ex}} \tag{4.3}$$

The goal of the BEXSLP2 is to minimize three different objectives. The first objective is to minimize the setup costs for stations and their corresponding BEX modules, i.e.,

$$\sum_{l \in L} (c_l x_l + c_l^{\text{modul}} y_l). \tag{4.4}$$

The second objective is to minimize the total charging costs. For this purpose let c_{lt}^{ch} refer to the costs for charging a battery at station $l \in L$ during time interval $t \in \mathcal{T}$, i.e.,

$$c_{lt}^{\text{ch}} = \begin{cases} c_l^{\text{dch}} & \text{for } t \in \mathcal{T}^{\text{dch}}, \\ c_l^{\text{nch}} & \text{else.} \end{cases} \tag{4.5}$$

The total cost of fully charging a battery c_{lt}^{chret} at location l starting with interval $t \in \mathcal{T}$ is therefore

$$c_{lt}^{\text{chret}} = \sum_{t' \in \mathcal{T}^{\text{ch}}(t)} c_{lt'}^{\text{ch}}. \tag{4.6}$$

Then, considering the assignment variables a_{qli}^t over all locations, O/D pairs, vehicle types, and opening times, the total charging costs are

$$\sum_{l \in L} \sum_{q \in Q} \sum_{i \in \mathcal{I}} \sum_{t \in \mathcal{T}_l^{\text{ex}}} c_{lt}^{\text{chret}} \cdot i \cdot a_{qli}^t. \tag{4.7}$$

Finally, besides minimizing the station setup and battery charging costs, our last objective is also to minimize the total travel delay induced by the detours for charging; thus, the sum of the differences in travel times between the routes with the charging at the assigned stations and the corresponding direct routes, calculated by

$$\sum_{l\in L}\sum_{q\in Q}\left(\tilde{w}_q^l - w_q\right)\cdot\sum_{t\in\mathcal{T}^{\text{ex}}}\sum_{i\in\mathcal{I}} a_{qli}^t. \tag{4.8}$$

We linearly combine the different objectives with weights $\alpha_{\text{setup}} > 0$, $\alpha_{\text{charging}} > 0$ and $\alpha_{\text{delay}} > 0$ to obtain the total objective function.

In summary, we express BEXSLP2 by the following MILP.

$$\begin{aligned}
\min\ & \alpha_{\text{setup}}\sum_{l\in L}(c_l x_l + c_l^{\text{modul}} y_l) + \\
& \alpha_{\text{charging}}\sum_{l\in L}\sum_{q\in Q}\sum_{i\in\mathcal{I}}\sum_{t\in\mathcal{T}_l^{\text{ex}}} c_{lt}^{\text{chret}}\cdot i\cdot a_{qli}^t + \\
& \alpha_{\text{delay}}\sum_{l\in L}\sum_{q\in Q}\left(\tilde{w}_q^l - w_q\right)\cdot\sum_{t\in\mathcal{T}_l^{\text{ex}}}\sum_{i\in\mathcal{I}} a_{qli}^t
\end{aligned} \tag{4.9}$$

$$e_l^{\max}\cdot x_l \geq y_l \qquad \forall l\in L \tag{4.10}$$

$$\sum_{l\in L|t\in\mathcal{T}_l^{\text{ex}}} a_{qli}^t \leq d_{qi}^t \qquad \forall t\in\mathcal{T},\ i\in\mathcal{I},\ q\in Q \tag{4.11}$$

$$\sum_{t'\in\mathcal{T}^{\text{ch}}(t)}\sum_{q\in Q}\sum_{i\in\mathcal{I}} i\cdot a_{qli}^{t'} \leq s_l^{\text{ini}} x_l + s^{\text{modul}} y_l \qquad \forall l\in L,\ t\in\mathcal{T}_l^{\text{ex}} \tag{4.12}$$

$$\sum_{q\in Q}\sum_{l\in L}\sum_{t\in\mathcal{T}_l^{\text{ex}}}\sum_{i\in\mathcal{I}} i\cdot a_{qli}^t \geq d_{\text{sat}} \tag{4.13}$$

$$\sum_{l\in L|c_l>0} x_l + \sum_{l\in L} y_l \leq z^{\text{modules}} \tag{4.14}$$

$$x_l\in\{0,1\} \qquad \forall l\in L \tag{4.15}$$

$$y_l\in\{0,\ldots,e_l^{\max}\} \qquad \forall l\in L \tag{4.16}$$

$$0\leq a_{qli}^t \leq \min\left(\frac{s_l^{\text{ini}} + e_l^{\max}\cdot s^{\text{modul}}}{i}, d_{qi}^t\right) \qquad \forall l\in L,\ t\in\mathcal{T}_l^{\text{ex}},\ i\in\mathcal{I},\ q\in Q \tag{4.17}$$

The objective function (4.9) minimizes the total setup costs, the total charging costs, as well as the total detours of customers as defined by Equations 4.4, 4.7, and 4.8. Inequalities (4.10) link variables x_l and y_l and correspond to (4.2). Constraints (4.11) enforce that the total demand assigned from an O/D pair q to locations does not exceed d_{qi}^t during all periods. Inequalities (4.12) ensure that the required battery modules are available at all locations over all periods. The minimal satisfied demand to be fulfilled over all time intervals is expressed by inequality (4.13). Similarly, Constraint (4.14) restricts the number of BEX modules available. Finally, the domains of the variables are given in (4.15)–(4.17).

Theorem 4.0.1. *The BEXSLP2 is $\mathcal{NP}$-hard.*

Proof. $\mathcal{NP}$-hardness of the BEXSLP2 is proven by providing a reduction from the well-known capacitated facility location problem (CLP) known to be $\mathcal{NP}$-hard.

Let Λ be a set of all facilities and M be the set of customers. Further, let γ_λ denote the fixed cost of opening facility $\lambda \in \Lambda$ and let $\omega_{\lambda\mu}$ refer to the costs of shipping a product from facility $\lambda \in \Lambda$ to customer $\mu \in M$. Moreover, δ_μ denotes the demand of customer $\mu \in M$. Let o_λ refer to the capacity of a facility $\lambda \in \Lambda$. Let binary variables ψ_λ indicate whether a facility $\lambda \in \Lambda$ is opened or not, and let variables $\beta_{\lambda\mu}$ refer to the fraction of demand of customers μ assigned to a facility λ. The goal is to minimize the overall costs. Then, a linear model for the CLP is formulated as follows.

$$\min \sum_{\lambda\in\Lambda}\sum_{\mu\in M} \omega_{\lambda\mu}\delta_\mu\beta_{\lambda\mu} + \sum_{\lambda\in\Lambda} \gamma_\lambda\psi_\lambda \tag{4.18}$$

$$\text{s.t.} \sum_{\lambda\in\Lambda} \beta_{\lambda\mu} = 1 \qquad \forall\mu \in M \tag{4.19}$$

$$\sum_{\mu\in M} \delta_\mu\beta_{\lambda\mu} \leq o_\lambda\psi_\lambda \qquad \forall\lambda \in \Lambda \tag{4.20}$$

$$\beta_{\lambda\mu} \geq 0 \qquad \forall\lambda \in \Lambda, \forall\mu \in M \tag{4.21}$$

$$\psi_\lambda \in \{0,1\} \qquad \forall\lambda \in \Lambda \tag{4.22}$$

Constraint 4.18 is the objective function of the CLP, which minimizes both the total setup cost and the cost of shipping the product. Equation 4.19 states that all customer demand must be fulfilled, whereby any combination of facilities can fulfill the demand. Constraint 4.20 ensures that the capacities of a facility are not exceeded. Finally, the domains of the variables are given with 4.21 and 4.22.

Given an instance to the CLP, we construct a corresponding BEXSLP2 instance in which the set of locations V corresponds to the set of facilities Λ and the set of O/D-pairs Q corresponds to the set of customers M. The constructed BEXSLP2 instance only has a single time interval $\mathcal{T} = \{1\}$, and depleted batteries are immediately available in the next interval, i.e., $t^{\mathrm{c}} = 0$. Additionally, there is only a single battery type $I = \{1\}$. Moreover, $w_q = 0$ for all $q \in Q$ and $\tilde{w}_q^l$ corresponds to the associated shipping costs $\omega_{\lambda\mu}$ with $\lambda = l, \mu = q$. The demand d_{ql}^1 of an O/D-pair $q \in Q$ corresponds to δ_μ with $\mu = q$. The fixed costs c_l for opening a station l with some initial configuration are γ_λ. The initial configuration of a station l is chosen according to o_λ with $l = \lambda$. Besides the first, no further modules can be added to a station. There are no restrictions on the number of stations that can be opened. Costs for charging batteries c_{l1}^{ch} are zero for all stations $l \in L$. Consequently, $\alpha_{\text{charging}} = 0$, while $\alpha_{\text{setup}} = \alpha_{\text{delay}} = 1$. Finally, we require all demands to be satisfied by a station, i.e., $d_{\text{sat}} = 1$.

Let (x, y, a) be a feasible solution to this derived BEXSLP2 instance. Note that $y_l = 0$ for all $l \in L$. A corresponding feasible solution to the CLP (ψ, β) can be derived by

- opening all locations $\lambda \in \Lambda$ for which $x_l = 1$ with $l = \lambda$
- and by assigning a^1_{ql1} demand of customer μ to facility λ with $l = \lambda, q = \mu$.

Constraint (4.12) ensures that the capacities of a facility are not exceeded, and Constraint (4.13) ensures that all demand is satisfied. Therefore, the resulting solution is a feasible CLP solution. Additionally, the objective value of the solution (x, y, a) is also the objective value of the corresponding CLP solution (ψ, β). Since all applied transformations require polynomial time, the BEXSLP2 is $\mathcal{NP}$-hard.

□

CHAPTER 5

Iterated Greedy Heuristic

In this section, we present an approach for constructing a solution to the BEXSLP2 using an iterated greedy heuristic. Starting from an initial solution, an iterated greedy algorithm consists of two alternate phases. In the first phase, a solution will be partially destroyed, while the heuristic will try to repair the partial solution in the second phase. Typically, the repair operation is used to construct the initial solution. Nevertheless, the initial solution can also be created using a different greedy heuristic. Moreover, each repair phase consists of multiple iterations in our case, with each iteration considering only a subset of the problem and the already partially constructed solution. Our repair heuristic uses a very similar formulation to the original MILP. The main change is that the formulation now focuses on solving the MILP for selected time intervals which are a subset of $\mathcal{T}$, and each iteration builds upon the solution of the previous iteration. After constructing an initial solution, the destruction and repair steps alternate in which we destroy parts of the solution (i.e., stations, modules, and the assigned demand) and then repair the partial solution in the following step. The best existing solution is used as a starting point for the subsequent destruction operation. To construct the initial solution, we use the same procedures that we also use to repair a solution.

5.1 Repair & Construction Heuristic

We construct a solution iteratively over $\mathcal{T}$ considering only the customer demand of a subset of time intervals (and unassigned demand from previous intervals) at each iteration. Let $(\hat{x}, \hat{y}, \hat{a})$ refer to a partial solution concerning a set of time intervals $\mathcal{T}' \subseteq \mathcal{T}$ or a partial solution created by the destroy and repair cycle. Furthermore, let $\tau \subseteq \mathcal{T}$ be the time intervals selected in this iteration. No element of τ is yet in $\mathcal{T}'$. After each execution of the MILP, the elements of τ will be added to $\mathcal{T}'$.

Moreover, let

$$d_{\text{sat}}[\tau] = \sum_{q \in Q} \sum_{t \in \tau} \sum_{i \in \mathcal{I}} i \cdot d_{qi}^t. \tag{5.1}$$

We solve the following MILP multiple times for the new time intervals $\tau \subseteq \mathcal{T} \setminus \mathcal{T}'$:

$$\begin{aligned}
\min\ & \alpha_{\text{setup}} \sum_{l \in L} (c_l x_l + c_l^{\text{modul}} y_l) + \\
& \alpha_{\text{charging}} \sum_{l \in L} \sum_{q \in Q} \sum_{i \in \mathcal{I}} \sum_{t \in \mathcal{T}_l^{\text{ex}}} c_{lt}^{\text{chret}} \cdot i \cdot a_{qli}^t + \\
& \alpha_{\text{delay}} \sum_{l \in L} \sum_{q \in Q} \left(\tilde{w}_q^l - w_q \right) \cdot \sum_{t \in \mathcal{T}_l^{\text{ex}}} \sum_{i \in \mathcal{I}} a_{qli}^t
\end{aligned} \tag{5.2}$$

$$e_l^{\max} \cdot x_l \geq y_l \qquad \forall l \in L \tag{5.3}$$

$$\sum_{l \in L | t \in \mathcal{T}_l^{\text{ex}}} a_{qli}^t \leq d_{qi}^t \qquad \forall t \in \tau,\ i \in \mathcal{I},\ q \in Q \tag{5.4}$$

$$\sum_{t' \in \mathcal{T}^{\text{ch}}(t)} \sum_{q \in Q} \sum_{i \in \mathcal{I}} i \cdot a_{qli}^{t'} \leq s_l^{\text{ini}} x_l + s^{\text{modul}} y_l \qquad \forall l \in L,\ t \in \mathcal{T}_l^{\text{ex}} \tag{5.5}$$

$$\sum_{q \in Q} \sum_{l \in L} \sum_{t \in \mathcal{T}_l^{\text{ex}} \cap \tau} \sum_{i \in \mathcal{I}} i \cdot a_{qli}^t \geq d_{\text{sat}}[\tau] \tag{5.6}$$

$$\sum_{l \in L | c_l > 0} x_l + \sum_{l \in L} y_l \leq z^{\text{modules}} \tag{5.7}$$

$$x_l \in \{\hat{x}_l, 1\} \qquad \forall l \in L \tag{5.8}$$

$$y_l \in \{\hat{y}_l, \ldots, e_l^{\max}\} \qquad \forall l \in L \tag{5.9}$$

$$\hat{a}_{qli}^t \leq a_{qli}^t \leq \min\left(\frac{s_l^{\text{ini}} + e_l^{\max} \cdot s^{\text{modul}}}{i}, d_{qi}^t \right) \qquad \forall l \in L,\ t \in \tau,\ i \in \mathcal{I},\ q \in Q \tag{5.10}$$

As mentioned, the MILP is almost identical to the original BEXSLP2 MILP (4.9)–(4.17). However, already assigned demand $\hat{a}$ and previously decided location capacities $(\hat{x}, \hat{y})$ are preserved in the new solution with (5.8)–(5.10). Moreover, the MILP also aims to satisfy all demand, as does the original MILP. However, only concerning the customer demand at time intervals in τ. Note, however, that it is not guaranteed that the MILP can always extend a solution. Opening a station provides fewer BEX modules than extending a station. Sometimes opening a station can result in no feasible solution because fulfilling all demand is no longer possible.

5.1.1 Repair Strategies

This section describes the different strategies for repairing an incomplete solution. While all strategies use the MILP formulation described above, the strategies differ in the order and groups in which the time intervals are selected. One key takeaway is that,

in general, it may be advantageous to reduce the number of iterations in our iterated greedy procedure, which would have $t_{\max}$ iterations if each iteration only considered one interval. This is because some overhead will always remain, even when solving a single iteration is trivial. To reduce this effect, we do not consider iterations with zero associated demand. We also group different intervals in a single iteration step, referred to as buckets. We propose different approaches that can improve the quality of the solution further. Reducing the number of iterations/buckets too drastically decreases the performance because the remaining iterations are hard to solve. Therefore, the strategies below try to find a balance between reducing the number of iterations without drastically increasing the time a single iteration takes.

The following repair strategies share two parameters. The first parameter, *sz*, describes the size of a bucket, i.e., how many succeeding time intervals are grouped in a single iteration. The second parameter, *co*, indicates after how many time intervals the procedure is cut off, and the remaining instance is solved by combining all remaining timestamps into one iteration. The idea of this parameter is that the remaining time intervals barely impact the previously constructed solution and are less challenging to solve. This results from the non-uniformly distributed demand of the different intervals. Therefore, after a certain number of iterations of the repair procedure, solving the remaining time intervals in a single iteration is more efficient. To simplify the notation, whenever we use a cutoff, it is a cutoff of 12 iterations. Consequently, the name will be simplified to, e.g., *sz=1*. The sequence in which the buckets will be selected is again determined pseudo-randomly using tournament selection. The buckets will be selected by the demand associated with the time frame, with more significant demand being more likely to be picked first.

Strategy *sz=1*. This is the baseline strategy. We rebuild the solution one time period at a time until all time periods with demand associated with them have been considered.

Strategy *sz=tc*. A unique variation of the first strategy is if the bucket size is $t_c + 1$ intervals. This strategy then considers an entire charging cycle, i.e., the time it takes to charge a battery fully. Therefore, the demand associated with a $t \in \mathcal{T}$ is the demand associated with $\mathcal{T}^{\text{ch}}(t)$, which contains an entire loading cycle. The demand for each t can then be determined by

$$\sum_{q \in Q} \sum_{i \in \mathcal{I}} \sum_{t' \in \mathcal{T}^{\text{ch}}(t)} i \cdot d_{qi}^{t'} \qquad \forall t \in \mathcal{T} \tag{5.11}$$

The elements in $\mathcal{T}^{\text{ch}}(t)$ are removed from the list of available elements and are no longer considered in the calculation for the associated demand. The order in which the intervals are selected is determined using tournament selection introduced in Chapter 2.2.5.

Strategy *sz=dyn*. Another unique case is *sz=dyn*. In this variation, the size of the bucket is calculated dynamically. Each bucket should hold the same amount of demand.

The charging cycle with the most associated demand determines the number of buckets. The charging cycle with the most demand associated with it is calculated by

$$t^{\text{hd}} = argmax_{t \in T} \left(\sum_{q \in Q} \sum_{i \in \mathcal{I}} \sum_{t \in \mathcal{T}^{\text{ch}}(t)} i \cdot d_{qi}^{t} \right) \qquad \forall t \in \mathcal{T} \tag{5.12}$$

which will be called t^{hd}. We use the demand in this time interval as the upper limit of how much demand a single bucket should contain. Consequently, we determine the number of buckets we want to create by

$$\lceil \frac{\sum_{q \in Q} \sum_{i \in \mathcal{I}} \sum_{t \in \mathcal{T}} i \cdot d_{qi}^{t}}{\sum_{q \in Q} \sum_{i \in \mathcal{I}} \sum_{t \in \mathcal{T}_{\text{hd}}^{\text{ch}}} i \cdot d_{qi}^{t}} \rceil \tag{5.13}$$

We then use the same idea already used in the other strategies to determine the order of the intervals with a slight variation. Utilizing tournament selection, we determine a charging cycle to fill the bucket. Additionally, if space remains in the bucket, we add adjacent time intervals to the already selected intervals, i.e., at the start intervals, $t-1$ and $t+t_c+1$ are considered. Since $\mathcal{T}$ is a cyclical collection of intervals, we consider the first and last interval $t_{\max}$ neighbors. Moreover, the interval with more demand is always chosen. The bucket is full if either there is no eligible interval (no adjacent time interval which is in $\mathcal{T}$ but not already selected) or the demand is equal to or more than the allowed amount. For the special case *sz=dyn*, no cutoff parameter is needed because the bucket's fixed size works very similarly to a cutoff.

Using the abovementioned parameters, we concluded after initial testing that the variants *sz=1*, *sz=tc*, and *sz=dyn* are promising. If a cutoff is used (i.e., strategies *sz=1* and *sz=tc*), we use a cutoff of 12 because we neither want the last bucket to be too big (and consequently very performance intensive) nor too small.

5.2 Destroy Strategies

In this section, we discuss strategies to destroy a solution partially. We focus on strategies that destroy a station, i.e., setting the corresponding x, y, and a variables to zero. This allows the MILP to choose a different set of locations than the destroyed one.

Destroying a time period t or certain a variables has the disadvantage that removing all associated demand to a station is improbable. Therefore, it seldom happens that a station can be closed completely so that a new (different) station can be built instead.

Given a solution (x, y, a) to the BEXSLP2, let $L(x) \subseteq L$ be the set of all locations at which a station is built.

Strategy $rloc = X$**.** From the set $L(x)$, we randomly destroy $\lceil X \cdot |L(x)| \rceil$ stations.

Strategy $wloc = X$**.** In this strategy, instead of destroying stations entirely at random, we destroy them in a random weighted fashion, weighted by the induced objective cost divided by the amount of demand assigned to the station. Again, as already described in Chapter 2.2.5, we use tournament selection to determine which stations will be destroyed. To determine the ranking of the stations, we calculate for each station $l \in L(x)$ the induced objective value per unit of assigned demand using the following equations:

$$o_l^{\text{setup}} = \alpha_{\text{setup}}(c_l x_l + c_l^{\text{modul}} y_l) \tag{5.14}$$

$$o_l^{\text{charging}} = \alpha_{\text{charging}} \sum_{q \in Q} \sum_{i\mathcal{I}} \sum_{t \in \mathcal{T}_l^{\text{ex}}} c_{lt}^{\text{ch}} \cdot i \cdot a_{qli}^t \tag{5.15}$$

$$o_l^{\text{delay}} = \alpha_{\text{delay}} \sum_{q \in Q} \left(\tilde{w}_q^l - w_q \right) \cdot \sum_{t \in \mathcal{T}_l^{\text{ex}}} \sum_{i \in \mathcal{I}} a_{qli}^t \tag{5.16}$$

$$o_l^{\text{demand}} = \sum_{q \in Q} \sum_{i \in \mathcal{I}} \sum_{t \in \mathcal{T}_l^{\text{ex}}} a_{qli}^t \tag{5.17}$$

$$o_l^{\text{total}} = \frac{o_l^{\text{setup}} + o_l^{\text{charging}} + o_l^{\text{delay}}}{o_l^{\text{demand}}} \tag{5.18}$$

We then use the o_l^{total} to sort the list, with the highest value being the first element. We apply this procedure until $\lceil X \cdot |L(x)| \rceil$ are destroyed.

5.2.1 Destroying additional demand

This section is inspired by the idea that destroying only stations alone will likely result in the same station being built again in the subsequent repair iteration. Therefore, destroying some portion of the remaining demand is advantageous. We use this method in combination with the strategies mentioned above. We denote the modified version by adding a prime to the strategy, i.e., $wloc'$.

Additionally, to the already destroyed stations, we select the demand which contributes the most to the objective function. However, defining which a variables are responsible for the construction cost is not trivial. One idea is to split the construction cost equally between all a variables assigned to a station. This, however, is not true since if the station is already built to accommodate a demand peak, then assigning demand in downtime has virtually no cost besides the detour and the charging cost. Theoretically, the most relevant variables are those at the peak periods (of a given station). If there are multiple

periods that force the station to be of an actually chosen size, they are all equally at fault. This, however, will likely lead to all a variables at the peak periods being very costly. Consequently, we decided that no explanation considers the construction cost and, simultaneously, weighs a variables fairly. Therefore, we use only the charging and delay part of the objective function and ignore the construction cost. We, thus, calculate for each a variable:

$$o^{\mathrm{a}}_{ltiq} = \alpha_{\mathrm{charging}} \cdot c^{\mathrm{chret}}_{lt} \cdot i \cdot a^t_{qli} + \alpha_{\mathrm{delay}} \cdot \left(\tilde{w}^l_q - w_q\right) \cdot a^t_{qli} \qquad \forall l \in L,\ t \in \mathcal{T}^{\mathrm{ex}}_l,\ i \in \mathcal{I},\ q \in Q \tag{5.19}$$

We then sort the resulting list by associated weight and remove those a variables pseudo-randomly. To do this, we again use tournament selection to select those a variables which cause the most weight until the specified amount of demand has been destroyed. We decided always to destroy half of the given percentage of the total demand, which is illustrated by the following equation:

$$\lceil \frac{X}{2} \cdot \sum_{q \in Q} \sum_{i \in \mathcal{I}} \sum_{t \in \mathcal{T}} i \cdot d^t_{qi} \rceil \tag{5.20}$$

It should be noted that the destruction of stations occurs before removing the additional demand. Therefore, stations selected to be destroyed already have corresponding a variables of zero. Consequently, the additional demand is only destroyed from not destroyed stations.

CHAPTER 6

Instances

In this chapter, we describe the instances used to compare the algorithms. First, we describe how we converted a BEXSLP instance given to us by Honda R&D to a BEXSLP2 instance. In total, we received three completely different instances and a unique variation of one instance containing street network data. Afterward, a test instance creator is briefly described that Rauscher [Rau22] developed in his thesis regarding the same problem. The instance set he created contains six different groups with increasingly more O/D pairs and stations. Each group contains 30 instances. In total, 180 instances are included in this instance set.

6.1 Honda Instances

Test instances are derived from given BEXSLP instances. In the BEXSLP, an underlying graph G_{BEXSLP} represents a geographical region partitioned into equally sized areas, where $V(G_{\text{BEXSLP}})$ corresponds to the set of areas and $E(G_{\text{BEXSLP}})$ contains a directed edge for each pair of geographically adjacent areas. Additionally, for each edge in $E(G_{\text{BEXSLP}})$, the distance, the travel time $t_{\text{travel}} : E \to \mathbb{R}^+$, as well as the required power consumption (kWh) for traveling between the areas is specified. Note that the size of the area affects how close the given routes/stations represent reality.

From G_{BEXSLP} we derive a graph $G = (V, A, w)$ with

- $V = V(G_{\text{BEXSLP}})$
- $A = \bigcup_{(v_1,v_2)\in E(G_{\text{BEXSLP}})}\{(v_1, v_2), (v_2, v_1)\}$
- $w((v_1, v_2)) = t_{\text{travel}}(\{v_1, v_2\})$ for all $(v_1, v_2) \in A$

This graph calculates the distance $w_q \geq 0$, the expected travel time when taking the most direct route without exchanging batteries. Additionally, we calculate $\tilde{w}_q^l$, the expected

travel time when making the fastest possible detour to location $l \in L$ for exchanging batteries there. As a result, all instances contain only the distances for any possible detour.

Stations are assumed to be built in the center of an area. For each area of the graph, we know if a station can be built in the area. If that is the case, we also know the day/night time charging times, the respective charging costs, and the costs for opening a new station and installing BEX modules.

Furthermore, for already existing stations, we know the area in which the station is built, the number of already existing BEX modules, the start and end times within which batteries can be exchanged, and the start and end times within which batteries can be charged at the station. Additionally, each BEXSLP instance has a global maximum number of BEX modules a station can have and a global maximum number of battery slots an initial/additional BEX module can have.

From the given data, we can then derive the set L of stations and associated attributes straightforwardly. However, note that the initial station configuration for non-existing stations is the sum of costs for opening a station and adding the first BEX module. Moreover, the initial station configuration costs are zero for already existing stations. As no opening times or times at which batteries can be charged are given for not yet existing stations $l \in L$, we assume that $\mathcal{T}_l^{\text{ex}} = \mathcal{T}$.

In the given BEXSLP instances, different types of scooters $\mathcal{I}$ exist, with each type $i \in \mathcal{I}$ having i batteries. Additionally, a set of users U is given. Each user $u \in U$ has an associated vehicle type and a state of charge (SOC) level given as an interval $[\theta_u^{\text{LL}}, \theta_u^{\text{UL}}]$ in which the user feels most comfortable, i.e., if possible the users SOC should always remain between the lower and upper bound.

Users have associated data points marked with a time stamp between 0 and 14400 which also contains the current state of charge (SOC) and position of the user's vehicle and has a flag indicating whether a user is leaving home or returning home. Note that the trip data describes direct trips from an origin to a destination without detours to battery-swapping stations.

We derive Q from the user trip data by extracting all trips T. Each trip has a flag that describes whether the user is leaving or returning home. Each extracted trip $\tau \in T$ is associated with

- the respective user $u(\tau)$,
- the scooter type $i(\tau)$ of the user,
- an origin $o(\tau)$,

- a destination $\rho(\tau)$,
- the state of charge $SOC^{\text{start}}(\tau)$ of the scooter at the start of the trip,
- the state of charge $SOC^{\text{end}}(\tau)$ of the scooter at the end of the trip,
- a trip time $t(\tau) = \lfloor \frac{t_{\text{start}}(\tau)+t_{\text{end}}(\tau)}{2 \cdot 60} \rfloor \bmod 24$ where $t_{\text{end}}(\tau)$ and $t_{\text{start}}(\tau)$ refer to the start and end time of τ, respectively.

The origin and destination of a trip τ are derived from the start and end of τ such that $o(\tau) \leq \rho(\tau)$. Trips τ with the same origin and destination refer to the same O/D pair $(o(\tau), \rho(\tau)) \in Q$. Moreover, the demand d_{qi}^t of an O/D pair $q = (v_1, v_2) \in Q$ at a time $t \in \mathcal{T}$ for vehicle type $i \in \mathcal{I}$ is defined as follows:

First, we calculate the total power consumption of a user $u \in U$ over all of his trips, i.e.

$$p_u = \sum_{\tau \in T | u = u(\tau)} SOC^{\text{end}}(\tau) - SOC^{\text{start}}(\tau). \tag{6.1}$$

Then, the minimum and maximum amount of battery swaps on u is given by

$$ex_u^{\min} = \frac{p_u}{100 - \theta_u^{\text{LL}}} \qquad \text{and} \qquad ex_u^{\max} = \frac{p_u}{100 - \theta_u^{\text{UL}}}, \tag{6.2}$$

respectively.

Thus, as the time horizon spans ten days, on average, a user needs to exchange batteries $ex_u^{\text{avg}} = \frac{ex_u^{\min} + ex_u^{\max}}{2 \cdot 10}$ times a day. Based on this average number of battery exchanges, the demand d_{qi}^t at time t is calculated as

$$d_{qi}^t = \sum_{\tau \in T | t = t(\tau)} ex_{u(\tau)}^{\text{avg}} \cdot \frac{SOC^{\text{end}}(\tau) - SOC^{\text{start}}(\tau)}{p_{u(\tau)}}. \tag{6.3}$$

From the data received from Honda, we calculated that $t^{\text{c}} = 2$.

Honda R&D Japan provided four instances, referred to as case1, case2, case3, and case3 street network. Table 6.1 shows the characteristics of the derived BEXSLP2 instances referred to by the same names. Note that charging costs are the same for daytime and nighttime for all stations and instances. Additionally, only the case2 instance has more than one type of vehicle. Case3 street network contains precise location data.

6.2 Artificial Instance Set (AIS)

In this section, we describe the process of creating a new artificial instance set (AIS), which will be used to evaluate the BEXSLP2. The main problem with the converted

Table 6.1: Key characteristics of the four Honda instances, with n representing the number of potential station locations, m referring to the number of O/D pairs, n_d referring to the number of non-zero demand values d_{qi}^t over all $t \in \mathcal{T} q \in Q, i \in I$, $e^{\max}$ representing the maximum number of BEX modules which can be added to a station, and z^{modules} being the maximum number of new modules which may be built.

inst.	n	m	n_d	$e^{\max}$	z^{modules}
case1	63	898	1074	5	3
case2	67	428	493	3	8
case3	67	322	407	3	8
case3 street network	70	677	801	2	8

instances is that they offer slight variations in the user data. Therefore, it was crucial to develop instances that offered more variation so that we could test our developed algorithms more reliably. As mentioned, Rauscher [Rau22] developed an instance creator that constructs various instances of different sizes. Consequently, this is only a very brief introduction. Further information can be found in his thesis. The number of station locations and O/D pairs measures the size of an instance. Six different instance groups were created of the sizes:

Table 6.2: This table provides the number of locations, O/D pairs, and the number of instances on all the existing instance groups in the AIS instance set.

group name	locations (n)	O/D pairs (m)	instances
(50 100)	50	100	30
(100 200)	100	200	30
(200 400)	200	400	30
(300 600)	300	600	30
(400 800)	400	800	30
(500 1000)	500	1000	30

For each instance group, 30 different instances were created. The idea was to recreate the settings from the BEXSLP as closely as possible while adding more variation and increasing the size. The cost of building stations and adding additional modules was decreased; however, the ratio was kept. The costs for building the station are chosen uniformly at random from $\{5000, \cdots, 7000\}$ while the cost for adding a module is between $\{2000, \cdots, 4000\}$. The various other elements of the station are slightly varied as well. O/D pairs and resulting demand are generated as well. Demand is distributed unevenly; there is a peak in the morning and the evening, as in the BEXSLP instances. In all instances, the parameter z^{modules} is chosen to create a challenge for the algorithms without being impossible to solve. At most, 3% of all possible modules can be built. Additional information and further details can be found in the work of Rauscher [Rau22]

CHAPTER 7

Results and Discussion

In this section, we evaluate our algorithms developed for solving the BEXSLP2. All algorithms are implemented in Julia[1] 1.9.3 using the JuMP package and Gurobi[2] 9.1 as underlying MILP solver. All test runs have been executed on an Intel Xeon E5-2640 v4 2.40GHz machine in single-threaded mode with a global time limit of four hours per run (if not mentioned otherwise) and depending on the instance group can use the following maximum amount of memory depicted in Table 7.1

Table 7.1: Maximum allowed memory to be used for each instance.

Instance Group	Maximum allowed memory
(50 100)	4 GB
(100 200)	4 GB
(200 400)	6 GB
(300 600)	12 GB
(400 800)	24 GB
(500 1000)	36 GB
case1-case3 & street network	36 GB

First, we evaluate the BEXSLP2 using the MILP formulation given by Equations (4.9)–(4.17). Afterward, we present the results for our iterated greedy heuristic. For both implementations, we present the results for the AIS instances and the results for the Honda instances.

[1] https://julialang.org/
[2] https://www.gurobi.com/

7.1 MILP Results

We first evaluate the MILP using the AIS instances and, afterward, discuss the results for the Honda instances.

7.1.1 Results for the Artificial Instance Set

We first examine the AIS instances using different weights for the objective function simulating different interests when evaluating the program. Changing the α_{charging} has little to no impact on both the solution's performance and quality. However, changing α_{setup} and α_{delay} has huge implications. Nevertheless, increasing the value of α_{delay} brings about identical results to reducing the α_{setup}. Therefore, we focus only on changing α_{delay}. We apply the following configurations to test the program:

1. $\alpha_{\text{setup}} = 0.01, \alpha_{\text{charging}} = 0.01, \alpha_{\text{delay}} = 0.1$
2. $\alpha_{\text{setup}} = 0.01, \alpha_{\text{charging}} = 0.01, \alpha_{\text{delay}} = 1.0$
3. $\alpha_{\text{setup}} = 0.01, \alpha_{\text{charging}} = 0.01, \alpha_{\text{delay}} = 10.0$

Since only the α_{delay} value is adapted, the different versions will be referred to as $\alpha_{\text{delay}} = 10$, $\alpha_{\text{delay}} = 1$, $\alpha_{\text{delay}} = 0.1$.

	$\alpha_{\text{delay}} = 10$		$\alpha_{\text{delay}} = 1$		$\alpha_{\text{delay}} = 0.1$	
(n m)	gap (%)	time (s)	gap (%)	time (s)	gap (%)	time (s)
(50 100)	0.0	207.5	0.0	299.2	0.0	44.7
(100 200)	0.4	2,050.4	0.1	2,214.4	0.0	440.1
(200 400)	15.0	14,415.6	14.1	14,415.6	1.1	10,644.8
(300 600)	67.6	14,425.8	52.5	14,426.1	10.1	14,426.3
(400 800)	81.8	14,444.3	57.4	14,443.9	29.4	14,443.4
(500 1000)	82.8	14,457.8	61.2	14,459.3	37.3	14,458.3

Table 7.2: This table shows the optimality gap as well as the run time for the artificially created instance set for the configurations $\alpha_{\text{delay}} = 10$, $\alpha_{\text{delay}} = 1$, $\alpha_{\text{delay}} = 0.1$

Table 7.2 shows the optimality gap as well as the run time for the artificially created instance set for the configurations $\alpha_{\text{delay}} = 10$, $\alpha_{\text{delay}} = 1$, and $\alpha_{\text{delay}} = 0.1$. As expected, the median computation times and the average gaps increase as the size of the instances increases. Moreover, in Table 7.2 for the first two instance groups, in which most of the instances could be solved to optimality, we can also see how different α weightings affect the solving time. The most challenging instances to solve (run time-wise) are those where a certain equilibrium exists between delay and construction cost. This was also noted in the earliest versions of the Honda instance set. Consequently, increasing or decreasing

α_{delay} leads to faster solving times, which can also be seen in Figure 7.1. It displays the gaps for the different instance groups and α_{delay} values. This, however, does not mean that the gap is the largest for the $\alpha_{\text{delay}} = 1$ instances, since typically, the $\alpha_{\text{delay}} = 10$ configuration has higher gap percentages.

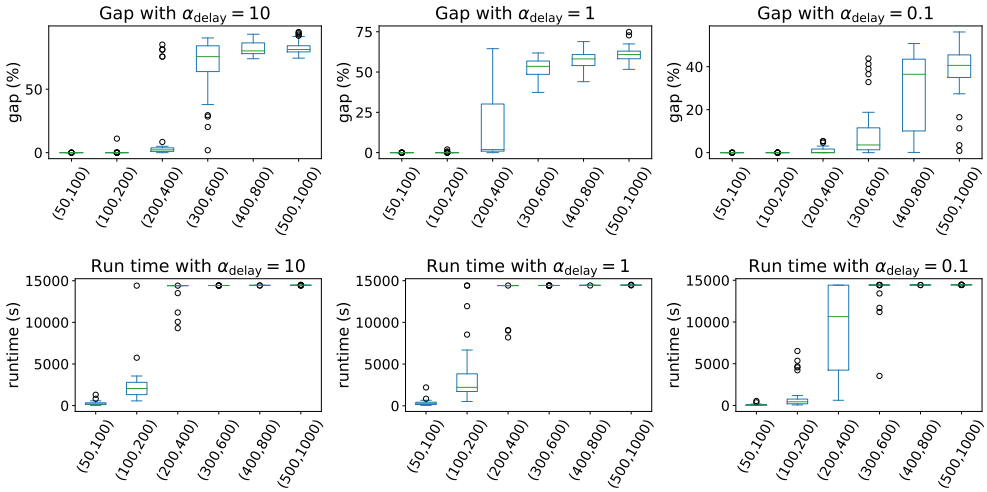


Figure 7.1: This figure displays the gaps and run times for the various instance groups and α_{delay} values.

The difference in the gap between the different α_{delay} values can be partly explained by the increase and decrease in the absolute value of the objective. The total cost for constructing stations for all configurations remains relatively stable because of the necessity to serve all customers. The base value (without the α_{delay} modifier) of the induced delay varies drastically between the three configurations depending on the weight of α_{delay}. This behavior is expected because an increase in α_{delay} renders it crucial to minimize the delay in the objective function.

A similar effect can also be seen in the next step, in which we assess the quality of the solution when only optimizing for one specific term of the objective function. For this purpose, we first evaluate the model for each objective term individually by setting each respective α value to one and the others to zero, i.e., ignoring the other optimization goals.

In Figure 7.2 the gaps for all instance groups of the AIS are shown. It becomes evident that the real issue at hand is to solve it optimally for the delay. Limiting the number of modules allowed to be built and optimizing the delay proves exceptionally challenging. Additionally, it can be seen that finding lower bounds when optimizing for delay only appears to be very challenging. Otherwise, a gap of nearly 100% cannot be explained for the largest instance group. Optimizing setup and charging costs can be done optimally within the given time in nearly all cases. On average, it takes less than 1000 seconds for the largest instances to be solved when only considering either setup or charging costs.

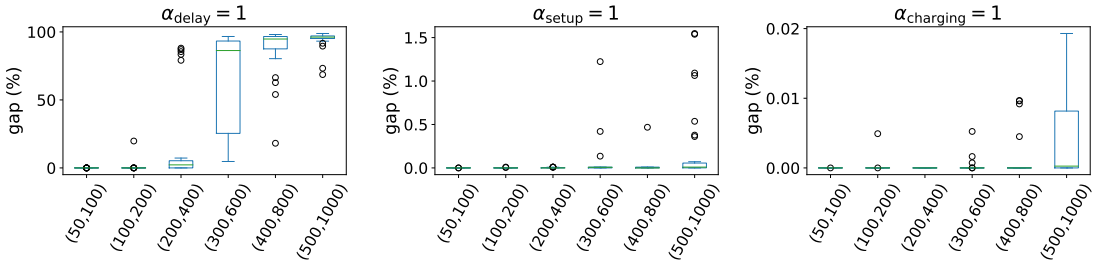


Figure 7.2: This figure displays the gaps and run times for the various instance groups when optimizing for a single objective, e.g., $\alpha_{\text{delay}} = 1$ with both α_{setup} and α_{charging} being zero.

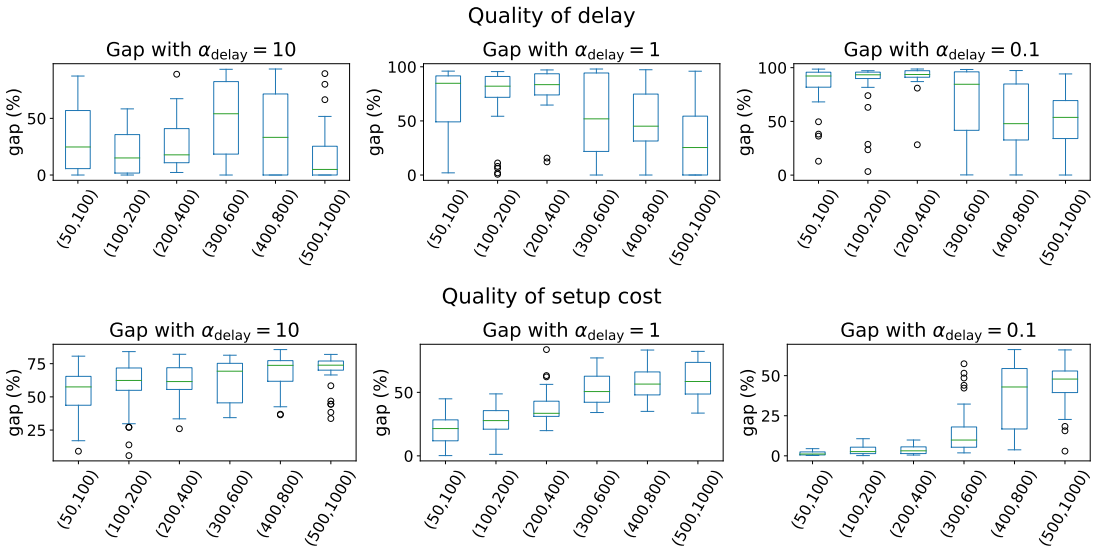


Figure 7.3: This figure displays the quality of the MILP solution compared to the optimal values of delay and setup cost.

Another salient aspect is the quality of the best-found MILP solution regarding the optimal values for delay and setup. Note that our instance sets contain some instances where the optimal delay/best-found lower bound is zero. Therefore, for these instances, we cannot calculate a gap to the respective lower bounds according to Equation 2.15 (unless the optimal solution has been found within the time limit). For these instances, we have assumed – similar to Gurobi – a gap of 100%. Some of the gaps regarding the delay shown in Figure 7.3 are slightly distorted due to this aspect.

Nevertheless, while not all instances could be solved optimally when considering only a single objective, the quality of the generated solutions is sufficient to convey an impression of the best possible values we can achieve for each objective term. The influence of

increasing or decreasing the α_{delay} value can be observed. For example, with a higher α_{delay}, the setup cost becomes less optimal while the delay value approaches the optimal value.

Results for the Honda Instance Set

For each Honda instance, Honda R&D Japan also provided the respective BEXSLP solutions with specific weights for the terms in the objective function. Therefore, we apply these solutions as references to the solutions generated by our MILP. For this purpose, however, we must convert the given BEXSLP solutions into BEXSLP2 solutions. This was done by fixing the stations and BEX modules according to the BEXSLP solutions and then applying the MILP solver to assign the demand optimally. However, it has to be noted that to generate a feasible solution to case1, it was necessary to add one more BEX module to the solution. The BEXSLP2 algorithm determined where this module must be built. Table 7.3 presents an overview of the provided BEXSLP solutions w.r.t. to the α weights used for their generation. The table also compares the number of stations and modules between the provided BEXSLP solutions and their BEXSLP2 counterpart solutions. Noteworthy is that for case2 α_{setup} and α_{charging} are zero. Therefore, the BEXSLP2 algorithm builds stations and BEX modules until the total cap of eight BEX modules is reached.

Table 7.3: Configurations used for generating the provided BEXSLP solutions. The number of stations and BEX modules between the BEXSLP solutions and their BEXSLP2 counterpart solutions are also shown.

				BEXSLP		BEXSLP2	
	α_{delay}	α_{setup}	α_{charging}	n_{stations}	n_{modules}	n_{stations}	n_{modules}
case1	50	0.0005	0.0005	22	24	23	23
case2	1	0	0	29	29	29	29
case3	1	0.0001	0.0001	22	26	27	27
case3	10	0.0001	0.0001	22	26	27	27
case3	100	0.0001	0.0001	22	26	29	29
case3	1000	0.0001	0.0001	25	29	29	29
case3	10000	0.0001	0.0001	26	29	29	29

Table 7.4 draws up a comparison between the solutions generated by our MILP, referred to as BEXSLP2, and the solutions derived from the BEXSLP solutions, referred to as BEXSLP, for case1 and case2. It presents the time required for our MILP to solve the instances, the gap of the generated solutions to the best-found lower bounds, the unweighted objective value, and the unweighted values of each objective term. Since Gurobi found the optimal solutions within the time limit for both instances, we expect the derived solutions to be worse than the generated solutions. For case1, the objective of the solution derived from the BEXSLP solution is roughly 20% worse, while in case2, the

best-found solution is roughly 46% higher than the solution of the BEXSLP2. Moreover, as previously mentioned, the charging costs are the same for all stations and times of an instance. Therefore, the generated solutions, as well as the derived solutions, have the same charging costs.

Furthermore, note that for case2, only α_{delay} is greater than zero. Therefore, the setup costs are very high. Both the BEXSLP2 and the BEXSLP solution build all eight possible modules. In case1, the delay value is better than in the BEXSLP2. This is because the provided solution is not feasible in the BEXSLP2. Therefore, another module had to be built, which resulted in increased setup costs and a better delay value.

Table 7.4: Run time, gap, weighted total objective value, and unweighted objectives (delay, setup, and charging) of solutions generated from the BEXSLP2 model and solutions derived from Honda's BEXSLP approach.

			objective	
	time (s)	gap (%)	BEXSLP2	BEXSLP
case1	5426s	0.00	161305.08	194381.99
case2	24s	0.00	28.10	41.13

	delay		setup		charging	
	BEXSLP2	BEXSLP	BEXSLP2	BEXSLP	BEXSLP2	BEXSLP
case1	76.18	71.26	3148362.00	3814818.00	1559.89	1559.89
case2	28.10	41.13	7834736.00	7834736.00	1525.89	1525.89

As we have received multiple BEXSLP solutions for case3 w.r.t. different weighting of α_{delay}, we now give a more detailed evaluation of the case3 instance. To gain further insights about the impact of α_{delay} we tested additional values for α_{delay} while maintaining α_{setup} and α_{charging} unchanged. Table 7.5 provides an overview of all tested α_{delay} values and shows how they each affect the solving time. Additionally, the table also shows a comparison of the weighted objective values between the derived solutions and their BEXSLP2 counterparts. Note that for $\alpha_{\text{delay}} \in \{1, 10\}$, it was not possible to find an optimal solution within the time limit. However, the table demonstrates that the solving time decreases as α_{delay} increases. Moreover, note that the first four configurations result in the same solution. However, the configurations require vastly different solving times. The instances appear easier to solve as the setup costs become less important.

Figure 7.4 compares the total objective values between our generated solutions and those derived from the BEXSLP solutions. The figure shows that the solutions generated by our MILP have, in general, a better objective value than the solutions derived from the BEXSLP solutions. Additionally, we see that the difference between the objective values increases with increasing values of α_{delay}. Figure 7.5 compares the objective values of the

Table 7.5: Run time, gap, and the weighted total objective value for all tested configurations for case3.

			objective	
α_{delay}	time (s)	gap (%)	BEXSLP2	BEXSLP
1	14408s	8.0%	36813.83	58936.09
10	14407s	7.1%	37714.36	60333.08
100	2061s	0.00%	46719.66	89049.58
200	38s	0.00%	56725.56	-
500	22s	0.00%	85625.47	-
1000	23s	0.00%	119276.41	185186.38
2000	18s	0.00%	170085.57	-
5000	18s	0.00%	321470.26	-
10000	17s	0.00%	573778.07	1146554.45

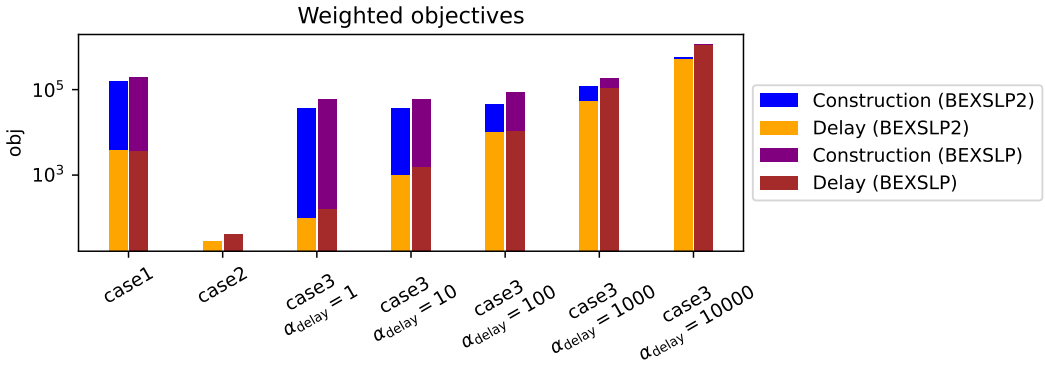


Figure 7.4: Comparison of objective values between BEXSLP and BEXSLP2 solutions.

BEXSLP and BEXSLP2 solutions. The objectives are compared by calculating the gap between the objective values of solutions derived from the BEXSLP solutions to their BEXSLP2 counterparts. For case1 and case2, we observe gaps of 20.5% and 46.37%, respectively. For case3 with $\alpha_{\text{delay}} = 1$ we already obtain a gap of 60.1% which increases as α_{delay} increases, resulting in gaps up to 99.82% for $\alpha_{\text{delay}} = 10000$.

In the following, we examine case3 with all tested configurations in more detail. Figure 7.6 shows how the number of stations and BEX modules develops as α_{delay} increases. As expected, more stations and BEX modules are added to the solutions as α_{delay} increases due to the fact that setup costs become increasingly less important. In the instances with $\alpha_{delay}1 - 500$, only modules are built (besides a necessary station to fulfill demand in the middle of the night). Configuration $\alpha_{\text{delay}} = 1000$ adds three stations but removes one module from an expanded station by a BEX module. This is a result of the very high delay value. For the configurations $\alpha_{\text{delay}} = 2000$, $\alpha_{\text{delay}} = 5000$ and $\alpha_{\text{delay}} = 10000$

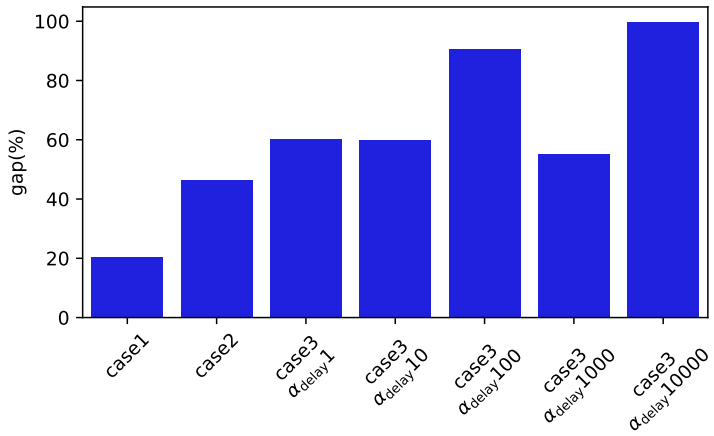


Figure 7.5: Gaps between BEXSLP and BEXSLP2 solutions. The gaps show how much worse a solution derived from a BEXSLP solution is, compared to its BEXSLP2 counterpart.

another station was opened. All configurations with a $\alpha_{\text{delay}} >= 1000$ use the maximum number of modules that the algorithm is allowed to construct.

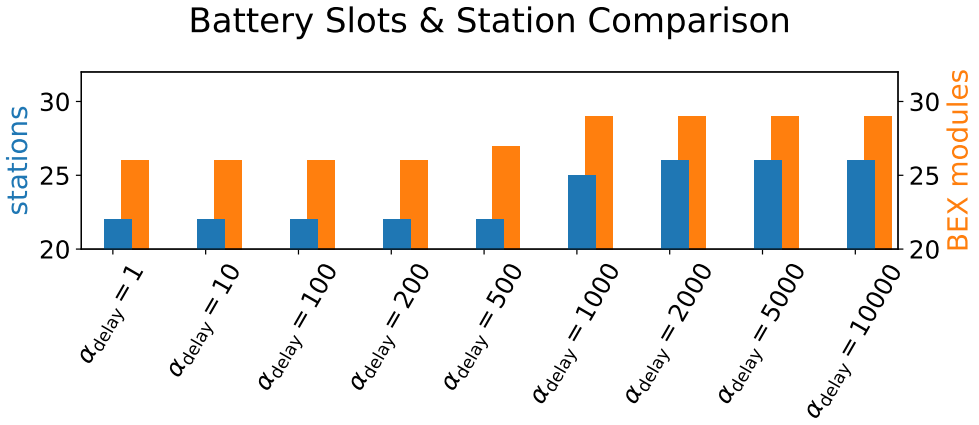


Figure 7.6: number of stations and BEX modules to be built for the different configurations.

Figure 7.7 shows a visualization of solutions to case3 regarding the different configurations. Additionally, it is easy to spot that gradually, more and more stations are added to the solution as α_{delay} increases.

Next, we evaluate the quality of the individual terms of the objective function. Similar to what was done for the AIS instances, we first calculate the optimal value for each

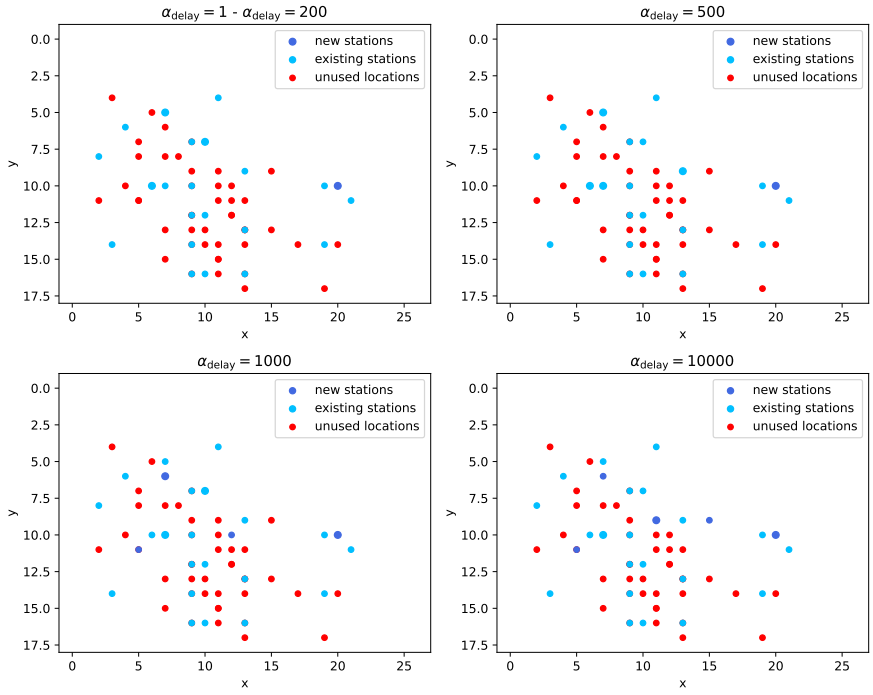


Figure 7.7: Geographic station placement for the different configurations of case3.

term by setting all other weights to zero. It should be mentioned that for case3, the optimal delay is not zero; therefore, we calculate gaps for the delay in the same way as for the setup and charging costs. Figure 7.8 shows the quality of the different unweighted terms of the objective function for the generated and derived solutions. As previously mentioned, all stations have the same charging costs. Thus, the charging costs are always the same and consequently optimal. The figure demonstrates that for the configurations $\alpha_{\text{delay}} \in \{1, 10, 100, 200\}$, the generated MILP solutions have optimal construction costs. Only for larger values of α_{delay} construction costs become increasingly less important. Eventually, the optimal delay value is reached for $\alpha_{\text{delay}} = 10000$. We also notice that the delay improves for the derived solutions with increasing α_{delay}; however, not as much as the solutions generated by our MILP.

Finally, in Figure 7.9, we compare the weighted objectives between our generated solutions and those derived from the BEXSLP solutions. The figure shows the weighted objective values for each configuration and how much each term individually contributes to the total objective. The delay becomes increasingly essential for both groups while the setup costs stay roughly the same for all configurations. However, our MILP seems better at managing increasing α_{delay} values. It should be noted, though, that our comparison requires converting BEXSLP solutions into BEXSLP2 solutions and, thus, naturally favors generated BEXSLP2 solutions over derived BEXSLP solutions. Therefore, our generated solutions are expected to feature better results in this context.

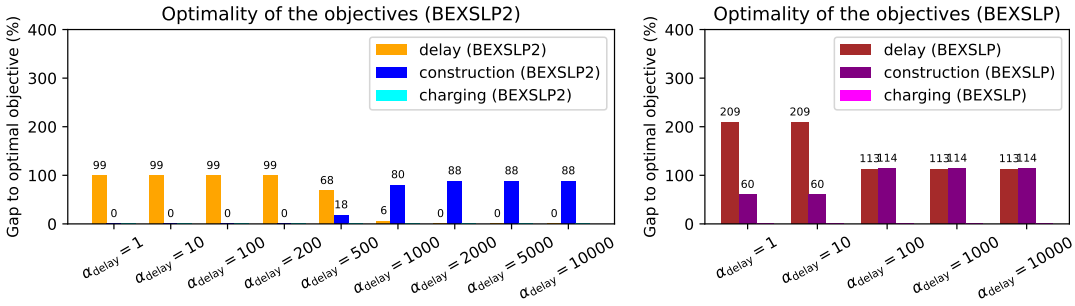


Figure 7.8: Optimality of the different objectives for both the BEXSLP2 solution and the recreated Honda solutions.

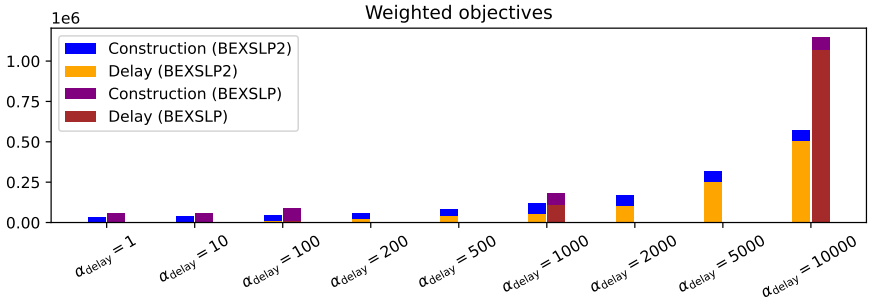


Figure 7.9: Weighted objectives of the BEXSLP2 and the recreated Honda solutions.

Case3 Street Network

Honda R&D Japan also provided a modified case3 in which the mentioned grid was no longer used. Instead, distances for $\tilde{w}_q^l$ and w_q are calculated via a more detailed street network. Table 7.6 provides an overview of the various configurations' runtime, gap, and objectives. Again, as was done for case3, we use the same configurations with increasing α_{delay}. One thing that can be noticed immediately is that the street network's runtime is lower than the grid's. This, however, does not consider the time to create the instance that is likely to be higher for the street network.

Also, the delay value is consistently higher than in the grid version of case3. The charging values are lower. This results from the conversion process, where demand is assigned depending on the battery consumed. In the street network, the energy consumption is lower; consequently, the instance converter assigns a little less demand to the trips. This also brings about another implication which can be seen when comparing the setup costs. While for the runs with a smaller α_{delay} the setup cost is lower compared to the grid network in the runs with $\alpha_{\text{delay}} = 2000$, $\alpha_{\text{delay}} = 5000$ and $\alpha_{\text{delay}} = 10000$ it is higher. This can also be seen in Figure 7.10. We identify that fewer modules have to be built for smaller α_{delay} values. While for $\alpha_{\text{delay}} = 2000$ and above, more stations are built,

Table 7.6: Run time, gap, and the weighted objectives for all tested configurations for case3 utilizing the street network.

			objectives			
α_{delay}	time (s)	gap (%)	delay	setup	charging	total
1	362.46	0.00	271.20	16518.42	19.48	16809.10
10	361.62	0.00	2712.01	16518.42	19.48	19249.91
100	28.85	0.00	27120.10	16518.42	19.48	43658.00
200	9.25	0.00	54240.21	16518.42	19.48	70778.11
500	10.18	0.00	103471.95	33036.84	19.48	136528.26
1000	6.62	0.00	166578.32	62417.10	19.48	229014.90
2000	6.31	0.00	320233.10	72210.52	19.48	392463.11
5000	14.14	0.00	793863.32	75278.94	19.48	869161.74
10000	17.86	0.00	1587726.64	75278.94	19.48	1663025.06

increasing construction cost.

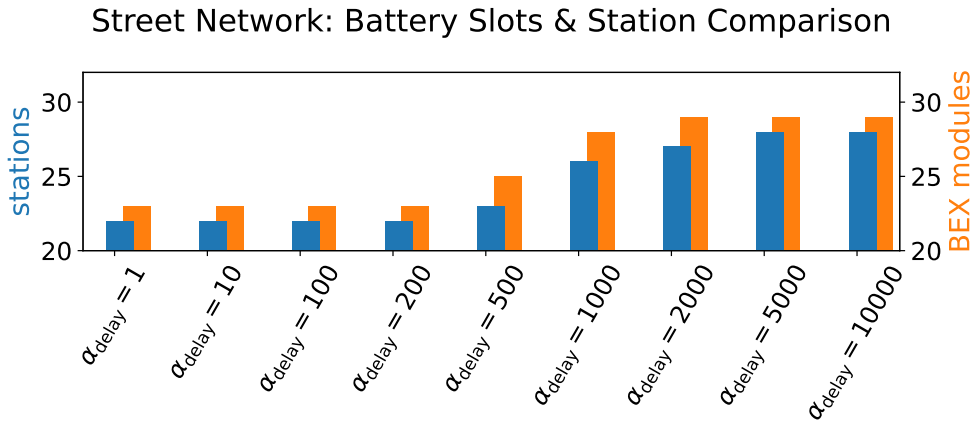


Figure 7.10: number of stations and BEX modules to be built for case3 using the street network.

In Figure 7.11 a trend similar to the one in the regular case3 instance can be observed. While the setup cost is optimal initially, increasing α_{delay} values shifts the priority. Consequently, additional BEX modules must be installed for $\alpha_{\text{delay}} = 500$ and higher. This results in the delay value approaching optimality and the construction cost becoming increasingly less optimal.

Nevertheless, a direct comparison to the case3 grid case is not trivial. While the data describes the same case in both instances, some inaccuracies inevitably result from the conversion to the grid. Figure 7.12 represents the station placement of the street network where stations are projected on a city map. When comparing these results to case3, one

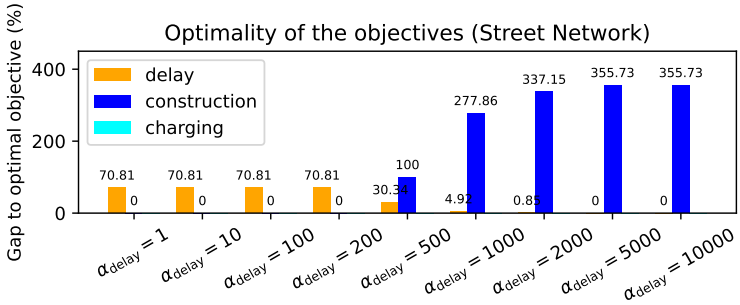


Figure 7.11: Optimality of the different objectives for the various configurations of case3 with the street network.

can identify that the positions of the stations are derived from the same data.

7.2 Iterated Greedy Results

This section discusses the results for both the AIS instances and the Honda instances for the iterated greedy heuristic.

7.2.1 Results for the Artificial Instance Set

This section discusses the performance of the previously presented iterated greedy heuristic on the artificial instance set. First, we look at the results of the different construction heuristics. Afterward, we evaluate the different destroy and repair operations individually to gain a better understanding of their respective performance. Finally, we combine everything and analyze the achieved results. Again, we will be using three different configurations in which we vary the α_{delay} parameter:

1. $\alpha_{\text{setup}} = 0.01, \alpha_{\text{charging}} = 0.01, \alpha_{\text{delay}} = 0.1$
2. $\alpha_{\text{setup}} = 0.01, \alpha_{\text{charging}} = 0.01, \alpha_{\text{delay}} = 1.0$
3. $\alpha_{\text{setup}} = 0.01, \alpha_{\text{charging}} = 0.01, \alpha_{\text{delay}} = 10.0$

Therefore, if not explicitly specified otherwise, it can be assumed that $\alpha_{\text{delay}} = 0.1$ and $\alpha_{\text{setup}} = 0.01$ for all shown results. We use Equation 2.15 to calculate the optimality gaps.

Construction Heuristics

In this section, we evaluate the performance of all the construction heuristics introduced in Chapter 5. The construction heuristics in question are *sz=1*, *sz=tc*, *sz=dyn* as can be seen in Figures 7.14 and 7.13, which display the gap and the run time of the different

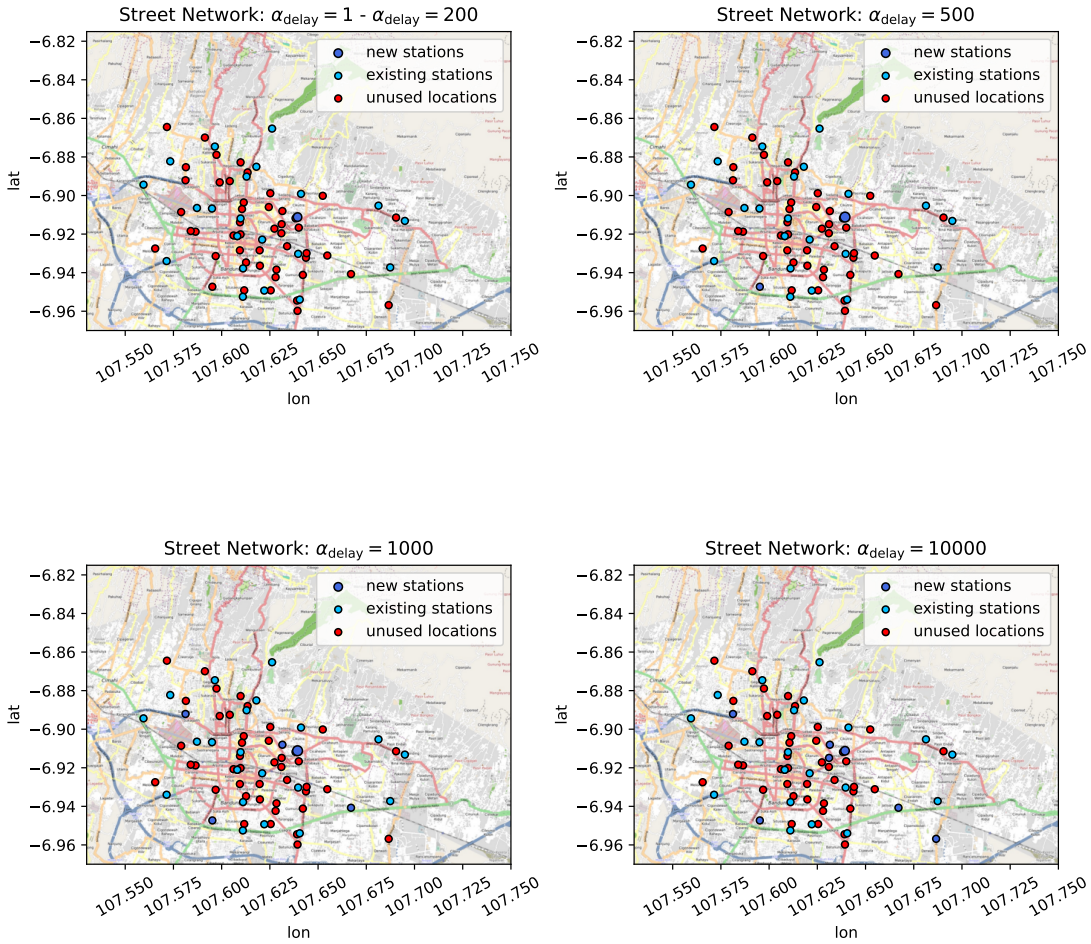


Figure 7.12: Geographic station placement for the different configurations of case3 utilizing the street network.

construction heuristics in the various instance groups and configurations. From the gathered results, one can infer there is no single best construction heuristic. For the smaller instance sizes, all construction heuristics take the same time. However, when looking at the largest instance size *(500 1000)*, we see that *sz=1* is much faster than the others. This is due to the fact that the computational requirements for the smaller instances are shallow; therefore, it can be advantageous to group iterations together to reduce some overhead. However, as soon as the iterations become increasingly more challenging, *sz=1* constantly achieves faster run times.

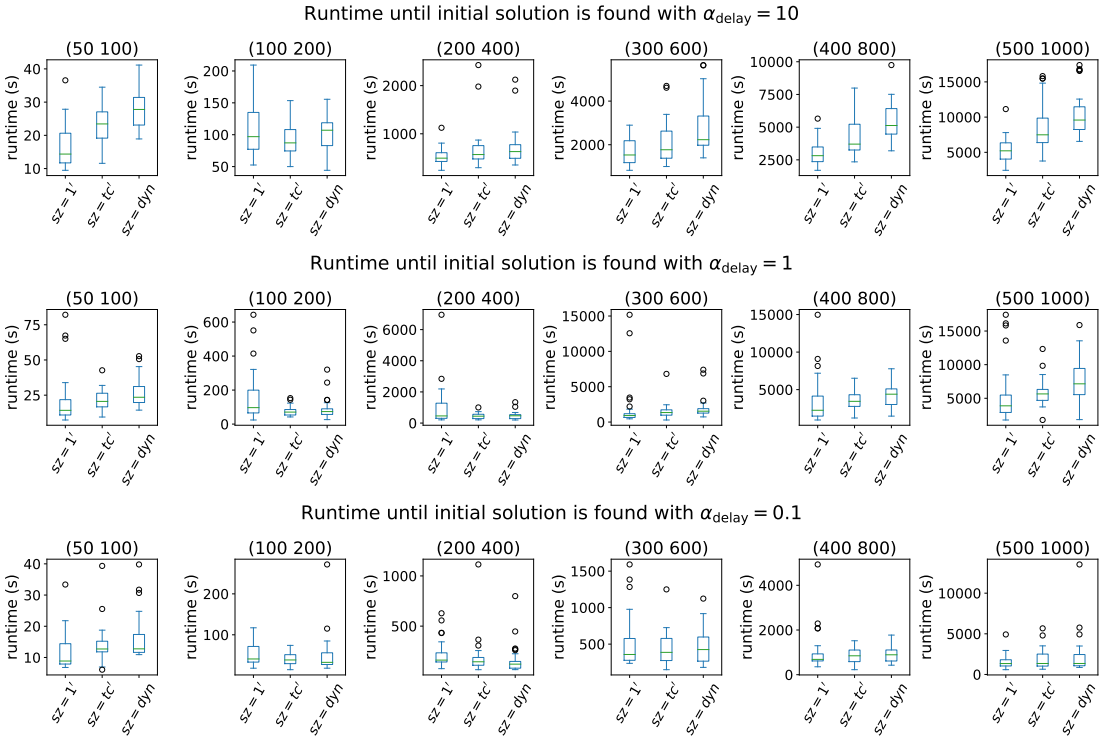


Figure 7.13: Run time until the initial solution is found for $\alpha_{\text{delay}} = 10$, $\alpha_{\text{delay}} = 1$, and $\alpha_{\text{delay}} = 0.1$ as well as the various configurations and instance groups.

Nevertheless, achieving good results regarding the optimality gap for a good construction heuristic is also relevant. Furthermore, we see that *sz=1* performs worse than the other heuristics. Additionally, as was already the case for the MILP results, it can be noted that solving instances with the parameter α_{delay} set to 10 leads to the most significant gaps. This, of course, is to be expected since all previous results indicate the same behavior; both the run time and the optimality increase in this configuration. A key takeaway is that all three of them compare well against the normal MILP. For size *(300 600)*, we can already see that all the construction heuristics outperform the MILP in every configuration. Especially the variants *sz=tc* and *sz=dyn* achieve remarkable results. Naturally, these results come at a cost. Both take quite some time longer than *sz=1*. The variants *sz=tc* and *sz=dyn* achieve similar results. Judging by performance alone, there is no clear winner between those two. However, considering the time it takes to find an initial solution, we can say that using *sz=tc* over *sz=dyn* in all cases is probably advisable. Nevertheless, this does not hold true for the comparison between *sz=tc* and *sz=1*. In this case, it depends on the performance of the destroy and repair part of the heuristic as well as the available time.

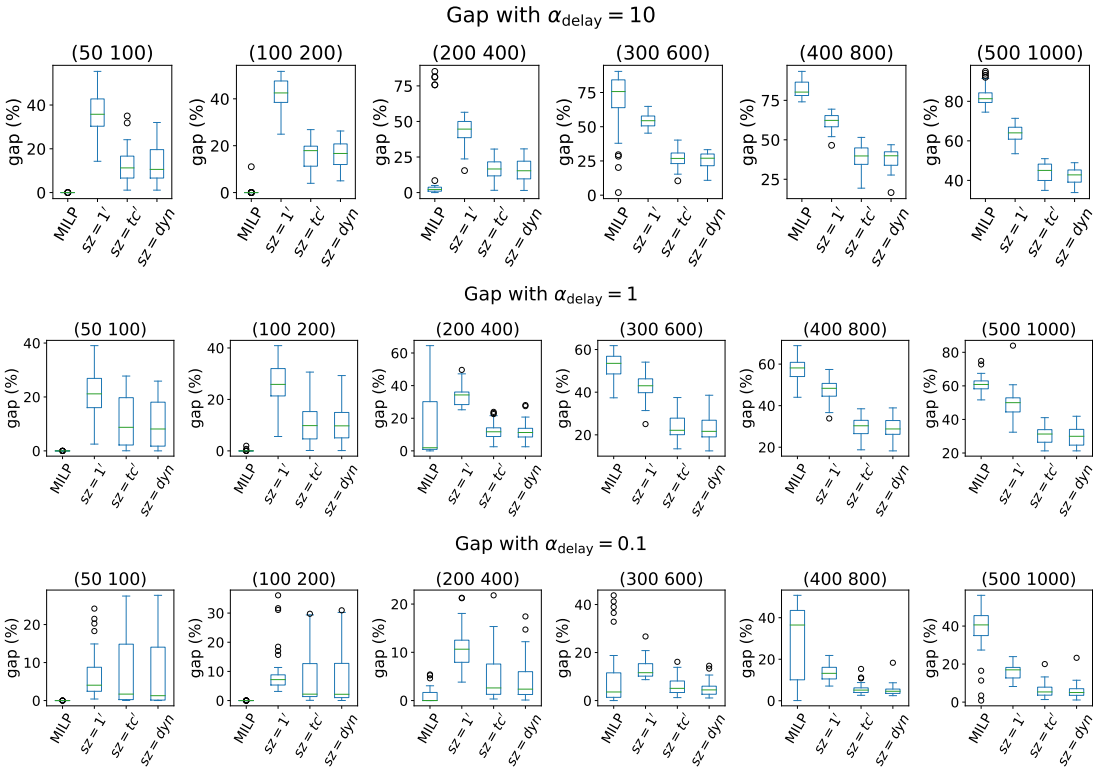


Figure 7.14: Achieved optimality gaps of the initial solution for $\alpha_{\text{delay}} = 10$, $\alpha_{\text{delay}} = 1$, and $\alpha_{\text{delay}} = 0.1$ as well as the various configurations and instance groups.

Destroy Operators

In this chapter, we look at the different destroy operators, namely *rloc*, *rloc'*, *wloc*, and *wloc'*. Each operation destroys 20% of the existing stations, and in the case of *rloc'* and *wloc'*, an additional 10% of the existing a variables will be destroyed. We compare the achieved gaps to each other and the initial starting value. For this, we use the result of the worst (regarding optimality) construction heuristic *sz=1* and always use *sz=tc* as a repair operation. In order to enable a fair comparison between the various destroy operators, we always use the same starting solution (generated by the construction heuristic) for each instance. We collect the results for each destroy operation after two hours of run time. This is approximately the time left for the destroy & repair procedure to improve the result for the biggest instances.

The first thing worth mentioning is that all destroy operations use very little time of the fixed time limit, as seen in Table 7.7. Depending on the operator and the instance group, results vary from less than 0.001 to 1.383 seconds for one destroy operation. The fastest one, an average being *rloc* followed by *wloc*. The chosen α_{delay} value has minimal impact. The time spent destroying increases with larger instance groups and higher

α_{delay} values. Interestingly, we can observe that *wloc* achieves more iterations than *rloc*. This is because, on average, fewer a variables are destroyed with the *wloc* operator than the *rloc* one. Usually, less optimal stations typically do have less demand assigned to them. Therefore, when we remove a less optimal station, we typically remove fewer a variables, making it easier to reconstruct the solution. More detailed results can be seen in Table 7.7.

Table 7.7: The average time spent destroying (called atsd, which is the average time it took in a single iteration for the destroy part in seconds) and the number of iterations (iter) for all destroy operators for $\alpha_{\text{delay}} = 10$, $\alpha_{\text{delay}} = 1$ and $\alpha_{\text{delay}} = 0.1$ are shown.

Results for $\alpha_{\text{delay}} = 10$

	rloc		*rloc′*		*wloc*		*wloc′*	
(n, m)	atsd	iter	atsd	iter	atsd	iter	atsd	iter
(50 100)	0.000	2,579.9	0.007	1,843.4	0.003	2,453.3	0.010	1,704.9
(100 200)	0.001	982.1	0.020	939.9	0.010	799.8	0.030	979.7
(200 400)	0.010	94.8	0.120	66.5	0.060	96.6	0.174	71.5
(300 600)	0.022	41.6	0.276	29.1	0.136	45.0	0.433	28.5
(400 800)	0.039	22.2	0.580	13.7	0.269	23.2	0.883	13.7
(500 1000)	0.062	14.4	0.931	8.0	0.385	15.8	1.272	9.3

Results for $\alpha_{\text{delay}} = 1$

	rloc		*rloc′*		*wloc*		*wloc′*	
(n, m)	atsd	iter	atsd	iter	atsd	iter	atsd	iter
(50 100)	0.000	2,835.2	0.007	2,042.4	0.003	1,973.4	0.009	1,709.4
(100 200)	0.001	774.2	0.022	605.5	0.010	593.7	0.028	1,065.9
(200 400)	0.008	118.8	0.122	79.6	0.053	99.6	0.173	65.4
(300 600)	0.015	56.7	0.268	36.8	0.115	45.0	0.442	24.8
(400 800)	0.029	30.4	0.487	19.5	0.185	28.3	0.784	14.3
(500 1000)	0.051	17.2	0.818	11.8	0.317	17.4	1.383	8.9

Results for $\alpha_{\text{delay}} = 0.1$

	rloc		*rloc′*		*wloc*		*wloc′*	
(n, m)	atsd	iter	atsd	iter	atsd	iter	atsd	iter
(50 100)	0.000	3,453.0	0.006	2,926.4	0.002	2,849.2	0.007	2,543.1
(100 200)	0.001	953.9	0.021	727.1	0.007	1,477.1	0.021	2,054.2
(200 400)	0.006	171.4	0.113	106.5	0.045	122.9	0.135	105.9
(300 600)	0.013	67.3	0.246	51.7	0.088	60.7	0.336	43.2
(400 800)	0.024	37.5	0.489	29.1	0.194	29.0	0.621	26.2
(500 1000)	0.041	21.2	0.830	17.3	0.296	18.3	1.166	14.3

Regarding optimality, destroying additional demand positively affects the performance of

the destroy and repair phase. We also noticed that it significantly affects whether the 10% additional demand is random a variables or if we choose to destroy them by our weighted tournament selection. Only destroying the stations has the main disadvantage in that the solutions are more likely to be reconstructed the same as before. As was already mentioned, the solution created by the construction heuristic is already an improvement to the MILP solution for the bigger instance sizes. The destroy and (re)construction phase further improves the solution.

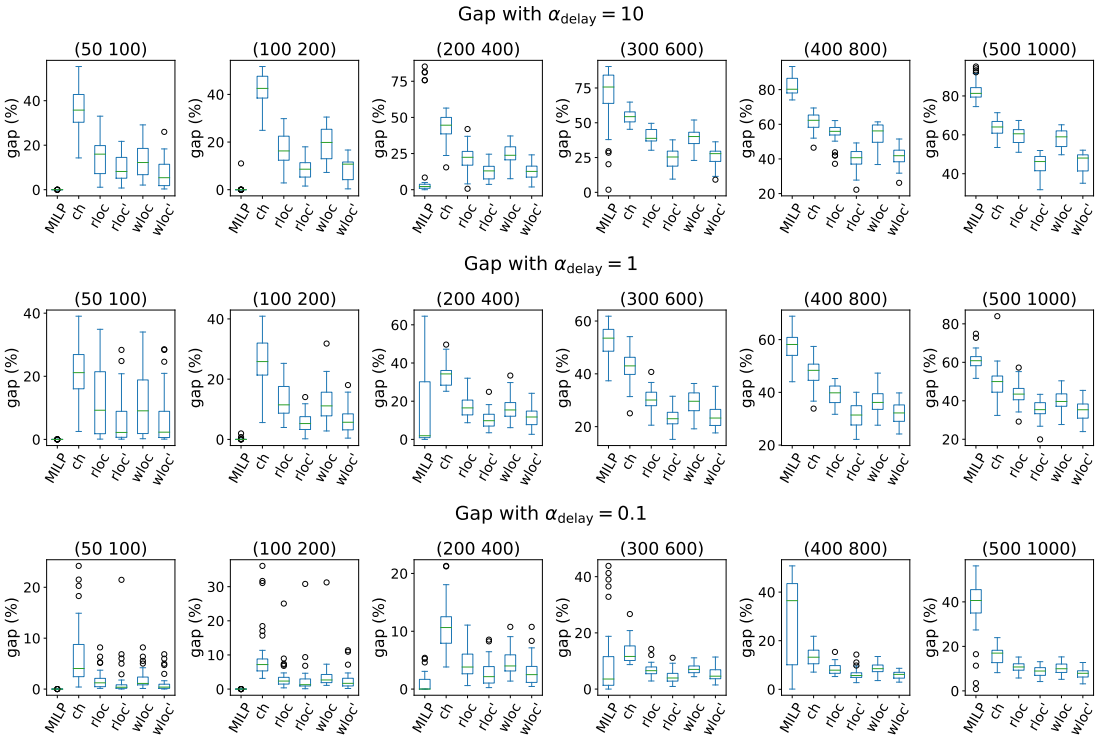


Figure 7.15: This figure shows the gap (%) of all destroy operators as well as the MILP solution and the starting value determined by the construction heuristic (ch) for $\alpha_{\text{delay}} = 10$, $\alpha_{\text{delay}} = 1$ and $\alpha_{\text{delay}} = 0.1$

However, results for *rloc* and *wloc* were expected to be more compelling. In two hours and on average 14.4 (for $n = 500$, $m = 1000$ and $\alpha_{\text{delay}} = 10$) iterations *rloc* only achieves an improvement from an average 63.4% gap to 59.5%. This is especially surprising considering that we started with the worst construction heuristic (to leave enough room for improvement to compare the destroy operators) and used a better one for reconstruction. The same also holds for *wloc*. The results for *rloc′* and *wloc′* are significantly better. Nevertheless, the number of iterations achieved in two hours is underwhelming due to the increased number of destroyed variables. 7.9 iterations on average for *rloc′* when using $\alpha_{\text{delay}} = 10$ and the largest instance group *(500 1000)* achieve a gap improvement from 65.6% to 50.0%. When comparing the random version with their weighted random counterparts, it is hard to determine a winner. The weighted

random has a slim advantage that is not significant. More detailed results for all instance groups and α_{delay} values can be found in Table 7.15.

Repair Operators

In this chapter, we look at the different repair operators, namely *sz=1*, *sz=tc*, and *sz=dyn*. We compare the achieved gaps to each other and the initial starting value. For this, we use the result of the worst (regarding optimality) construction heuristic *sz=1* and always use *rloc′* as a destroy operation. As with the destroy operators, all configurations use the same solution as a starting point created in an extra run. Again, the collected results for each repair operation after two hours of run time can be seen in Table 7.8.

The results are similar to those of the construction heuristic. Again, *sz=tc* and *sz=dyn* perform very similarly. It is very close, but in most cases, *sz=dyn* performs slightly better than *sz=tc*. This may be because it achieved as many iterations as *sz=tc*. One reason for this is the easier-to-solve MILP program since most of the solution remains intact, which may also cause the bad performance of *sz=1*, which achieves fewer iterations than both *sz=tc* and *sz=dyn*. This, to us, is very surprising. Again, as was already the case in all other results, $\alpha_{\text{delay}} = 0.1$ is the easiest of the configurations. The gap for this configuration is consistently lower, while the number of iterations achieved is higher.

7.2.2 Final Results for the Iterated Greedy Heuristic

We used the versions displayed in Table 7.9 of our iterated greedy heuristic for our final test. The table includes each tested configuration's construction heuristic, repair, and destroy operator.

Each of the configurations was tested for four hours. Configurations that use the same construction heuristic use the same initial solution as a starting point. The initial solution was created in an extra run. In contrast to our tests for the destroy & repair operators, we use the time it took to create the initial solution and subtract it from the total available time. Because our operators use an element of randomness, this procedure ensures that the randomness is limited and affects our results as little as possible. It also guarantees that the achieved results reflect the performance of each configuration.

Results for the Artificial Instance Set

Table 7.10 shows the achieved results of all tested configurations. Both test configurations using *sz=dyn* outperformed the *sz=tc* versions in all different settings. The final test results start outperforming the MILP at instance size *(200 400)* in the case of $\alpha_{\text{delay}} = 10$ and $\alpha_{\text{delay}} = 1$ and *(300 600)* if the chosen α_{delay} parameter is 0.1. In the case of *(500 1000) IG-v2* was able to outperform the MILP by 40% with the other test configurations not being too far behind. Again, a smaller α_{delay} value leads to better gaps. In the case of $\alpha_{\text{delay}} = 0.1$, the best result achieved has an average gap of only 4.6%. Gaps

Table 7.8: The optimality gap and the number of iterations (iter) for all repair operators for $\alpha_{\text{delay}} = 10$, $\alpha_{\text{delay}} = 1$ and $\alpha_{\text{delay}} = 0.1$ are shown.

Results for $\alpha_{\text{delay}} = 10$

	milp	const	*sz=1*		*sz=tc*		*sz=dyn*	
(n, m)	gap	gap	gap	iter	gap	iter	gap	iter
(50 100)	0.0%	37.3%	10.1%	1,405.0	5.9%	1,889.6	5.6%	1,433.0
(100 200)	0.7%	41.1%	20.9%	555.9	9.6%	1,338.9	6.3%	265.3
(200 400)	11.4%	44.8%	24.6%	61.1	11.9%	72.3	11.0%	63.7
(300 600)	71.1%	55.7%	40.5%	22.4	25.3%	26.7	24.6%	22.8
(400 800)	82.2%	61.6%	50.1%	11.3	39.5%	13.3	38.6%	11.3
(500 1000)	82.7%	63.4%	57.0%	7.6	46.5%	8.0	44.1%	7.4

Results for $\alpha_{\text{delay}} = 1$

	milp	const	*sz=1*		*sz=tc*		*sz=dyn*	
(n, m)	gap	gap	gap	iter	gap	iter	gap	iter
(50 100)	0.0%	19.3%	5.4%	1,652.5	7.1%	2,131.9	6.9%	1,968.9
(100 200)	0.1%	26.6%	11.3%	574.4	6.1%	734.5	6.1%	773.2
(200 400)	11.1%	32.8%	20.6%	50.5	10.2%	82.3	10.2%	93.7
(300 600)	52.5%	43.1%	32.7%	26.6	23.7%	40.1	25.2%	33.1
(400 800)	57.8%	46.1%	40.3%	12.6	32.6%	19.0	33.0%	17.0
(500 1000)	61.4%	49.1%	42.9%	9.1	36.5%	11.7	36.8%	9.8

Results for $\alpha_{\text{delay}} = 0.1$

	milp	const	*sz=1*		*sz=tc*		*sz=dyn*	
(n, m)	gap	gap	gap	iter	gap	iter	gap	iter
(50 100)	0.0%	4.7%	0.8%	1,778.3	0.9%	2,795.3	0.9%	2,972.4
(100 200)	0.0%	9.8%	2.6%	627.1	2.3%	471.1	1.9%	856.2
(200 400)	0.4%	9.6%	4.4%	79.7	1.6%	111.4	1.9%	140.2
(300 600)	9.6%	13.5%	8.0%	37.1	4.7%	52.3	4.5%	62.5
(400 800)	29.8%	12.7%	9.3%	17.5	6.4%	28.7	6.8%	34.1
(500 1000)	39.2%	15.7%	12.3%	12.2	8.9%	17.3	8.6%	19.4

Table 7.9: Tested configurations for both the Honda and the AIS instances.

Configuration	Construction Heuristic	Repair Operator	Destroy Operator
IG-v1	*sz=tc*	*sz=tc*	$wloc'$
IG-v2	*sz=tc*	*sz=tc*	$rloc'$
IG-v3	*sz=dyn*	*sz=dyn*	$wloc'$
IG-v4	*sz=dyn*	*sz=dyn*	$rloc'$

Table 7.10: The achieved gaps (%) for all test configurations for $\alpha_{\text{delay}} = 10$, $\alpha_{\text{delay}} = 1$ and $\alpha_{\text{delay}} = 0.1$ are shown.

Results for $\alpha_{\text{delay}} = 10$

(n, m)	MILP	*IG-v1*	*IG-v2*	*IG-v3*	*IG-v4*
(50 100)	0.0	10.4	9.1	6.5	6.7
(100 200)	0.4	8.2	8.2	6.5	5.7
(200 400)	15.0	11.6	12.6	10.8	11.1
(300 600)	67.6	21.7	22.3	20.4	21.5
(400 800)	81.8	36.8	37.1	35.3	35.3
(500 1000)	82.8	43.7	43.3	41.6	41.4

Results for $\alpha_{\text{delay}} = 1$

(n, m)	MILP	*IG-v1*	*IG-v2*	*IG-v3*	*IG-v4*
(50 100)	0.0	4.5	4.8	4.4	5.3
(100 200)	0.1	7.1	7.4	6.0	6.0
(200 400)	14.1	8.3	8.7	8.6	8.6
(300 600)	52.5	19.4	20.2	21.0	20.4
(400 800)	57.4	26.9	27.3	27.3	27.2
(500 1000)	61.2	30.8	30.8	28.8	29.3

Results for $\alpha_{\text{delay}} = 0.1$

(n, m)	MILP	*IG-v1*	*IG-v2*	*IG-v3*	*IG-v4*
(50 100)	0.0	2.4	3.4	1.7	1.6
(100 200)	0.0	2.2	2.3	1.9	2.5
(200 400)	1.1	2.2	2.4	2.2	2.1
(300 600)	10.1	3.9	3.8	3.6	3.5
(400 800)	29.4	3.9	3.7	3.6	3.5
(500 1000)	37.3	6.1	5.9	4.6	4.7

with a $\alpha_{\text{delay}} = 10$ are much more significant. However, since the optimal solution is unknown and we only work with the lower bound, this may be because the best found lower bound, which is used in the calculation of the gaps with $\alpha_{\text{delay}} = 10$ is poorer than with $\alpha_{\text{delay}} = 0.1$.

A reason for the performance of the *sz=tc* test configurations can be found when we look at the achieved iterations. In Table 7.11, we can see that in most instance groups, *sz=tc* has about the same number of iterations as *sz=dyn*. However, the performance in the biggest set *(500 1000)* is surprisingly poor. With $\alpha_{\text{delay}} = 10$ we only achieve two iterations with *sz=tc* while we achieve 6 with *sz=dyn* on the median. However, in the second largest instance group *(400 800) sz=tc* outperforms *sz=dyn* with 22 compared to 17 iterations. This is certainly why *sz=dyn* outshines *sz=tc* that drastically in the

Table 7.11: The achieved number of iterations for all test configurations for $\alpha_{\text{delay}} = 10$, $\alpha_{\text{delay}} = 1$ and $\alpha_{\text{delay}} = 0.1$ are shown.

Results for $\alpha_{\text{delay}} = 10$

(n, m)	*IG-v1*	*IG-v2*	*IG-v3*	*IG-v4*
(50 100)	3,304	4,104	4,578	2,691
(100 200)	425	428	408	419
(200 400)	111	112	129	114
(300 600)	52	46	42	40
(400 800)	22	22	17	16
(500 1000)	2	2	6	6

Results for $\alpha_{\text{delay}} = 1$

(n, m)	*IG-v1*	*IG-v2*	*IG-v3*	*IG-v4*
(50 100)	3,777	3,620	5,135	2,892
(100 200)	410	334	440	396
(200 400)	140	106	161	112
(300 600)	67	44	64	42
(400 800)	31	22	29	20
(500 1000)	2	2	13	10

Results for $\alpha_{\text{delay}} = 0.1$

(n, m)	*IG-v1*	*IG-v2*	*IG-v3*	*IG-v4*
(50 100)	4,976	5,438	7,550	5,112
(100 200)	765	809	902	686
(200 400)	207	183	261	207
(300 600)	96	76	102	82
(400 800)	52	43	56	46
(500 1000)	2	2	33	30

biggest instance set. Another interesting observation is that the number of iterations does not increase with smaller α_{delay} values for *sz=tc* while for *sz=dyn* it does increase.

Lastly, in Figure 7.16, we display the improvement of the destroy and repair heuristics of three instances of the (300 600) instance group using $\alpha_{\text{delay}} = 10$, $\alpha_{\text{delay}} = 1$ and $\alpha_{\text{delay}} = 0.1$. We used *sz=dyn* as construction and repair heuristic. With a weight of $\alpha_{\text{delay}} = 10$, we see constant improvements in the quality of the solution. While using $\alpha_{\text{delay}} = 0.1$, fewer solutions register as better solutions. We can also see that with a decrease of the α_{delay} value, the number of iterations increases, and the quality of the solution increases (smaller gaps). In all cases, we will likely see further improvements to the solution if we achieved more iterations. This is especially true for $\alpha_{\text{delay}} = 10$.

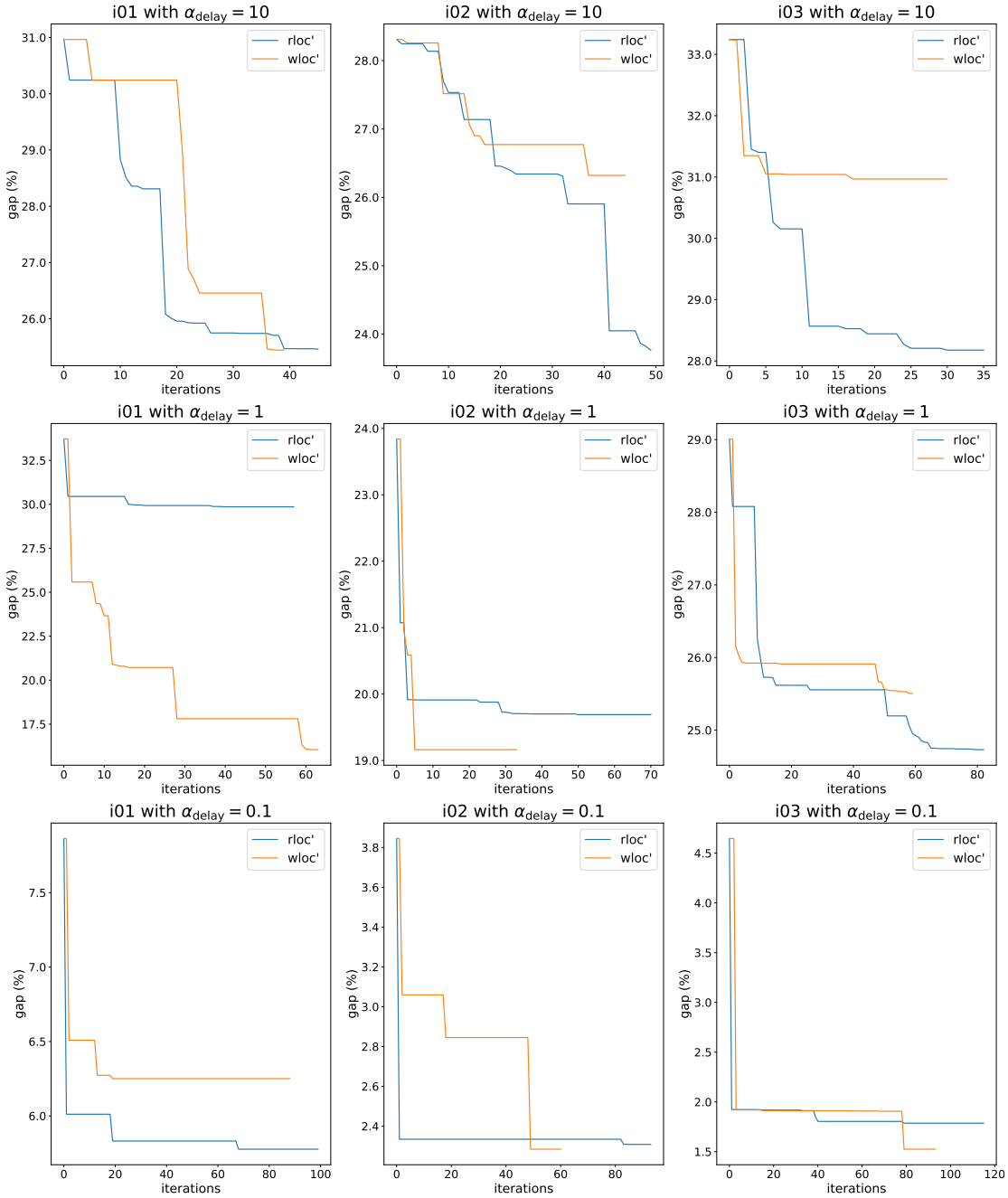


Figure 7.16: Comparison of how solutions are iteratively improved. Displayed are the first three instances (i01, i02, i03) of the (300 600) instance set with $\alpha_{\text{delay}} = 10$, $\alpha_{\text{delay}} = 1$ and $\alpha_{\text{delay}} = 0.1$. As construction and repair heuristic *sz=dyn* was used in all cases.

7.2.3 Results for the Honda Instance Set

In this section, we look at the results of the iterated greedy heuristic achieved on the various instances of the Honda instance set. We ran the same configurations already used for the BEXSLP instances, as seen in Table 7.9.

Table 7.12: Average gap (%) for all tested configurations for case1, case2 and case3 with all its different weightings. The MILP solution, the initially found solution of the construction heuristic (*sz=tc* ch and *sz=dyn* ch respectively), and the final solution are displayed.

	MILP	*sz=tc*			*sz=dyn*		
instance	MILP	*sz=tc* ch	*IG-v1*	*IG-v2*	*sz=dyn* ch	*IG-v3*	*IG-v4*
case1	0.0	23.8	0.2	0.1	23.7	0.2	0.1
case2	0.0	10.9	9.8	8.6	9.7	8.3	8.5
case3	8.0	24.0	8.0	8.0	22.2	8.0	8.0
$case3_{10}$	7.1	21.3	7.3	7.4	21.3	7.4	7.4
$case3_{100}$	0.0	13.7	1.8	2.0	13.6	2.4	2.5
$case3_{200}$	0.0	13.1	3.3	3.4	12.8	3.8	3.6
$case3_{500}$	0.0	12.0	0.9	0.8	11.4	0.8	4.7
$case3_{1000}$	0.0	11.7	7.3	6.0	11.6	6.5	6.1
$case3_{2000}$	0.0	15.8	7.9	5.4	13.3	4.3	6.5
$case3_{5000}$	0.0	18.7	8.3	6.0	17.6	7.1	6.9
$case3_{10000}$	0.0	12.5	8.2	9.6	11.1	8.4	8.3

Table 7.13: Number of iterations achieved for all tested configurations for case1, case2 and case3 with all its different weightings.

instance	*IG-v1*	*IG-v2*	*IG-v3*	*IG-v4*
case1	1199	1128	598	795
case2	11798	10032	11577	11038
case3	7090	6086	8192	10055
$case3_{10}$	20048	6931	8219	7647
$case3_{100}$	8917	8726	9359	7263
$case3_{200}$	9302	9259	10580	10072
$case3_{500}$	9414	9374	9133	13890
$case3_{1000}$	14754	7401	19727	17861
$case3_{2000}$	12563	13548	14317	33548
$case3_{5000}$	11777	14426	11208	11685
$case3_{10000}$	12304	26771	11532	20588

All final gaps are below 10%. In fact, all configurations find excellent solutions in all variations. Interestingly, increasing the α_{delay} value does not affect the quality of the

solution. The variant with α_{delay} times one is the closest to the MILP solution in both configurations. However, it affects the number of iterations of the various configurations achieved. The base variant of case3 ($\alpha_{\text{delay}} = 1$) completed between 6086 iterations and 10055 iterations. The number of iterations increases with increasing α_{delay} values. Interestingly, all configurations achieve their respective maximum of iterations at different α_{delay} weights. 20048 iterations are the maximum number of iterations achieved for *IG-v1* when using α_{delay} of 10. *IG-v2* achieved the most iterations with α_{delay} of 10000 while *IG-v3* achieved the maximum number of iterations with α_{delay} of 1000. Finally, *IG-v4* peaked at 33548 iterations when using α_{delay} of 2000. More details regarding the number of iterations for different instances and weightings can be found in Table 7.13.

Table 7.14: Average gap (%) for all tested configurations for case3 street network (sn) with all its different weightings. The MILP solution, the initially found solution of the construction heuristic (*sz=tc* ch and *sz=dyn* ch respectively), and the final solution are displayed.

	MILP	*sz=tc*			*sz=dyn*		
instance	MILP	*sz=tc* ch	*IG-v1*	*IG-v2*	*sz=dyn* ch	*IG-v3*	*IG-v4*
*case*3_*sn*	0.0	30.5	0.1	0.1	29.5	0.1	0.1
*case*3_sn_{10}	0.0	26.6	0.8	1.0	26.6	0.9	0.9
*case*3_sn_{100}	0.0	17.6	4.2	3.6	17.6	4.9	3.8
*case*3_sn_{200}	0.0	5.3	4.3	4.4	5.2	4.6	4.8
*case*3_sn_{500}	0.0	6.6	3.9	4.0	8.8	4.5	4.6
*case*3_sn_{1000}	0.0	11.8	3.1	2.1	10.7	2.7	2.4
*case*3_sn_{2000}	0.0	4.1	1.4	4.1	4.7	0.5	0.6
*case*3_sn_{5000}	0.0	4.9	2.3	4.9	4.9	4.9	1.8
*case*3_sn_{10000}	0.0	2.6	2.6	2.6	2.6	2.6	2.6

When examining Table 7.14, we take notice of the fact that the gaps for the street network variation of case3 lead to smaller gaps compared to the normal version. We already paid attention to the fact that the street network variation appears to be easier to solve when testing the street network variation using the MILP. Interestingly, the number of iterations, seen in Table 7.15, decrease dramatically in the street network variation. Again, it can be observed that the number of achieved iterations is lowest in the base variation (with $\alpha_{\text{delay}} = 1$). However, as was already the case for the normal variants of case3, there is no recognizable pattern.

Additionally, there is no single configuration that objectively performed best. In the street network variation of case3 *IG-v4* performs best overall, followed by *IG-v1*. In the normal variation of the cases, the other variants perform best. Due to the randomness of the repair operators, a single test on a single case does not suffice to provide a complete picture. However, all tests have in common that the repair & destroy part performs

Table 7.15: Number of iterations achieved for all tested configurations for case3 street network (sn) with all its different weightings.

instance	*IG-v1*	*IG-v2*	*IG-v3*	*IG-v4*
case3_sn	5200	902	1954	990
$case3\_sn_{10}$	2791	1550	2740	1637
$case3\_sn_{100}$	4303	2956	4446	4805
$case3\_sn_{200}$	3370	2787	3558	5226
$case3\_sn_{500}$	4745	2221	3924	1233
$case3\_sn_{1000}$	1769	3530	2786	3363
$case3\_sn_{2000}$	4414	4302	2712	4042
$case3\_sn_{5000}$	4671	15	5304	486
$case3\_sn_{10000}$	731	4401	1145	1843

better than in the AIS. Gaps of 20%-30% can be reduced to below 1%.

Furthermore, the higher the α_{delay}, the better the initial solution of the construction heuristic. This holds true for both the street network and the regular variation. Interestingly, the opposite is true for the repair & destroy procedure. With larger α_{delay} values, the construction heuristic can only be improved by small amounts or not at all. Some of that can be explained by considering that for $\alpha_{\text{delay}} = 1$, most of the total cost comes from the setup cost. Table 7.6 demonstrates that with $\alpha_{\text{delay}} = 10000$, only a fraction of the total cost is setup cost. This is because the number of modules is limited. Therefore, the problem is reduced to finding the optimal configuration of modules and stations regarding the delay.

CHAPTER 8

Conclusion and Future Work

This thesis considered the Battery Exchange Station Location Problem (BEXSLP2), which aims to provide a guideline for planning the best battery-swapping stations concerning setup cost, charging cost, and induced delay for customers. Such stations allow customers to swap their empty batteries with fully charged ones. All demand created by customers needs to be satisfied. The BEXSLP is then formulated as a mixed integer linear programming (MILP) program, which minimizes the objectives mentioned earlier. These objectives can also be weighted differently to allow different interest groups to use the formulation.

We evaluated the MILP formulation of the BEXSLP on two different sets of instances. The first instance set includes three instances derived from Honda's data. In comparison, the second one contained 180 instances inspired by the Honda instances but generated artificially. The largest instances proved too hard to solve for an exact approach. We achieved better solutions than the original BEXSLP in less time. Most of the instances provided by Honda were easily solvable in the four-hour time frame. The hardest part when solving the BEXSLP2 using the MILP solver was proving the given solution's optimality.

Therefore, we also proposed and implemented a matheuristic. A matheuristic combines the exact solving capabilities of MILP solvers with the scalability of a heuristic approach. In our case, we decided to use an iterated greedy heuristic. In this procedure, we construct a solution by selecting a subset of intervals and only solve the MILP for the subset. We then consider the next subset of intervals until all intervals have been considered. After the construction of the initial solution, we then destroy parts of the solution and use our construction heuristic as a means to repair the solution. We developed different destroy and repair operators. When evaluating those operators, we discovered that the construction heuristic beat the MILP solutions for the largest instance sizes. We also

noticed that only destroying stations and their associated demand does not lead to good quality solutions. Consequently, we also destroy random demand, allowing more flexibility when repairing the solution. We noticed that considering a full charging cycle yields the best results for the repair operators. The iterated greedy approach improved the quality of the solution tremendously for the largest instance sizes. However, the improvement of the destroy and repair operators could have been more impressive.

We also evaluated the instance sets with different weightings of the objective function. Generally speaking, increasing the delay leads to more significant gaps. The combination of minimizing the delay while at the same time being restricted by the number of modules that can be built is the most challenging task when solving the MILP. When comparing the BEXSLP with BEXSLP2, we improved the results significantly. We also exposed weaknesses in the BEXSLP when testing for different delay weightings. The BEXSLP was not able to adapt appropriately to these changing values.

8.1 Future Work

As mentioned, we were unsatisfied with the results achieved by the destroy & repair part of our iterated greedy algorithm. This was partly because we achieved fewer iterations than we hoped for the largest instance sizes. Additionally, it could also be advantageous to increase the achieved variation in the iterations and consequently reduce the number of identical solutions (identical with regards to the opened stations and modules built). We discussed the possibility of blacklisting specific solutions as well as the possibility of mutating the solution and letting the MILP only assign the demand. However, we ultimately decided against it because both ideas do not work well with our iterated greedy procedure. Therefore, using a genetic algorithm (GA) instead of the iterated greedy heuristic may be an interesting topic to explore.

Another potential area of research is to vary all existing parameters. This includes the parameters to weigh the objective function and some assumptions made while working on this problem. One possibility is to increase or decrease the length of a single time interval which in our case was one hour. Another interesting starting point is to no longer serve all customer demand but instead only satisfy a certain percentage or have a function determining if users are willing to travel the distance to the battery-swapping station or not. A further starting point for follow-up research may be to no longer have the customers managed and optimized by the MILP but instead take a more realistic approach of customers riding to one of the closest stations. Consequently, this may lead to some stations being overcrowded, which in turn means that these stations may no longer be able to fulfill the needs of all their customers.

Finally, we are most interested in determining how well our developed algorithms would hold up in a real-world scenario. This implies verifying whether the assumptions made

in this work still hold in such a scenario, comparing our algorithms to other works, and testing which provides the best results regarding user experience and customer participation rate.

List of Figures

List of Tables

List of Algorithms

Bibliography

[BBM08] Roberto Battiti, Mauro Brunato, and Franco Mascia. *Reactive search and intelligent optimization*, volume 45. Springer Science & Business Media, 2008.

[BF12] Alireza Boloori Arabani and Reza Zanjirani Farahani. Facility location dynamics: An overview of classifications and applications. *Computers & Industrial Engineering*, 62(1):408–420, 2012.

[BGM16] Mouna Kchaou Boujelben, Céline Gicquel, and Michel Minoux. A milp model and heuristic approach for facility location under multiple operational constraints. *Computers & Industrial Engineering*, 98:446–461, 2016.

[BMRBR09] Marco A. Boschetti, Vittorio Maniezzo, Matteo Roffilli, and Antonio Bolufé Röhler. Matheuristics: Optimization, simulation and control. In María J. Blesa, Christian Blum, Luca Di Gaspero, Andrea Roli, Michael Sampels, and Andrea Schaerf, editors, *Hybrid Metaheuristics*, pages 171–177, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[BT97] Dimitris Bertsimas and John N. Tsitsiklis. *Introduction to linear optimization*, volume 6. Athena Scientific Belmont, MA, 1997.

[BW05] Dimitris Bertsimas and Robert Weismantel. *Optimization over integers*, volume 13. Dynamic Ideas Belmont, 2005.

[CAT22] Ankit Chouksey, Anil Kumar Agrawal, and Ajinkya N. Tanksale. A hierarchical capacitated facility location-allocation model for planning maternal healthcare facilities in india. *Computers & Industrial Engineering*, 167:107991, 2022.

[Dan90] George B. Dantzig. Origins of the simplex method. In *A history of scientific computing*, pages 141–151. 1990.

[DT03] George Bernard Dantzig and Mukund N. Thapa. *Linear programming: Theory and extensions*, volume 2. Springer, 2003.

[ECMA17] Dimitrios Efthymiou, Katerina Chrysostomou, Maria Morfoulaki, and Georgia Aifantopoulou. Electric vehicles charging infrastructure location: a genetic algorithm approach. *European Transport Research Review*, 9(2):1–9, 2017.

[Gok20] Osman Gokalp. An iterated greedy algorithm for the obnoxious p-median problem. *Engineering Applications of Artificial Intelligence*, 92:103674, 2020.

[GYL18] Fang Guo, Jun Yang, and Jianyi Lu. The battery charging station location problem: Impact of users' range anxiety and distance convenience. *Transportation Research Part E: Logistics and Transportation Review*, 114:1–18, 2018.

[JCvEN21] Helia Jamshidi, Gonçalo HA Correia, J Theresia van Essen, and Klaus Nökel. Dynamic planning for simultaneous recharging and relocation of shared electric taxies: A sequential milp approach. *Transportation Research Part C: Emerging Technologies*, 125:102933, 2021.

[JORR20] Thomas Jatschka, Fabio F. Oberweger, Tobias Rodemann, and Günther R. Raidl. Distributing battery swapping stations for electric scooters in an urban area. In Nicholas Olenev, Yuri Evtushenko, Michael Khachay, and Vlasta Malkova, editors, *Optimization and Applications*, volume 12422 of *LNCS*, pages 150–165. Springer, 2020.

[Kar84] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311, 1984.

[Kha79] Leonid Genrikhovich Khachiyan. A polynomial algorithm in linear programming. In *Doklady Akademii Nauk*, volume 244, pages 1093–1096. Russian Academy of Sciences, 1979.

[LGL19] Hao Liu, Benhe Gao, and Yunhan Liu. Battery swap station location routing problem with capacitated electric vehicles and time windows. In *2019 IEEE 6th International Conference on Industrial Engineering and Applications (ICIEA)*, pages 832–836, 2019.

[LLKL22] Min-Der Lin, Ping-Yu Liu, Jia-Hong Kuo, and Yu-Hao Lin. A multiobjective stochastic location-allocation model for scooter battery swapping stations. *Sustainable Energy Technologies and Assessments*, 52:102079, 2022.

[LLYL21] Min-Der Lin, Ping-Yu Liu, Ming-Der Yang, and Yu-Hao Lin. Optimized allocation of scooter battery swapping station under demand uncertainty. *Sustainable Cities and Society*, 71:102963, 2021.

[NSdG19] Stefan Nickel and Francisco Saldanha-da Gama. *Multi-Period Facility Location*, pages 303–326. Springer International Publishing, Cham, 2019.

[PR05] Jakob Puchinger and Günther R. Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In *International work-conference on the interplay between natural and artificial computation*, pages 41–53. Springer, 2005.

[Rau22] Matthias Rauscher. A matheuristic for battery exchange station location planning for electric scooters, 2022.

[RS07] Rubén Ruiz and Thomas Stützle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049, 2007.

[RSS20] Pradeep Rathore, Sarada Prasad Sarmah, and Arti Singh. Location–allocation of bins in urban solid waste management: a case study of bilaspur city, india. *Environment, Development and Sustainability*, 22(4):3309–3331, 2020.

[Sch98] Alexander Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

[SR18] Thomas Stützle and Rubén Ruiz. Iterated greedy. *Handbook of heuristics*, pages 547–577, 2018.

[WN99] Laurence A. Wolsey and George L. Nemhauser. *Integer and combinatorial optimization*, volume 55. John Wiley & Sons, 1999.

[WYW+19] Shouxiang Wang, Lu Yu, Lei Wu, Yichao Dong, and Hongkun Wang. An improved differential evolution algorithm for optimal location of battery swapping stations considering multi-type electric vehicle scale evolution. *IEEE Access*, 7:73020–73035, 2019.

[YLK21] Shangyao Yan, Chih-Kang Lin, and Zong-Qi Kuo. Optimally locating electric scooter battery swapping stations and battery deployment. *Engineering Optimization*, 53(5):754–769, 2021.

[YS15] Jun Yang and Hao Sun. Battery swap station location-routing problem with capacitated electric vehicles. *Computers & Operations Research*, 55:217–232, 2015.